T-Kernel 2.0 Extension 仕様書

2012 年 12 月

T-Engineフォーラム

http://www.t-engine.org/

T-Kernel 2.0 Extension 仕様書 (Ver.2.00.00)

本仕様書に関しては、下記にお問い合わせください。

T-Engine フォーラム事務局
〒141-0031 東京都品川区西五反田2-20-1 第28 興和ビル
YRP ユビキタス・ネットワーキング研究所内
TEL：03-5437-0572 FAX：03-5437-2399
E-mail：office@t-engine.org

注

本仕様書において、POSIX とは、Portable Operating System Interface のことで、以下の規格書によって規定された、いわゆるUNIX系の Operating System Interface を指す。

ISO/IEC/IEEE 9945
Information technology - Portable Operationg System Interface (POSIX)
Base Specifications, Issue 7

また、標準C互換ライブラリの章で言及している標準Cライブラリとは、上記POSIXに加え、以下の規格書で規定されたライブラリ関数のことを指す。

JIS X 3010:2003 (ISO/IEC 9899:1999) プログラム言語C

本仕様は、プログラミングの容易性ならびに移植性を考慮し、POSIX との親和性をある程度持たせた上で、標準Cライブラリを利用しているプログラムも容易に移植できるように、標準Cライブラリの仕様についてもほぼそのまま踏襲している。
このため、本仕様書は、IECの許諾の元に上記規格書から一部の内容を引用して記述している。
ただし、本仕様の基本にある、T-Kernel 2.0 は、POSIX とはまったく異なる性質のOperating Systemであり、本仕様は、その拡張機能である。このため、本仕様は、POSIXとの互換性を保証するものではない。また本仕様にしたがって記述されたC言語プログラムは、必ずしもJIS C規格に準拠したプログラムであることを保証するものでもない。

IEC: (国際電気標準会議) International Electrotechnical Commission
ISO: (国際標準化機構) International Organization for Standardization
JIS: (日本工業規格) Japanese Industrial Standards

本仕様書内の関数宣言、構造体定義、あるいは数値等は、C言語の構文に従って記述している。

目次

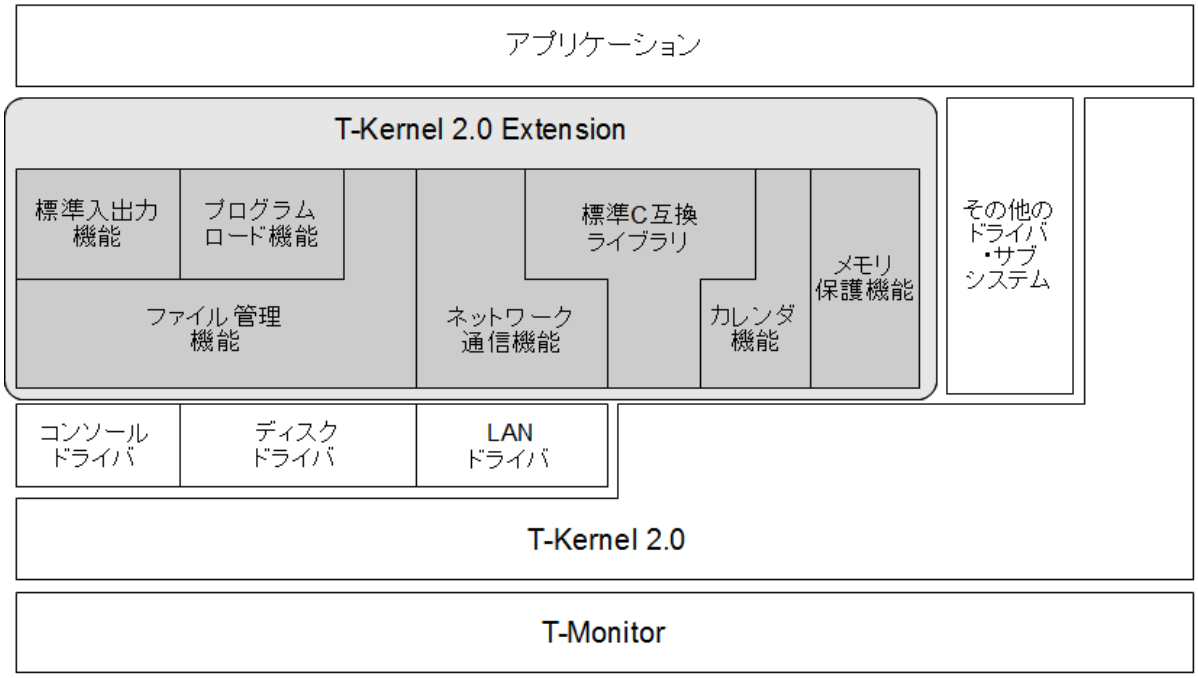
第1章　T-Kernel 2.0 Extension の概要

1.1. 概要

T-Kernel 2.0 Extension (以降 T2EX と記述) は、T-Kernel 2.0 の機能拡張プログラムである。
リアルタイムOSである T-Kernel の軽量性・高速性・リアルタイム性を最大限に活かしつつ高度な組込みシステムを実現するための軽量な拡張機能として設計されている。

T2EX がアプリケーションに対して提供する機能は、拡張SVC(拡張システムコール)、ライブラリ関数、マクロから構成されている。
これらの機能とアプリケーションとのインタフェースを総称して、API(Application Programming Interface)と呼ぶ。
すなわち、T2EX 仕様は T2EX API により規定されている。
APIを構成する個々のシステムコール、ライブラリ関数、マクロの一つ一つについては、APIコールと呼ぶ。
ファイル管理機能のAPIコール全体を指すような場合は、ファイル管理機能APIという呼び方を用いる。

図1.1 に T2EX を含めたソフトウェア・アーキテクチャを示す。
本拡張機能は T-Kernel 2.0 の追加機能(アドオン)として利用できる拡張機能という位置づけとなっており、
T-Kernel 2.0 および T2EX の両方のAPIを用いてアプリケーションプログラムを構築することができる。



(図1.1: T-Kernel 2.0 Extension の構成と位置づけ)

高度な組込みシステムの開発をサポートするため、T2EX では以下の各機能が提供される。

- メモリ保護機能
- ファイル管理機能
- ネットワーク通信機能
- カレンダ機能
- プログラムロード機能
- 標準C互換ライブラリ

提供される機能は機能単位ごとのモジュールとして提供されているため、必要に応じて一部のみを利用し、不要な機能モジュールを利用しない(取り外す)ことも可能である。

T-Engine フォーラムでは、本仕様書とともにT2EXの標準的なコードを、T-Engine リファレンスボード上の T-Kernel 2.0 の上に拡張として実装し、仕様書とともにソースコードを公開する。
この実装をT2EXリファレンス実装と呼び、本仕様書でも実装に依存した項目に関して、T2EXリファレンス実装ではどのように実装されているかを記載している。

1.2. T-Kernel 2.0 Extension の特徴

T2EX では、これまでのT-Kernel 2.0の実力および実績を踏襲し、その軽量性、高速性・リアルタイム性を最大限に発揮させながらも、大規模化・高機能化する組込みシステムにおける機能追加の要求を満たすことを目的として設計されている。以下に T2EX の特徴を示す。

プロセスのない軽量な拡張機能
システム全体を軽量なものとするため、T2EX ではプロセス機能を提供しない。
プロセスの機能は、比較的大規模なシステムのプログラムをモジュール別に開発する際には有効であるが、
リソースを分割した結果としてのプロセス間通信やリソースの切り替えに伴うオーバーヘッドが大きいことから、システム全体の軽量化・高速化とは相反するところがある。そのため、軽量化をめざす T2EX では、プロセスなしでシステム全体を構築することを前提とし、ファイル管理やネットワーク通信を含めた各種の拡張機能についても、すべてプロセスからではなくタスクから直接呼び出して利用可能にする。T-Kernel 2.0 のタスクから拡張機能が直接利用可能であることから、T-Kernel 2.0 のリアルタイム性を活かしつつ高度な機能の実現が可能である。

仮想記憶によらない効率的なメモリ保護機能
一般的なプロセスベースのメモリ保護は多重論理空間をベースとしたものであるが、空間の切り替えに伴う実行上のオーバーヘッドが大きい。T-Kernel 2.0 のタスクベースのプログラムでは、複数のタスク間の情報の交換に変数(メモリ)を介することが多く、単純に論理空間を分けてしまうとプロセス間通信によるオーバーヘッドが大きくなってしまう。軽量化を目指す T2EX では、効率よい実現が可能なシステムレベル・ユーザレベルに基づく2レベルのリング保護を提供する。これによって、本拡張の主なターゲットである特定目的の組込みシステムにおいて、比較的複雑度の高いケースでも必要十分な信頼性を確保する。

モジュール性
T2EX では、ファイル管理機能やネットワーク通信機能を含む多くの機能が提供されるが、これらの各機能はモジュール単位で分割されており、必要なモジュールのみを選択して利用し、それ以外の機能モジュールを取り外して用いることが可能である。
これによって、不要な機能によるRAM・ROMの使用を減らすことが可能である。

標準CやPOSIX仕様への親和性の向上
C言語の標準入出力や、ネットワーク通信機能など、本拡張が対象とする機能の多くについてはデファクトスタンダードが存在する。T2EX のAPI設計において、T-Kernel のタスクベースのプログラミングAPIとして最適な形式を指向する一方で、コードの再利用性や学習コスト削減の観点から、標準Cライブラリや POSIX仕様との親和性を考慮している。
具体的には、以下の点で工夫が行われている。

- 標準Cライブラリ・POSIX仕様のエラー番号(errno)のエラーコード(ER型)への統合
  T-Kernel 2.0 のエラーコードの体系内でエラー番号(errno)をそのまま扱うことができるようにエラーコードを拡張した。これにより、エラーコード(ER型)・エラー番号(errno_t型)の混在による混乱を生じることなく、標準CやPOSIX仕様のエラーコードを利用することが可能となる。

- 標準C互換ライブラリの提供
  標準入出力を含むC言語標準ライブラリの多くの機能を、エラー番号(errno)への出力を行わない点を除きそのまま提供する。これにより、他の多くのプラットフォームとのソースコードの共有が大幅に容易になる。

- POSIX仕様をベースとしたAPIコールの名称およびスタイル(引数など)
  ファイル管理機能やネットワーク通信機能において、デファクト・スタンダードであるPOSIX仕様の関数の名称およびスタイルをベースとした仕様を用いている。これにより、他の環境との間のコードの再利用性や学習コストの削減が可能となる。

- オリジナルの仕様と動作の異なる関数に対する、シンボル名・マクロ名の衝突の回避
  エラー番号(errno)の出力の有無を除けば、T2EX で提供される全ての関数・マクロについて、C言語標準ライブラリやPOSIX仕様に含まれるものと同名でありながら、異なる外部仕様を持たないように設計されている。これにより、関数の挙動についての誤解を避けることができるほか、他のプラットフォームとの間でのソースコードの再利用を行う際などのミスを避けることができる。

スレッドセーフに対する保証
標準CライブラリやPOSIX仕様にはスレッドセーフでない関数が含まれており、誤った利用により競合条件による思わぬバグを引き起こす可能性がある。T2EX では、すべてのAPIコールをスレッドセーフにすることで、利用者が意識することなく、このような誤りを回避できるようにしている。

1.3. T-Kernel 2.0 との関係

T2EX は T-Kernel 2.0 の追加機能(アドオン)であり、T-Kernel 2.0 のアプリケーションプログラムから使える機能セットを提供するものである。つまり、T2EX は T-Kernel 2.0 の上位にプロセス環境などの別のAPI階層を作り上げる拡張ではなく、T-Kernel 2.0 API を用いたアプリケーションプログラムの中で利用できる追加機能を提供する拡張である。

したがって、例えば以下のような T-Kernel 2.0, T2EX の API を両方用いた関数を定義し、T-Kernel 2.0 のプログラム中から直接利用することが可能である。
この例では、tk_set_flg, tk_wai_flg, tk_ext_tsk が T-Kernel 2.0 のシステムコール、so_recv, fprintf が T2EX によって提供されるAPIコールである。
このような特徴により、ファイル入出力やネットワーク通信を含む高度なアプリケーションにおいても、直接的に T-Kernel 2.0 の同期・通信機能を用いた高度なリアルタイムシステムを構築することが可能となる。

```
void socketReaderTask( INT stacd, void* exinf )
{
        ER ercd;
        UINT flgptn;

        for (;;) {
                /* ネットワークからデータを受信 */
                ercd = so_recv(sd, data, sizeof(data));
                if (ercd <= 0) {
                        fprintf(stderr, "Task terminated¥n");
                        break;
                }
                len = ercd;

                /* ネットワークからの受信完了を通知 */
                tk_set_flg(flgid, FLG_DATA_PRODUCED);

                /* 受信したデータが利用されるまで待つ */
                tk_wai_flg(flgid, FLG_DATA_CONSUMED, (TWF_ANDW|TWF_BITCLR), &flgptn, TMO_FEVR);
        }

        tk_ext_tsk();
}
```

なお、一部の T-Kernel 2.0 の機能については T2EX との混在した利用が制限される。
詳細は2.4節を参照すること。

1.4. POSIX API 仕様との関係

T2EX で提供されるAPIの多くは、可能な限りPOSIX仕様との親和性を持たせた設計としている。
しかしながら、T2EX は組込みシステムに特化して設計された拡張機能であることから、
POSIX API 仕様を大幅に単純化・軽量化した仕様となっており、異なる部分が多い。

本拡張機能における POSIX API 由来の機能について、オリジナルの仕様からの差分の概要をまとめると以下のとおりとなる。

・プロセス
　T2EX は T-Kernel 2.0 のタスクプログラミングを前提とするシステムであることから、
　POSIXのプロセスに相当する機能は提供されない。これに伴い、本来のPOSIX関数はプロセスの
　コンテキストで実行されるが、T2EX ではこれに伴いPOSIXのプロセスに相当する実行コンテキストは
　システムで単一となる。

　例を挙げると、POSIXにおけるファイルディスクリプタはプロセスごとに割り当てられている。
　あるプロセスでファイルディスクリプタを閉じた場合でも、別のプロセスではそのファイル
　ディスクリプタが開かれたまま保持されている可能性がある。
　T2EXでは全てのタスクでファイルディスクリプタの情報は共通であり、異なるタスク間で
　ファイルディスクリプタの状態が異なって見えることはない。

・スレッド
　POSIXにおけるスレッドに相当する機能として、T-Kernel 2.0 のタスク機能を代わりに用いる。
　このため、POSIXスレッド互換のAPIは提供されない。

・シグナル
　POSIX互換のシグナル機能については提供しない。
　代わりに T-Kernel 2.0 のタスク例外処理機能を利用できる。

・ユーザとグループ
　POSIXにおけるユーザやグループに関する機能は提供しない。ユーザ・グループに対するアクセス制御も
　行わない。
　これにより、POSIXではスーパーユーザの権限を必要とする操作であっても、T2EXでは任意のタスクから
　実行可能となる。

・ファイル
　POSIX仕様においてはディスク上のファイル・ソケット・デバイス・同期オブジェクトなどが
　(広義の)ファイルとして抽象化され、いずれもファイルAPIを通して操作できるように設計されている。
　一方、T2EX ではこのような抽象化されたファイル機能は提供せず、独立したモジュールとして
　機能単位ごとに明確に分割・分離されている。

　例えば、ファイル管理機能とネットワーク通信機能を例に挙げると、それぞれの機能ごとに

ディスクリプタは独立して定義され、APIセットについてもそれぞれ独立して提供される。
(例: fs_read, so_read)

・エラー番号
POSIX仕様においては、エラー番号はスレッド局所変数 errno に出力される。
T2EX では、T-Kernel 2.0 API との統一性の実現や実行効率の向上を目的として、
errno 変数へのエラー番号の出力を廃止し、代わりに POSIX のエラー番号を内包するように
拡張されたエラーコードを返すように変更されている。(詳細については 2.5 節を参照すること。)

なお、APIコールによっては細かい制限事項や変更点なども含まれることがある。詳細についてはAPIコールの定義を参照すること。

1.5. 機能モジュール間の依存関係

T2EX において提供される機能は、機能単位に分割されたモジュールとして実装されており、必要に応じて一部の機能モジュールのみを選択して利用することが可能となっている。ただし、一部のモジュール間には依存関係があるため、利用モジュールの選択においてはこの制約を満たさなければならない。
また、一部の機能についてはデバイスドライバに依存しており、これらの利用には対応するデバイスドライバが必要となる。

図1.1中でも概略が示されているが、以下にモジュール相互間およびモジュールとデバイスドライバとの間の依存関係を示す。

機能モジュール相互間の依存関係
- 標準入出力機能のうち、コンソールやファイルへの入出力を行う機能を利用するためには、ファイル管理機能が必要である。
- プログラムロード機能の利用には、ファイル管理機能が必要である。
- 標準C互換ライブラリのうち、ネットワークに関連する機能(inet.hなど)の利用にはネットワーク通信機能、時刻に関する機能(time.hなど)の利用にはカレンダ機能が必要である。

機能モジュールとデバイスドライバの間の依存関係
- ファイル管理機能を利用するためには、入出力の対象とする物理デバイスに対するドライバ(コンソールドライバ・ディスクドライバ)が必要である。
- ネットワーク通信機能を利用するためには、LANドライバが必要である。

コンソールドライバ・ディスクドライバ・LANドライバの仕様については、「T-Engine 標準デバイスドライバ仕様」を参照すること。

第2章　共通規定

2.1. データ型

T2EX は T-Kernel 2.0 の追加機能(アドオン)として利用されることから、
T-Kernel 2.0 仕様において定義されるデータ型についてはそのまま継承される。T-Kernel 2.0 仕様書の
3.1節で定義される以下のデータ型については、本拡張においても基本データ型として利用される。

```
typedef signed char          B;     /* 符号付き  8ビット整数 */
typedef signed short         H;     /* 符号付き 16ビット整数 */
typedef signed long          W;     /* 符号付き 32ビット整数 */
typedef signed long long     D;     /* 符号付き 64ビット整数 */
typedef unsigned char       UB;     /* 符号無し  8ビット整数 */
typedef unsigned short      UH;     /* 符号無し 16ビット整数 */
typedef unsigned long       UW;     /* 符号無し 32ビット整数 */
typedef unsigned long long  UD;     /* 符号無し 64ビット整数 */

typedef char                VB;     /* 型が一定しない  8ビットのデータ */
typedef short               VH;     /* 型が一定しない 16ビットのデータ */
typedef long                VW;     /* 型が一定しない 32ビットのデータ */
typedef long long           VD;     /* 型が一定しない 64ビットのデータ */
typedef void                *VP;    /* 型が一定しないデータへのポインタ */
```

注: T-Kernel 1.0 では exinf などのデータタイプを VP としていたが、
T-Kernel 2.0 では、CONST修飾子との関係から、原則としてVPは使わず、
VP の代わりに "void *" のデータタイプ定義を直接記述することにした。
T-Kernel 2.0 および T2EX でも、互換性維持のためにVP自体の定義は
残しているが、VP の新規の使用は推奨しない。

```
typedef volatile B          _B;     /* volatile 宣言付 */
typedef volatile H          _H;
typedef volatile W          _W;
typedef volatile D          _D;
typedef volatile UB         _UB;
typedef volatile UH         _UH;
typedef volatile UW         _UW;
typedef volatile UD         _UD;

typedef signed int          INT;    /* プロセッサのビット幅の符号付き整数、32ビット以上 */
typedef unsigned int        UINT;   /* プロセッサのビット幅の符号無し整数、32ビット以上 */

typedef INT                 ID;     /* ID一般 */
typedef W                   MSEC;   /* 時間一般(ミリ秒) */

typedef void            (*FP)();    /* 関数アドレス一般 */
typedef INT             (*FUNCP)(); /* 関数アドレス一般 */

#define LOCAL           static      /* ローカルシンボル定義 */
#define EXPORT                      /* グローバルシンボル定義 */
#define IMPORT          extern      /* グローバルシンボル参照 */

/*
 * ブール値
 *      TRUE = 1 と定義するが、0 以外はすべて真(TRUE)である。
 *      したがって、bool == TRUE の様な判定をしてはいけない。
 *      bool != FALSE の様に判定すること。
 */
typedef INT             BOOL;
#define TRUE            1           /* 真 */
#define FALSE           0           /* 偽 */

/*
 * TRON 文字コード
 */
typedef UH              TC;         /* TRON 文字コード */
#define TNULL           ((TC)0)     /* TRON コード文字列の終端 */
```

これらの T-Kernel 2.0 由来の基本データ型に加え、T2EX の提供する機能において利用される各種型定義が追加される。ほとんどは標準Cライブラリや POSIX 仕様に準拠したものであることから、他環境からのプログラムの移植やプログラムコードの相互利用が容易となる。
一部 T2EX 固有の型も追加されるが、いずれもPOSIX風のスタイルを持つ。(例: pm_entry_t)

T2EX に追加定義される基本的データ型のうち代表的なものを以下に示す。

```
typedef signed char         int8_t;     /* 符号付き  8ビット整数 */
typedef signed short        int16_t;    /* 符号付き 16ビット整数 */
typedef signed long         int32_t;    /* 符号付き 32ビット整数 */
typedef signed long long    int64_t;    /* 符号付き 64ビット整数 */
typedef unsigned char       uint8_t;    /* 符号無し  8ビット整数 */
typedef unsigned short      uint16_t;   /* 符号無し 16ビット整数 */
typedef unsigned long       uint32_t;   /* 符号無し 32ビット整数 */
typedef unsigned long long  uint64_t;   /* 符号無し 64ビット整数 */

typedef signed long         intptr_t;   /* ポインタのビット幅を持つ符号付き整数 */
typedef unsigned long       uintptr_t;  /* ポインタのビット幅を持つ符号無し整数 */

typedef int                 errno_t;    /* エラー番号(後述)を表す整数 */

typedef unsigned long       size_t;     /* サイズを表す符号無し整数 */
typedef signed long         ssize_t;    /* サイズを表す符号付き整数 */


typedef signed long         time_t;     /* 秒単位の時刻を表す整数型 */

struct timeval {                        /* マイクロ秒単位の時刻を表す構造体 */
                long        tv_sec;     /* 秒 */
                long        tv_usec;    /* マイクロ秒 */
};
```

その他の型については、第4章以降の各節を参照すること。

T-Kernel 2.0 由来の基本データ型と、標準Cライブラリや POSIX 仕様由来のデータ型が両方とも定義されることで、似た意味合いの型が重複して定義されることとなる。例えば、UB 型と uint8_t 型はほぼ同じ意味を持っている。
利用者は、これらの型について可能な限り本仕様および T-Kernel 2.0 仕様の意味論に沿う形でこれらを使い分けることが推奨される。
例えばイベントフラグのフラグパターンとしては uint32_t を用いるのは適切でなく UINT 型を用いるべきである。
一方、ファイルのシーク(fs_lseek64)の第2引数(ファイルオフセット)としては、関数プロトタイプ通りに off64_t を用いることが推奨される。

2.2. 文字列

T2EX の仕様において、char 型の配列で表現される文字列のエンコーディングは UTF-8であるものと規定する。
本拡張が定義する全てのAPIについても、この規定に基づいて動作する。

例として、ファイルのオープン(fs_open)の際に指定される引数のファイル名は UTF-8 の文字列を用いて行わなければならない。ISO-8859-1 や Shift-JIS 等の誤ったエンコーディングの文字列を与えた場合にも、与えられたファイル名は UTF-8 の文字列として解釈される。

2.3. 利用可能なコンテキスト

T2EX において提供されるすべてのAPIコールはタスク部・準タスク部において利用可能であるものと規定する。
一方、タスク独立部からはこれらのAPIコールは利用できない。

実行時コンテキストのチェックを行い不正なコンテキストからの実行に対して E_CTX のエラーを返す実装も可能であるが、このようなチェックの有無については実装依存とし、動作を保証しない(未定義とする)実装を許容するものとする。

例外的に、以下のAPIコールについてはタスク独立部からの利用が可能であるものとする。

- fs_break (ファイル操作の中止)
- so_break (ソケット操作の中止)

2.4. T-Kernel 2.0 APIの利用

2.3節に示されるとおり、T2EX のAPIはタスク部・準タスク部において利用することができる。
ただし、T2EX を利用するタスクを対象とした、タスク管理機能やタスク付属同期機能の利用については、以下の通り制限される。

・tk_ter_tsk
　　利用不可。利用した場合のシステムの動作は保証されない。

・tk_sus_tsk, tk_rsm_tsk, tk_frsm_tsk
　　利用不可。利用した場合のシステムの動作は保証されない。

・tk_sig_tev, tk_wai_tev(_u)
　　タスクイベント番号 1～4 をT2EXシステム内部での利用のために予約し、利用を禁止する。
　　アプリケーションからはタスクイベント番号として 5～8 を利用可能とする。

・tk_dis_wai, tk_ena_wai
　　利用不可。利用した場合のシステムの動作は保証されない。

・tk_ras_tex
　　利用不可。利用した場合のシステムの動作は保証されない。

上記以外の T-Kernel 2.0 API については、T2EX と混在させて利用した場合にも利用可能であり、
仕様通りに動作することが保証される。なお、tk_rel_wai を T2EX API コールを利用しているタスクに対して発行した場合、以下の動作となる。

・tk_rel_wai
　　利用可能。T2EX APIコールの実行中に tk_rel_wai が実行された場合、以下の動作となる。

　　・待ちに入っている T2EX APIコールの待ち状態は解除される可能性はあるが、
　　必ず解除されるとは限らない。
　　待ち状態が解除された場合は、T2EX APIコールは EX_INTR を返し、tk_rel_wai は E_OK を返す。
　　一方、待ちが解除されなかった場合は、tk_rel_wai は E_OBJ を返す。

T2EXのファイル管理機能・ネットワーク通信機能の専用の待ち解除APIコールとしては、tk_rel_wai とは別にそれぞれ fs_break, so_break が提供される。これらのAPIコールを利用することで、T2EXの各機能に限定して安全に待ちを解除することが可能である。

2.5. エラーコード

T2EX では、ファイル管理機能やネットワーク通信機能などの追加機能を提供することから、T-Kernel 2.0のエラーコードを拡張したエラーコード体系を定義する。
本拡張においては、標準Cライブラリや POSIX仕様との高い親和性を実現するため、エラー番号(errno)を以下のように T-Kernel のエラーコード(ER型)の体系に統合する。

- メインエラーコードとして EC_ERRNO を定義する
- POSIX仕様における errno に対応するエラーコードとして、
  EX_ の接頭辞を持つエラーコードを以下の通り定義する
  - エラーコードの名称として、エラー番号名の先頭の E を EX_ に置き換えた名前を用いる
  - メインエラーコードとして EC_ERRNO を持つ
  - サブエラーコードとして errno の値を持つ

例えば、ファイルディスクリプタが不正であるという異常を示すエラー番号 EBADF に対し、T2EX のエラーコードは以下の通り定義される。

```
#define EX_BADF             ERCD(EC_ERRNO, EBADF)
```

加えて、T2EX で追加されるエラーコード(ER型)とエラー番号(errno)とを相互変換するためのマクロとして、ERRNO, ERRNOtoER をそれぞれ定義する。

```
#define ERRNO(er)           (MERCD(er) == EC_ERRNO ? SERCD(er) : 0)
#define ERRNOtoER(eno)      (ERCD(EC_ERRNO, (eno)))
```

追加されるエラーコードの定義の一覧については、8.8節を参照すること。
なお、T-Kernel 2.0 において定義される E_ の接頭辞を持つエラーコードについては、T2EX を利用するアプリケーションでもそのまま利用可能である。

なお、POSIXのエラー番号に対応したエラーコードの導入に伴い、似た意味を持つエラーコードが複数導入されることとなる。
例えば、対象のリソースが利用中であることを示すエラーとして E_BUSY, EX_BUSY の2種類が定義されるが、これらの使い分けにあたっては、対象となるリソースの種別によって区別を行う方針とする。
ビジー状態にあるオブジェクトが T-Kernel 2.0 の同期オブジェクトであれば前者を利用するものとし、ファイルやネットワークが対象であれば後者を用いなければならない。
また、ミドルウェア・ライブラリ等では可能な限り T-Kernel 2.0 のエラーコードとT2EXで追加されたエラーコードの利用を統一し、混乱が生じないように配慮することを推奨する。

2.6. スレッドセーフ

本仕様書では、マルチタスク環境下における関数のスレッドセーフという性質について以下のとおり定義する。

関数のスレッドセーフ
　　ある関数の呼び出しが行われている間に、渡された引数から直接的または間接的に指定された
　　メモリ領域が外部から参照・改変されない条件のもとで、その関数を同時並行的に実行しても
　　問題が発生しないことが保証されるとき、その関数はスレッドセーフであるという。
　　これらが保証できないとき、その関数はスレッドセーフではないという。

T2EX API コールについても上記と同様にスレッドセーフという性質を定義する。

T2EX は T-Kernel によるタスクベースのプログラムを前提としたシステムであるため、システムの信頼性の実現の観点から、そのAPIコールがスレッドセーフであることは非常に重要性が高い。
このため、T2EX で提供される全てのAPIコールはスレッドセーフである。

なお、T2EX にはPOSIX仕様や標準Cライブラリをベースとしたコールが多く含まれるが、スレッドセーフではないAPIコールについては提供されず、代替となるスレッドセーフなAPIコールが提供される。
例としては 標準Cライブラリにおける localtime() はスレッドセーフではないことから提供されず、代替となるスレッドセーフなAPIコール localtime_r(), localtime_r_eno() が T2EX では提供される。

なお本仕様におけるAPIコールのスレッドセーフの定義に示されるとおり、引数によって直接的または間接的に指定されるメモリ領域がAPIコールを実行している間に外部から参照・改変される場合の動作について、T2EX APIコールは保証しない。
例えば、char* 型のグローバル変数 a に対して、2つのタスク A, B が以下に示されるそれぞれの処理を同時に実行した場合の動作は未定義である。このようなケースでは、引数のメモリ領域が重複しないように工夫することや排他制御の導入などにより、利用者の責任においてスレッドセーフの条件を満たす必要がある。

タスクA
```
strcpy(a + 3, "aaaa");
```

タスクB
```
strcpy(a + 2, "bbbbbb");
```

第3章　メモリ保護機能

3.1 概要

T2EX では、T-Kernel 2.0 のメモリ保護モデルに基づく、2レベルのメモリ保護機能を提供する。

T2EX システムは単一の論理アドレス空間によって構成され、タスクに固有の空間を持たない。
これは、論理アドレスと物理アドレスが一対一に対応づけられていることを意味するが、
必ずしも論理アドレスと物理アドレスは一致していなくともよい。

この条件の下で、OSカーネル(T2およびT2EX)やシステムプログラム(一部のデバイスドライバやタスク)で使用するメモリ領域をユーザアプリケーションから保護することにより、システム全体としての安定性・信頼性を向上させることを目的とする。

3.2 メモリ保護モデル

3.2.1 保護レベル

T-Kernel 2.0 における保護レベルは 0〜3 の4段階からなるが、T2EX ではこれらを2段階に分類し、
特権・ユーザの2レベルからなるメモリ保護を提供する。

・特権レベル
　　T-Kernel 2.0 における保護レベル 0〜1 に相当し、タスク部が保護レベル 0〜1 で実行される際や、
　　非タスク部(タスク独立部、準タスク部など)の実行の際の保護レベルにあたる。
　　特権レベルのプログラムはCPUの特権モードで動作する。

・ユーザレベル
　　T-Kernel 2.0 における保護レベル 2〜3 に相当し、タスク部が保護レベル 2〜3 で実行される際の
　　保護レベルである。
　　ユーザレベルのプログラムはCPUのユーザモードで動作する。

T-Kernel 2.0 および T2EX のAPIコールの呼び出し可能なレベルについては、システム構成情報を通して
メモリ保護の境界とは独立して指定される。T2EXでは、各保護レベルを以下のような用途に利用する。

保護レベル	用途
0	カーネル、サブシステム、デバイスドライバなど
1	システムアプリケーションタスク
2	ユーザアプリケーションタスク(T-Kernel・T2EX APIが利用可能)
3	[*予約] ユーザアプリケーションタスク(T-Kernel・T2EX APIが利用不可)

(*) プロセスなどの上位OS機能に利用するものとして予約し、T2EXでは利用しない

表中に示される通り、T-Kernel 2.0 および T2EX の API は0から2までの保護レベルでのみ利用可能である。
保護レベル3のタスクからT-Kernel 2.0またはT2EXのAPIを利用した場合の動作は保証されない。

3.2.2 メモリアクセス権

特権レベル、ユーザレベルそれぞれのプログラムからの各メモリ領域へのアクセス権を 表 3.1 に示す。
表中において、R は読込み(read)権、W は書きこみ(write)権、EXは実行(execute)権、- はアクセス不可を示す。

表3.1 T-Kernel 2.0 Extension におけるメモリアクセス権

メモリ領域	特権レベル	ユーザレベル
特権レベルのプログラム領域	R,EX	-
特権レベルの読み書き可能静的データ領域	R,W	-
特権レベルの読込み専用静的データ領域	R	-
ユーザレベルのプログラム領域	R,EX	R,EX
ユーザレベルの読み書き可能静的データ領域	R,W	R,W
ユーザレベルの読込み専用静的データ領域	R	R
動的割り当て特権レベルメモリブロック	R,W	-
動的割り当てユーザレベルメモリブロック	R,W	R,W
未使用(未割り当て)メモリ	-	-

なお、表 3.1 は各メモリ領域に対する標準的なアクセス権を示したものであるが、ハードウェア
の制限や以下のようなケースに対応するため、実装に依存して、表3.1 とは異なるアクセス権と
することを許容する。

・特権レベルの静的データ領域、動的割り当て特権メモリブロックに対するユーザレベルの読込み権(R)の追加
　デバイスドライバなどの特権レベルのデータ領域をユーザレベルのアプリケーションから
　直接読めるようにすることで、メモリコピーなしでデータの受け渡しが可能となる。

・データ領域や動的割り当てメモリブロックに対する実行権(EX)の追加
　JIT(Just-In-Time)コンパイルなどの技術を実現するためには、データ領域に実行権を
　割り当てることが必要となる。

・プログラム領域への書込み権(W)の割り当て
　自己書き換えを行うことで動作するプログラムでは、プログラム領域への書込み権が必要となる。

3.2.3 静的メモリ保護と動的メモリ保護

T2EXでのメモリ保護は、静的メモリ保護と動的メモリ保護の2つに分けられる。

静的メモリ保護
　表 3.1 で示したメモリ領域のうち、動的割り当てメモリブロックを除く領域は、システム生成時
　に決定される各メモリ領域のアドレスに対して固定的なアクセス権を設定することで保護を行う。
　通常、システム生成時のリンカスクリプトにより各アドレス領域(セクション)のアクセス権を設定
　することになるが、具体的な設定方法は実装に依存する。

動的メモリ保護
　T-Kernel 2.0 の API コールにより動的に獲得されるメモリ領域に対して、指定した保護レベル
　のアクセス権を設定することで保護を行う。
　表 3.1 で示したメモリ領域のうち、動的割り当てメモリブロック領域が対象となる。

3.2.4 T-Kernel 2.0 のメモリ関連機能との関係

T2EX メモリ保護機能の下では、T-Kernel 2.0 の メモリ保護に関連する API コールは以下の動作となる。

tk_set_tsp
　正常なパラメータの指定に対しては、何もせずに E_OK を返す。

SetTaskSpace
　正常なパラメータの指定に対してアクセス権情報の変更のみを行い、タスク固有空間の切り替えは行わない。
　特権レベルの実行時保護レベルからのみ利用可能。

LockSpace, UnlockSpace
　正常なパラメータの指定に対しては、何もせずに E_OK を返す。

MapMemory
　paddr に NULL を指定した場合は、T-Kernel 2.0 の仕様書通りの動作となる。
　有効な物理アドレスが paddr に指定された場合、メモリ領域のアクセス権の設定を行わずに論理アドレスへの
　変換のみを行い E_OK を返す。(特定の物理アドレス領域に対する権限の設定は、静的メモリ保護の設定を
　通して行うことを前提とする。)

CnvPhysicalAddr
　T-Kernel 2.0 の仕様書通りの動作。

tk_cre_mpf, tk_cre_mpl, tk_get_smb
　attr において TA_RNG0～1 が指定された場合には特権レベルのプログラムのみからアクセス可能な
　メモリを割り当て、TA_RNG2～3 が指定された場合には任意のレベルからアクセス可能なメモリを割り当てる。

ChkSpaceBstrR, ChkSpaceBstrRW, ChkSpaceR, ChkSpaceRE, ChkSpaceRW, ChkSpaceTstrR, ChkSpaceTstrRW
　T-Kernel 2.0 の仕様書通りの動作。

Kmalloc, Kcalloc, Krealloc, Kfree,
Vmalloc, Vcalloc, Vrealloc, Vfree
　T2EX のシステム構成情報(A.1節)によって指定されたレベルのメモリの割り当て、リサイズおよび解放を行う。
　特権レベルの実行時保護レベルからのみ利用可能。

Smalloc, Scalloc, Srealloc, Sfree
　ユーザレベルのメモリの割り当て、リサイズおよび解放を行う。特権レベル・ユーザレベルの両方から利用可能。

CreateLock, DeleteLock, Lock, Unlock

特権レベルの実行時保護レベルからのみ利用可能。
ユーザレベル向けには、以下の代替APIコールが提供される: CreateULock, DeleteULock, ULock, UUnlock

CreateMLock, DeleteMLock, MLock, MLockTmo, MLockTmo_u, MUnlock
特権レベルの実行時保護レベルからのみ利用可能。
ユーザレベル向けには、以下の代替APIコールが提供される: CreateUMLock, DeleteUMLock, UMLock, UMLockTmo, UMLockTmo_u, UMUnlock

DI, EI, SetIntMode, EnableInt, DisableInt, ClearInt, CheckInt
特権レベルの実行時保護レベルからのみ利用可能。

StartPhysicalTimer, StopPhysicalTimer, GetPhysicalTimerCount, DefinePhysicalTimerHandler, GetPhysicalTimerConfig
特権レベルの実行時保護レベルからのみ利用可能。

GetSpaceInfo
T-Kernel 2.0 の仕様書通りの動作。

SetMemoryAccess
T-Kernel 2.0 の仕様書通りの動作。

in_d, in_w, in_h, in_b, out_w, out_d, out_w, out_h, out_b
特権レベルの実行時保護レベルからのみ利用可能。

WaitUsec, WaitNsec
特権レベルの実行時保護レベルからのみ利用可能。

3.2.5 メモリ保護違反時の処理

タスク部または準タスク部の実行中に、そのレベル(特権レベルまたはユーザレベル)からアクセス権のないメモリ領域をアクセスしようとした状態を「メモリ保護違反」と呼ぶ。
「メモリ保護違反」が検出された場合には、メモリ保護違反例外が発生し、その例外は T2EX 内部で管理される。

ユーザレベルのタスクのタスク部においてメモリ保護違反例外が発生した場合、その例外は T2EX 内部で管理される例外処理用のタスクに通知される。このタスクをシステム例外処理タスクと呼ぶ。
システム例外処理タスクは保護レベル 0 で動作する優先度 1 のタスクであり、メモリ保護違反例外に対して処理関数 TaskMemFaultHdr() を同タスクのコンテキストから呼び出す。

一方、タスク独立部や準タスク部、あるいはシステムレベルのタスクのタスク部でメモリ保護違反例外が発生した場合、
その例外はタスク独立部のコンテキストで呼び出される処理関数 RawMemFaultHdr() で処理される。

ユーザはこれらの関数の内容を改変することで、ファイルシステムの同期処理やシステムリセットなど、目的に応じた独自の例外処理を行うことができる。

T2EX では上記の標準構成に加え、システム例外処理タスクを用いない簡易構成をサポートする。
この構成では、システム例外処理タスクは生成されず、全てのメモリ保護例外はタスク独立部のコンテキストで呼び出される関数 RawMemFaultHdr() で処理される。ユーザがこの関数の内容を改変し、目的に応じた処理例外を行うことが可能である点については、標準構成と共通である。

例外処理関数はそれぞれ以下の通りの形式を持つ。

○ void TaskMemFaultHdr(MemFaultInfo* fault);

fault で示されるメモリ保護違反例外に対する処理を行う。
システム例外処理タスクのコンテキストから呼び出される。

faultの内容は以下に示す通りである:

ID	tskid	例外を発生させたタスクID
UW	vecno	例外の割り込みベクタ番号
UW	excinfo	保護違反例外情報 (実装依存)
void*	excaddr	例外の発生したアドレス(アクセス対象)

---(以下に実装独自に他の情報を追加してもよい)---

○ void RawMemFaultHdr(MemFaultInfo* fault);

fault で示されるメモリ保護違反例外に対する処理を行う。タスク独立部のコンテキストから呼び出される。

faultの内容は TaskMemFaultHdr() と共通である。

第4章　ファイル管理機能

4.1 概要

以降の説明において、T2EX におけるファイル管理機能を提供するプログラムモジュールを「ファイルマネージャ」と呼ぶ。

ファイルマネージャは、木構造型のディレクトリとファイルからなるファイルシステムにアクセスする機能を提供する。API 名称のプレフィクスとして ″fs_″ (file system) を持つ。

ファイルマネージャは、POSIX 仕様におけるファイル入出力機能に近いAPIセットを提供する。大きな違いは、T2EX の API の戻値が負の場合はエラーコードを表している点である。

また、64 ビット大容量ファイルをサポートしており、64bitのファイルサイズおよびファイルオフセットを直接指定可能である。ただし、一度に読み書きできるサイズは 32bit に制限される。32ビットのAPIと、64ビットのAPIは独立して用意されており、並存して使用することが可能である。

時刻を必要とするAPIでは、POSIX 仕様の時刻データ形式の他に T-Kernel 2.0 の時刻データ形式(SYSTIM, SYSTIM_U)を使える API を追加している。

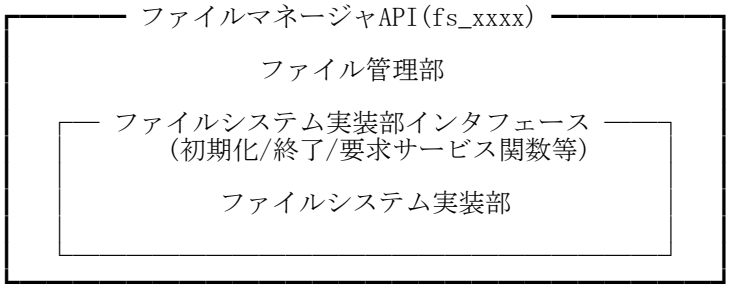


図 4.1 ファイルマネージャの構造

ファイルマネージャは、図4.1に示したような構造になっている。

以下の節で示すように、ファイルマネージャAPIは、最初にファイル管理部で解釈され、FAT や NTFS 等の特定のファイルシステム形式のファイル操作を行うファイルシステム実装部の関数を呼び出す。

ファイルシステム実装部は、特定のファイルシステムを処理するためのプログラムコードで、ストレージデバイスに対するデバイスドライバを利用して、FAT、exFAT、NTFS、ext2、フラッシュROM用ファイルなどのファイルシステムにしたがったファイル管理の機能を提供する。

T2EXでは複数のファイルシステム実装部を利用することができ、ユーザ定義のファイルシステム実装部のほか、FATに対応したT2EX標準添付のファイルシステム実装部として「基本FATファイルシステム実装部」が利用できる。

T2EXでは複数のファイルシステム実装部を利用することができ、T2EX標準添付のファイルシステム実装部である「基本FATファイルシステム実装部」に加えて、ユーザ定義のファイルシステム実装部が利用できる。

4.2 概念・用語

□ファイルシステム形式

一般に FAT, NTFS, ext2 と呼ばれる、デバイス上にファイル構造を配置し管理するための論理フォーマットの事を、本書ではファイルシステム形式と呼ぶ。

□ファイルシステム

一般には、木構造型のディレクトリとファイルからなるデバイス上に展開されたファイルの体系または、それを管理する機能の総称を指す。

本書では一つのファイルシステム形式を持つデバイス上のディレクトリツリー全体を指し、システムのディレクトリツリーに接続される単位となる。

例えば、ファイルシステムのサイズ、ファイルシステムの統計情報、ファイルシステムの同期、読込み専用ファイルシステムといった場合の「ファイルシステム」は、システムに接続されている一つのファイルシステム形式を持つ単位を意味する。

□ファイルシステム実装部

ファイルシステム形式に対するアクセスを実際のデバイス上のアクセスとして処理するためのユーザが記述することが可能なプログラムコードのことで、動的にファイルマネージャに登録することができる。

T2EX では、FAT ファイルシステム形式を扱う基本FATファイルシステム実装部が提供されており、ファイルマネージャの初期化時にシステムに登録され、利用できるようになる。別のファイルシステム形式を扱うファイルシステム実装部を記述してシステムに登録することで、別のファイルシステム形式を扱うことができる。

また、同じファイルシステム形式に対して複数のファイルシステム実装部を登録し、デバイスやメディア単位で切り替えながら利用する事もできる。

ファイルシステム実装部は、システムに登録し(fs_regist)、実際に操作対象となるデバイスと関連付ける(fs_attach)ことで利用可能となる。それにより、デバイス内のディレクトリツリーは、システムに組み込まれるため、パス名を用いてディレクトリやファイルのオープン(fs_open)、読込み(fs_read)、書込み(fs_write)といったファイル操作ができるようになる。

ファイルシステム実装部は、ファイルシステム形式と一対一に対応している必要はない。例えば、一つのファイルシステム実装部で複数のファイルシステム形式を扱うようなプログラムを記述することもできる。

□基本FATファイルシステム実装部

基本FATファイルシステム実装部は、FAT ファイルシステム形式をサポートするファイルシステム実装部であり、標準で提供される。

FAT12, FAT16, FAT32 のファイルシステム形式を扱うことができ、VFATのロングファイル名も使用可能である。区画情報のあるデバイス(メディア)にも、区画情報の無いデバイス(メディア)にもアクセス可能である。ただし、基本区画のみが利用可能であり、拡張区画には対応していない。

基本FATファイルシステム実装部は、fs_main() によるファイルマネージャの初期化を行うと登録されるため、fs_regist() による登録は必要ない。

ファイルシステム実装部の登録を解除するには、登録解除(fs_unregist)を行う。基本FATファイルシステム実装部を登録解除(fs_unregist)することも可能である。

□ディレクトリ

ファイルや他のディレクトリを示すエントリを含んでいる特殊なファイルである。

ディレクトリの中に別のディレクトリを入れることにより、階層的なディレクトリ構造を作ることができる。

1つのデバイス内のファイルとディレクトリは、全体として木構造になる。

□ファイル名

ファイルシステムにおいてファイルまたはディレクトリを特定するための名前である。

ファイル名はユニコード(UTF-8)の文字列で、以下の文字以外を使用することができる。

'/' ':' '¥' '*' '?' '"' '<' '>' '|' '¥0'(ヌル文字)

ファイル名の最大長は NAME_MAX で定義される。ただし、使用するファイルシステム実装部によっては、ファイル名の最大長がこれより短い可能性がある。

以下の特殊なファイル名が存在する。

"." ディレクトリツリー上の現在の位置のディレクトリを表す。
".." ディレクトリツリー上で一つ上の位置のディレクトリを表す。

□ルートディレクトリ

ルートディレクトリは、他のすべてのディレクトリの上位に位置する仮想的なディレクトリである。

ルートディレクトリの直下には、現在接続されている全てのデバイス上の最上位ディレクトリが仮想的なサブディレクトリとして存在する。

□カレントディレクトリ

カレントディレクトリとは、ファイル操作の現在の基準位置としているディレクトリを示す。

後述する相対パス名が与えられた場合の、ファイルまたはディレクトリに対する検索の開始位置となる。

T2EX においては、カレントディレクトリはタスク毎ではなく、システム全体で共通に一つだけ存在する。すなわち、全てのタスクは一つのカレントディレクトリを共有している。

□接続ポイント

ルートディレクトリにデバイス上のファイルシステムを接続する場合のサブディレクトリの名称である。

fs_attach() によって操作対象とするデバイス(あるいはメディア)を指定する際に、ファイルの置かれたデバイスの名称(デバイス名)と、接続ポイントを指定し、両者の関連付けを行う。

一つの接続ポイントは、一つのファイルシステムを参照する。

接続ポイントは、最大で14文字の 'a'～'z' 'A'～'Z' '0'～'9' '_' のみからなる文字列で、'/' を含むことはできない。

□パス名

ファイルシステム内の特定のファイルまたはディレクトリの位置を示す名前の文字列表現である。

パス名は、fs_open()によりファイルをオープンする際のパラメータとして指定するほか、ファイルの生成(fs_creat())、ファイルの状態取得(fs_stat()など)、ファイルの名前や位置の変更(fs_rename())、ファイルの削除(fs_unlink())などのパラメータとしても利用する。

パス名の文字コードは、ユニコード(UTF-8)である。

パス名の最大長は PATH_MAX で定義される。

パス名の文字列は、ルートディレクトリを表す '/' を基点として、対象デバイス内の該当するファイルに到達するまでのディレクトリツリーの経路を以下のように '/' で区切って順番に並べたものである。

/接続ポイント/ディレクトリ名1/ディレクトリ名2/.../ディレクトリ名n/ファイル名

このように、ルートディレクトリからの絶対的な位置を表すパス名を絶対パス名と呼ぶ。絶対パス名は "/" で始まる。

カレントディレクトリからの相対的な位置を表すパス名を相対パス名と呼ぶ。相対パス名は、"./" で始まるが、"./" は省略可能なため、"/" で始まらないパス名は相対パス名となる。

□デバイス名

ファイルマネージャにおけるデバイス名とは、ファイルが存在しているデバイスの名称である。
一般的に次の要素により構成される。

・種別　　デバイスの種別を示す名前
・ユニット　　物理的なデバイスを示す英字
・サブユニット　論理的なデバイスを示す数字

通常は、種別+ユニット+サブユニットの形式をとるが、ユニット、サブユニットはデバイスによっては存在しない場合がある。

デバイス名の文字列は、ユニコード(UTF-8)を用いる。

□ファイルシステム実装部名

ファイルシステム実装部の名前である。この名前はファイル名の規約に従う。

以下の名前のファイルシステム実装部が、システムで予め定義されている。

"FIMP_FAT"	基本FATファイルシステム実装部
"FIMP_AUTODETECT"	将来のための予約(ファイルシステム形式自動検出)

□ファイルディスクリプタ

ファイルをオープンすることで、ファイルを識別するために新規に割り当てられる0以上の整数である。

ファイルの読み書きや各種操作を行う場合にファイルディスクリプタを使用する。

システムで以下のファイルディスクリプタがあらかじめ割り当てられており、特殊な用途に使用される。

STDIN_FILENO	0	標準入力
STDOUT_FILENO	1	標準出力
STDERR_FILENO	2	標準エラー出力

これらは、システムのコンソールデバイスに割り当てられている。これらのファイルディスクリプタをクローズ(fs_close)しても、クローズ操作を行った後、再度自動的にオープンされ、これらのファイルディスクリプタを他

のファイルに対して用いることはできない。

□ディスクキャッシュ

ディスクなどのデバイス上に置かれたファイルの書込みや読み込みを効率的に行うため、バッファとなる領域をメモリ上に設け、このバッファを経由してデバイス上のファイルを操作する実装とする場合が多い。この場合にバッファとして使用するメモリ領域をディスクキャッシュと呼ぶ。

ファイルの書込み、および読み込みを効率的に行うため、T2EX では、ディスクキャッシュを使用する。

fs_write() による書込みでは、書込みデータをディスクキャッシュに転送するだけで、実際のディスクへの書込みの完了は保証しない。

fs_close(), fs_fsync(), fs_fdatasync() 等を呼び出すことで、ディスクキャッシュ上のファイルのデータが実際のディスクに書き込まれ、ディスクキャッシュ上のデータを物理デバイスに反映させることができる。

□物理デバイスとの同期

ディスクキャッシュの内容が物理デバイスにも書き込まれ、両者の内容が完全に一致していることを、「ディスクキャッシュと物理デバイスが同期した状態」と呼ぶ。

ディスクキャッシュは揮発性のメモリ上に置かれているため、不意のシステムダウンなどにより情報が失われる可能性がある。したがって、適切なタイミングで物理デバイスとの同期をとる必要がある。

物理デバイスと同期をとるために、O_SYNC オープンモードがあり、APIコールとして、fs_sync、fs_fsync、fs_fdatasync がある。

□同期書込み

ディスクキャッシュと物理デバイスを同期させるために、ディスクキャッシュだけでなく物理デバイスへも同時に書き込む操作。

□ファイルのメタデータ

ファイルの最終アクセス時刻、最終更新時刻、最終状態変更時刻、ファイルサイズなどのデータ以外のファイルの管理情報。

□ファイルの日時指定

ファイルのメタデータには、最終アクセス時刻、最終更新時刻、最終状態変更時刻が含まれる。これらの時刻を扱う API では、time_t 型に加え、SYSTIM および SYSTIM_U 型を扱える API も提供する。

ただし、ファイルシステムの種類により、ファイル状態に関する時刻の正確な意味および精度には違いがある。

例:
- 最終アクセス時刻、最終更新時刻、最終状態変更時刻の全てを保持していないものがある。
- 時刻の精度がファイルシステムにより異なる。

FAT ファイルシステムでは、時刻の解像度は、更新時刻 2秒、アクセス時刻 1日である。最終状態変更時刻は存在せず、最終更新時刻と同じと見なす。

□ファイルモード

ファイルのタイプとアクセスするモードをファイルモードと呼び mode_t 型のデータで表す。

mode_t
ファイルのタイプおよびアクセスモードを示すデータ型。
この型のデータに対し、以下のビットマスクやマクロを適用できる。

S_IFMT	0170000	ファイルタイプを示すフィールドのビットマスク

以下のタイプに対するシンボルを定義する:

S_IFBLK	0060000	ブロック・デバイス
S_IFCHR	0020000	キャラクタ・デバイス
S_IFREG	0100000	通常のファイル
S_IFDIR	0040000	ディレクトリ
S_IFLINK	0120000	シンボリック・リンク

ファイルタイプは、ファイルシステムの種類によって使用できるタイプが異なる。
※ S_IFLINK は、FATファイルシステムでは使用しない。

ファイルタイプがどのタイプか検査するために、以下のマクロを提供する。
m は stat 構造体などの st_mode の値である。
マクロは、検査が真の場合 0 以外の値を、偽の場合 0 を返す。

S_ISBLK(m)	ブロック・デバイス
S_ISCHR(m)	キャラクタ・デバイス
S_ISDIR(m)	ディレクトリ
S_ISREG(m)	通常ファイル
S_ISLNK(m)	シンボリックリンク

さらに以下に示すアクセス許可属性を示す、モードビットを定義する:

S_IRWXU	0000700	所有者による読み、書き、実行/検索
S_IRUSR	0000400	所有者による読込み許可
S_IWUSR	0000200	所有者による書込み許可
S_IXUSR	0000100	所有者による実行/検索許可
S_IRWXG	0000070	グループによる読み、書き、実行/検索
S_IRGRP	0000040	グループによる読込み許可
S_IWGRP	0000020	グループによる書込み許可
S_IXGRP	0000010	グループによる実行/検索許可
S_IRWXO	0000007	その他による読み、書き、実行/検索
S_IROTH	0000004	その他による読込み許可
S_IWOTH	0000002	その他による書込み許可
S_IXOTH	0000001	その他による実行/検索許可
S_ISUID	0004000	実行時にユーザIDをセット
S_ISGID	0002000	実行時にグループIDをセット
S_ISVTX	0001000	ステッキービット

※実行/検索許可とはファイルが通常ファイルの場合は、実行許可を表し、ファイルがディレクトリの場合は、その中のファイルを探索することを許可することである。

これらのモード指定は、接続されているファイルシステムの種類によってはサポートされないことがあり、その場合、未サポートの指定は単に無視される。

T2EX では、所有者/グループという概念は存在せず、所有者、グループ、他人の区別は無い。このため、ファイルをオープンしたり、データの読み書きといったファイル操作を行おうとした場合、以下のように処理される。

・いずれかの読込み許可属性がセットされていれば読込み可能である。
・いずれかの書込み許可属性がセットされていれば書込み可能である。
・いずれかの実行/検索許可属性がセットされていれば実行/検索は可能である。

□ファイル生成フラグ

fs_open() の flag 内で使われるファイル生成のためのフラグである。

O_CREAT	ファイルが存在しない場合生成する
O_EXCL	排他的にオープンする
O_TRUNC	ファイル内容を切り詰める

□ファイル状態フラグ

fs_open(), fs_fcntl() で使用する、ファイルのオープン状態を表すフラグである。
ファイルディスクリプタの属性として扱われる。

O_APPEND	追加モード
O_NONBLOCK	ノンブロッキングモード
O_SYNC	書込み時にファイルI/O処理の同期を保証

□ファイルアクセスモード

fs_open()、fs_fcntl() で使用する、ファイルのアクセスモードを表すフラグである。
ファイルディスクリプタの属性として扱われる。

O_RDONLY	読込み専用でオープンされている
O_RDWR	読み書き用にオープンされている
O_WRONLY	書込み専用でオープンされている

ファイルアクセスモードは、mode_t 型のデータ型を持つ。
mode_t 型データからファイルアクセスモードを取り出すためのマスクは以下を用いる。

O_ACCMODE	ファイルアクセスモードマスク

□ファイルオフセット

ファイルの読み書きを行う際の現在位置(読み書き開始位置)を、ファイルオフセットと呼ぶ。
ファイルオフセットを表すデータ型として以下がある。

off_t	32 ビット整数でのオフセット
off64_t	64 ビット整数でのオフセット

4.3 サポートしない機能

T2EX のファイルマネージャは POSIX のファイル機能のサブセットをサポートしている。
T2EX は、プロセスが無くユーザという概念もないため、ファイルの所有者やアクセスできるユーザのグループといったものが存在しない。また、シンボリックリンクもサポートしない。
したがって、POSIX で規定されている以下の機能は提供しない。

ファイルのアクセス権および所有者・グループ関連
- umask()
- chown()
- fchown()
- lchown()

ハードリンク
- link()
- linkat()

シンボリックリンクおよび関連する機能
- symlink()
- symlinkat()
- readlink()
- lstat()

ファイルディスクリプタの複製
- dup()
- dup2()

ファイルロック
- flock()
- lockf()

その他
- chroot()

デバイス上のファイルが、ユーザやグループ情報およびそれらのアクセス権情報を持っている場合、そのようなファイルにアクセスし更新を行った場合、元のファイルが持っていたユーザ・グループ・アクセス権に関する情報が欠損してしまう可能性がある。
独自に記述したファイルシステム実装部が扱うファイルでユーザやグループなどの情報を利用したい場合には、fs_ioctl() 等を用いてそのファイルシステム実装部固有の機能として実装する必要がある。

4.4 API

ファイルマネージャのAPIコールの戻値が負の場合は、T2EX 拡張エラーコードを意味する。

4.4.1 fs_main - ファイルマネージャの初期化・終了

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER      er = fs_main(INT ac, UB* arg[]);
```

【パラメータ】

INT	ac	arg[] の要素数または負の値
UB*	arg[]	引数文字列の配列

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK　　正常終了

【解説】

ファイルマネージャを初期化(ac >= 0)または終了 (ac < 0) する。
初期化時は、arg[] に ac 個の文字列を引数として渡すことができる。
arg の内容は実装依存である。T2EXリファレンス実装では、引数文字列は使用していない。

4.4.2 fs_regist - ファイルシステム実装部の登録

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_regist(const char *fimpnm, const fs_fimp_t *fimp, void *info);
```

【パラメータ】

const char*	fimpnm	ファイルシステム実装部名
const fs_fimp_t	fimp	ファイルシステム実装部定義情報
void*	info	任意パラメータ

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_INVAL	不正なパラメータ
EX_EXIST	指定のファイルシステム実装部名が登録済み
EX_ENOBUFS	登録可能なファイルシステム実装部の数を超えた
その他	ファイルシステム実装部のregistfn()が返したエラーコード

【解説】

fs_regist() は、fimp で指定したファイルシステム実装部を fimpnm で指定した名前でシステムに登録する。
fimp はファイルシステム実装部の要求サービス関数を含むファイルシステム実装部構造体へのポインタである。
info はファイルシステム実装部の初期化関数 (fimp->registfn) に渡すパラメータで、ファイルシステム実装部ごとに定められた初期化情報を渡すために用いる。

fs_regist() は、ファイルシステム実装部の登録に成功した場合 E_OK を返す。
基本FATファイルシステム実装部は、ファイルマネージャの初期化を行う事で登録済み状態となる。
したがって、基本FATファイルシステム実装部を用いるには fs_regist() による登録は必要ない。

基本FATファイルシステム実装部以外のファイルシステム実装部を使用してFATファイルシステムを処理したい場合には、fimpnm に "FIMP_FAT" 以外の名前を指定した fs_regist() を実行してそのファイルシステム実装部を登録し、それを fs_attach() で接続して利用する。
あるいは、fs_unregist() により基本FATファイルシステム実装部を登録解除した後に、別のファイルシステム実装部を "FIMP_FAT" の名前で登録し、それを利用することも可能である。

fs_fimp_t は、ファイルシステム実装部の定義情報を含む構造体で、ユーザによるファイルシステム実装部の実装に関しては 4.5 節 ファイルシステム実装部 を参照。

【関連項目】

fs_unregist, fs_attach, fs_detach

4.4.3 fs_unregist - ファイルシステム実装部の登録解除

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_unregist(const char *fimpnm);
```

【パラメータ】

const char*	fimpnm	ファイルシステム実装部名

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_INVAL	不正なパラメータ
EX_NOENT	未登録のファイルシステム実装部名が指定された
EX_BUSY	指定のファイルシステム実装部は使用中
その他	ファイルシステム実装部のunregistfn()が返したエラーコード

【解説】

fs_unregist() は、fimpnm で指定した名前で登録済みのファイルシステム実装部の登録を解除する。
fspnm で指定した名前のファイルシステム実装部がデバイスと接続されている場合、登録解除に先立ち
fs_detach() によりその接続を解除しておく必要がある。

fs_unregist() は、ファイルシステム実装部の登録解除に成功した場合 E_OK を返す。
指定のファイルシステム実装部が fs_attach() によりデバイスと関連付けられている場合は EX_BUSY を返す。

【関連項目】

fs_regist, fs_attach, fs_detach

4.4.4 fs_attach - ファイルシステムの接続

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_attach(const char* devnm, const char* connm, iconst char *fimpnm, int flags, void *info);
```

【パラメータ】

const char*	devnm	接続するデバイス名
const char*	connm	接続ポイント
const char*	fimpnm	ファイルシステム実装部名
int	flag	接続フラグ
	DEV_FLAG_READONLY	読込み専用デバイス
	0	その他のデバイス
void*	info	任意パラメータ

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_INVAL	不正なパラメータ
EX_EXIST	指定の接続ポイントが登録済み
EX_ENOBUFS	接続可能な数を超えた
EX_NOENT	未登録のファイルシステム実装部名が指定された
EX_NOTSUP	サポートされていないファイルシステム実装部
EX_BUSY	デバイスが接続中
EX_IO	I/Oエラー
その他	ファイルシステム実装部のattachfn()が返したエラーコード

【解説】

fs_attach() は、devnm で指定したデバイスを fimpnm で指定した名前のファイルシステム実装部で処理されるものとして関連付け、システムのディレクトリツリーに connm の接続ポイントで接続する。

devnm はファイルシステム実装部が規定する形式でフォーマットされたデバイスの名称である。
デバイスを使用しないファイルシステム実装部の場合は NULL を指定できる。

connm はファイルシステム実装部を接続する接続ポイントである。
fs_attach() の実行後、"/<connm>/～" のパス名を用いることで、接続したデバイス内のファイルを特定できる。
例えば、connm として "usr" を指定した場合、接続したデバイス上のファイルは "/usr" というディレクトリ下に置かれる。

fimpnm は、登録済みのファイルシステム実装部名である。

flags は属性を付けて接続する場合に使用する。
ここに DEV_FLAG_READONLY を指定した場合、書込み可能なデバイスであっても読込み専用のデバイスとして接続する。
この属性を付けない場合は flags を 0 とする。

info はファイルシステム実装部の初期化関数（fimp->attachfn）に渡すためのパラメータである。
アプリケーションはファイルシステム実装部に固有の初期化情報を渡すためにこれを使用できる。
基本FATファイルシステム実装部を接続する場合は 0 を指定する。

fs_attach() は、接続に成功した場合 E_OK を返す。

fimpnm に "FIMP_AUTODETECT" を指定した場合、接続したデバイス内のデータ形式に応じて、
それを処理する登録済みのファイルシステム実装部が自動的に選択される。
ただし、現時点ではこの機能をサポートしないため、fimpnm に "FIMP_AUTODETECT" が指定された場合は EX_NOTSUP を返す。

【関連項目】

fs_detach

4.4.5 fs_detach - ファイルシステムの切り離し

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_detach(const char* connm);
```

【パラメータ】

const char*	connm	切断する接続ポイント

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_BUSY	デバイスが使用中
EX_FAULT	引数のアドレスが不正
EX_NOENT	未登録の接続ポイント

【解説】

fs_detach() は、connm で指定された接続されているデバイスの接続を解除する。
接続を解除しようとしているデバイス中にオープン中のファイルがある場合 EX_BUSY のエラーとなる。

【関連項目】

fs_attach

4.4.6 fs_open - ファイルのオープンまたは作成

【C言語インタフェース】

```
#include <t2ex/fs.h>

int fd = fs_open(const char* path, int oflags, mode_t mode);
```

【パラメータ】

const char*	path	オープンするファイルのパス名
int	oflags	ファイルまたはディレクトリのオープンフラグ
mode_t	mode	生成するファイルのファイルモード(O_CREAT指定時)

【リターンパラメータ】

int	fd	ファイルディスクリプタ またはエラーコード

【エラーコード】

EX_ACCES	アクセス権違反 - ファイルが存在し、oflag で指定した操作が不可 - ファイルが存在せず、親ディレクトリが書込み属性を持っていない
EX_EXIST	O_CREAT と O_EXCL が指定されたが、ファイルがすでに存在する
EX_INTR	fs_break() による中止
EX_INVAL	不正なパラメータ
EX_IO	I/Oエラー
EX_ISDIR	O_WRONLY または O_RDWR に対し、path がディレクトリである
EX_NAMETOOLONG	ファイル名が長すぎる - path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX) - path 全体の長さが長すぎる(最大PATH_MAX)
EX_NFILE	システムでオープン可能なファイル数を超えた
EX_NOENT	ファイルが存在しない - O_CREAT が指定されていない場合、ファイルが存在しない - O_CREAT が指定されていて、パス名のパス部分が見つからない - path の文字列が空
EX_NOSPC	デバイスの容量不足
EX_NOTDIR	ディレクトリではない - path のプレフィクス部にディレクトリではなものがある - O_DIRECTORY が指定されていて path がディレクトリではない
EX_ROFS	O_WRONLY, O_RDWR, O_CREAT, O_TRUNC が指定されたファイルが 読込み専用ファイルシステムにある

【解説】

path で指定されたパス名を持つファイルまたはディレクトリをオープンし、そのファイルディスクリプタを返す。
ファイルディスクリプタは、そのファイルに対するこの後の操作で、ファイルを参照するのに使用する。

ファイルディスクリプタは、現在オープンして使用されていない最小の値を返す。

ファイルの現在位置を示すファイルオフセットはファイルの先頭に設定される。

オープンフラグは以下のように指定する。

```
oflags := (O_RDONLY|O_RDWR|O_WONLY)
        | [O_APPEND] | [O_CREAT| [O_EXCL]] | [O_DIRECTORY] | [O_NONBLOCK]
        | [O_SYNC] | [O_TRUNC]
```

読み書きするモードとして以下のいずれかを指定する。

O_RDONLY
: 読込み専用にオープンする。

O_RDWR
: 読み書き用にオープンする。

O_WRONLY
: 書込み専用にオープンする。

さらに、以下の値は、論理和で追加で指定できる。

O_APPEND
: 書込み(fs_write)を行う前にファイルオフセットをファイル終端に設定する。

O_CREAT
: ファイルが存在しなければファイルを新規作成する。
ファイルが存在する場合、O_EXCL が指定されていないならばこのフラグは何の効果も無い。
ファイルを作成する場合のモードとして mode の値が使われる。

O_DIRECTORY
path がディレクトリでない場合、EX_NOTDIR のエラー

O_EXCL
O_CREAT とともに使用し、ファイルがすでに存在する場合エラーとなる。
ファイルが存在するかの検査およびファイルが存在しない場合の新規作成は、同じパス名に対して fs_open() を実行する別のタスクに対してアトミックである。
O_CREAT が指定されていない場合の結果は未定義である。

O_NONBLOCK
ノンブロッキング(non-blocking)オープンがサポートされている場合、ノンブロッキングモードでオープンする。
ノンブロッキングオープンした場合、デバイスが準備可能になる前に待つことなく fs_open() は終了する。その後の動作はデバイス依存である。

O_SYNC
書込み時には同期書込みの完了を待つ。すなわち、対象ファイルに対する書込み時には、
ディスクキャッシュだけでなく、物理デバイスへの書込みも行い、ディスクキャッシュと
物理デバイスが同期した状態になってから書込み処理を終了する。

O_TRUNC
ファイルが存在していて、通常ファイルであり、なおかつ O_RDWR または O_WRONLY でのオープンが成功した場合、ファイルのサイズは 0 に切り詰められる。
ファイルがコンソールのような端末デバイスファイルの場合は、何の効果も無い。
通常ファイル以外の場合は、その効果は実装依存とする。
O_RDWR または O_WRONLY のどちらも指定されていない場合の動作は未定義である。

【関連項目】

fs_close

4.4.7 fs_close - ファイルのクローズ

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_close(int fd);
```

【パラメータ】

int	fd	クローズするファイルディスクリプタ

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_BADF	fd が不正
EX_INTR	fs_break() による中止
EX_IO	I/Oエラー

【解説】

ファイルディスクリプタ fd で示される、オープン中のファイルまたはディレクトリをクローズする。
fd は解放され、その後の fs_open() で再利用できるようになる。

クローズ処理が fs_break() で中止された場合、fs_close() は EX_INTR を返し、fd はオープンされたままの状態となる。
クローズ処理中のファイルシステムへの読み書きで I/Oエラーが発生した場合、fs_close() は EX_IO を返す。このとき、fd はオープンされたままの状態となる。

【関連項目】

fs_open

4.4.8 fs_lseek, fs_lseek64 - ファイルの読み書きオフセット位置の変更

【C言語インタフェース】

```
#include <t2ex/fs.h>

off_t offs = fs_lseek(int fd, off_t offset, int whence);
off64_t offs64 = fs_lseek64(int fd, off64_t offset64, int whence);
```

【パラメータ】

int	fd	ファイルディスクリプタ
off_t	offset	ファイルオフセット
off64_t	offset64	ファイルオフセット(64ビット)
int	whence	変更指定
		SEEK_SET　絶対位置で指定
		SEEK_CUR　現在位置からの相対位置で指定
		SEEK_END　ファイル終端からの相対位置で指定

【リターンパラメータ】

off_t	offs	変更後のファイルオフセット またはエラーコード
off64_t	offs64	変更後のファイルオフセット(64ビット) またはエラーコード

【エラーコード】

EX_BADF	fd が不正
EX_INVAL	不正なパラメータ - ファイルオフセットが定義されないファイルディスクリプタを指定した - 変更結果のファイルオフセットが負の値になる - 変更結果のファイルオフセットがファイル終端位置を超えた
EX_OVERFLOW	変更結果のファイルオフセットが off_t または off64_t で表現できない

【解説】

オープンしているファイル fd のファイルオフセットを whence で指定した方法により変更する。

SEEK_SET
　　offset または offset64 の位置に変更する。

SEEK_CUR
　　現在位置 + (offset または offset64) の位置に変更する。

SEEK_END
　　ファイルサイズ + (offset または offset64) の位置に変更する。

コンソールデバイスのような位置変更の機能が無いファイルを指定すると EX_INVAL のエラーを返す。

ファイル終端を越える位置にファイルオフセットを設定することはできない。その場合、EX_INVAL のエラーを返す。

4.4.9 fs_read - ファイルの読込み

【C言語インタフェース】

```
#include <t2ex/fs.h>

int nb = fs_read(int fd, void* buf, size_t count);
```

【パラメータ】

int	fd	ファイルディスクリプタ
void*	buf	読込み用バッファ
size_t	count	読み込むバイト数

【リターンパラメータ】

int	nb	0以上の読み込んだバイト数 またはエラーコード
void*	buf	読み込まれたデータ

【エラーコード】

EX_AGAIN	待たずに読み込めるデータが無かった。(O_NONBLOCK フラグがセットされている場合)
EX_BADF	fd が不正
EX_INTR	fs_break() による中止
EX_IO	I/Oエラー
EX_ISDIR	fd がディレクトリである。
EX_OVERFLOW	通常ファイルで、count が正で、読込み開始位置がファイル終端を越えている
EX_NOBUFS	システム内のリソース不足
EX_NOMEM	メモリ不足

【解説】

fd で示されたファイルから count バイトのデータを buf で指定されたバッファに読み込む。

count が 0 の場合、fs_read() はエラーが発生していないか調べ、エラーが無ければ 0 を返す。

ディスク上のファイルのようにシークをサポートしているファイルでは、fd に対応したファイルオフセット位置から読み込みを開始する。ファイルオフセットは、実際に読み込んだバイト数だけ増加する。

コンソールのようなシークをサポートしないファイルでは、現在位置から読み込む。このようなファイルのファイルオフセットは未定義である。

現在のファイル終端を越えたデータ転送は発生しない。開始位置がファイル終端以降である場合 0 を返す。

count が SSIZE_MAX より大きい場合の動作は実装依存とする。

ノンブロッキング読み込みをサポートしているファイルからの読み込みで、現在利用可能なデータが無い場合:
・ファイル状態フラグの O_NONBLOCK がセットされていれば、EX_AGAIN のエラーを返す。
・ファイル状態フラグの O_NONBLOCK がクリアされていれば、いくらかのデータが利用可能になるまで fs_read() を呼び出した
　タスクは待たされる。

データを読み込む前に fs_break() により中止された場合、fs_read() は、EX_INTR を返す。
1バイトでもデータを読み込んだ後で、fs_break() により中止された場合、fs_read() は読み込んだバイト数を返し、ファイルオフセットもそのバイト数だけ増加する。

4.4.10 fs_write - ファイルへの書込み

【C言語インタフェース】

```
#include <t2ex/fs.h>

int nb = fs_write(int fd, const void* buf, size_t count);
```

【パラメータ】

int	fd	ファイルディスクリプタ
const void*	buf	書込みバッファ
size_t	count	書込みバイト数

【リターンパラメータ】

int	nb	書き込んだバイト数 またはエラーコード

【エラーコード】

EX_AGAIN	待たずに書き込めるデータが無かった。(O_NONBLOCK フラグがセットされている場合)
EX_BADF	fd が不正
EX_FBIG	ファイルサイズの制限を越える
EX_INTR	fs_break() による中止
EX_IO	I/Oエラー
EX_NOBUFS	システム内のリソース不足

EX_NOSPC　　デバイスの容量不足

【解説】

fd で示されたファイルに buf で指定されたバッファから count バイト書き込む。

count が 0 でファイルが通常ファイルの場合、fs_write() は、エラーが発生していないか検査し、エラーが無ければ 0 を返す。
count が 0 でファイルが通常ファイルでない場合の動作は未定義である。

シーク可能なファイルでは、データの書込みはそのファイルのファイルオフセットの位置から始める。
ファイルオフセットは、実際に書き込まれたバイト数だけ増加する。
通常ファイルでは、書き込んだデータの最後の位置がファイルのサイズ以上の場合、ファイルサイズはその位置 + 1 となる。

コンソールのようなシーク不可能なファイルでは、書込みは常に現在位置に対して行う。
そのようなデバイスに対するファイルオフセット値は未定義である。

ファイル状態フラグの O_APPEND がセットされている場合、書込み前にファイルオフセットはファイル終端に設定され、常にファイル終端から書き込まれる。

ファイルサイズの制限に達するまでの容量、またはメディアの物理的な空き容量を超えて書き込もうとすると、空き容量のバイト数だけ書き込んで正常終了する。さらに書き込もうとするとエラーとなる。

データを書き込む前に fs_break() により中止された場合、fs_write() は、EX_INTR を返す。
1バイトでもデータを書き込んだ後で fs_break() により中止された場合、fs_write() は書き込んだバイト数を返し、ファイルオフセットもそのバイト数だけ増加する。

count が SSIZE_MAX を超えている場合、結果は実装依存とする。

ノンブロッキング書込み可能なファイルへの書込みを試みた場合、データを直ちに書き込めない場合:
・ファイル状態フラグの O_NONBLOCK がセットされている場合、fs_write() はタスクを待たせない。いくらかのデータを書き込めた
　ならば fs_write() は書き込めたバイト数を返す。それ以外の場合 EX_AGAIN を返す。
・ファイル状態フラグの O_NONBLOCK フラグがクリアされている場合、fs_write() は、データが受け付けられるようになるまで
　呼び出したタスクを待たせる。

データを書き込んで正常終了した場合(count が 0 より大きい)、fs_write() はファイルの最終更新時刻および最終状態変更時刻を更新する。

4.4.11 fs_stat, fs_fstat_us, fs_stat_ms, fs_fstat, fs_stat_ms, fs_stat_us, fs_stat64, fs_stat64_ms, fs_stat64_us, fs_fstat64, fs_stat64_ms, fs_fstat64_us - ファイルの状態取得

【C言語インタフェース】

```
#include <t2ex/fs.h>

/* ファイルサイズが 32 ビット */
ER er = fs_stat(const char* path, struct stat* buf);
ER er = fs_stat_ms(const char* path, struct stat_ms* mbuf);
ER er = fs_stat_us(const char* path, struct stat_us* ubuf);

ER er = fs_fstat(int fd, struct stat* buf);
ER er = fs_fstat_us(int fd, struct stat_us* ubuf);
ER er = fs_fstat_ms(int fd, struct stat_ms* mbuf);


/* ファイルサイズが 64 ビット */
ER er = fs_stat64(const char* path, struct stat64* buf64);
ER er = fs_stat64_ms(const char* path, struct stat64_ms* mbuf64);
ER er = fs_stat64_us(const char* path, struct stat64_us* ubuf64);

ER er = fs_fstat64(int fd, struct stat64* buf64);
ER er = fs_fstat64_ms(int fd, struct stat64_ms* mbuf64);
ER er = fs_fstat64_us(int fd, struct stat64_us* ubuf64);
```

【パラメータ】

```
const char*             path            ファイルのパス名
int                     fd              ファイルディスクリプタ

struct stat*            buf             ファイル情報取得バッファ(time_t 形式)
struct stat_ms*         mbuf            ファイル情報取得バッファ(SYSTIM 形式)
struct stat_us*         ubuf            ファイル情報取得バッファ(SYSTIM_U 形式)

struct stat64*          buf64           ファイル情報取得バッファ(64ビットサイズ、time_t 形式)
struct stat64_ms*       mbuf64          ファイル情報取得バッファ(64ビットサイズ、SYSTIM 形式)
struct stat64_us*       ubuf64          ファイル情報取得バッファ(64ビットサイズ、SYSTIM_U 形式)
```

【リターンパラメータ】

```
ER                      er              エラーコード
struct stat*            buf             ファイル情報(time_t 形式)
struct stat_ms*         mbuf            ファイル情報(SYSTIM 形式)
struct stat_us*         ubuf            ファイル情報(SYSTIM_U 形式)

struct stat64*          buf64           ファイル情報(64ビットサイズ、time_t 形式)
struct stat64_ms*       mbuf64          ファイル情報(64ビットサイズ、SYSTIM 形式)
struct stat64_us*       ubuf64          ファイル情報(64ビットサイズ、SYSTIM_U 形式)
```

【エラーコード】

```
E_OK                    正常終了
EX_ACCES                path に含まれるディレクトリに検索許可属性の無いものがある
EX_IO                   I/Oエラー
EX_NAMETOOLONG          ファイル名が長すぎる
                         - path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX)
                         - path 全体の長さが長すぎる(最大PATH_MAX)
EX_NOENT                path に含まれるファイルが存在しないか path が空
EX_NOTDIR               path のプレフィクス部にディレクトリではないものがある
EX_OVERFLOW             st_size, st_ino, st_blocks が結果の型で表現できない値
```

【解説】

パス名 path または、ファイルディスクリプタ fd で指定されたファイルに関する情報を、個々の関数に応じて ubuf, mbuf, buf, ubuf64, mbuf64, buf64 で指定した領域に格納する。

path のファイルに対する読込み、書込み、実行許可属性は必要ない。

構造体 stat, stat_us, stat_ms, stat64, stat64_us, stat64_ms の構造を、以下に定義する。

```
        struct  stat    {
                dev_t           st_dev;                 /* ファイルがあるデバイスID */
                ino_t           st_ino;                 /* ファイル通し番号 */
                mode_t          st_mode;                /* ファイルモード */
                nlink_t         st_nlink;               /* リンクの数 */
                uid_t           st_uid;                 /* 所有者ID */
                gid_t           st_gid;                 /* グループID */
                dev_t           st_rdev;                /* デバイスタイプ */
                off_t           st_size;                /* ファイルサイズ(バイト数) */
                time_t          st_atime;               /* 最終アクセス時刻 */
                time_t          st_mtime;               /* 最終更新時刻 */
                time_t          st_ctime;               /* 最終状態変更時刻 */
                blksize_t       st_blksize;             /* I/Oブロックサイズ(バイト数) */
                blkcnt_t        st_blocks;              /* 割当ブロック数 */
                /* 実装依存の追加情報 */
        };

        stat_ms は stat の st_atime, st_mtime, st_ctime の宣言を以下に置き換えたものである。
                SYSTIM          st_atime;               /* 最終アクセス時刻(ミリ秒単位) */
                SYSTIM          st_mtime;               /* 最終更新時刻(ミリ秒単位) */
                SYSTIM          st_ctime;               /* 最終状態変更時刻(ミリ秒単位) */

        stat_us は stat の st_atime, st_mtime, st_ctime の宣言を以下に置き換えたものである。
                SYSTIM_U        st_atime_u;             /* 最終アクセス時刻(μ秒単位) */
                SYSTIM_U        st_mtime_u;             /* 最終更新時刻(μ秒単位) */
                SYSTIM_U        st_ctime_u;             /* 最終状態変更時刻(μ秒単位) */

        stat64 は、stat の st_size の宣言を以下に置き換えたものである。
                off64_t         st_size;                /* 64ビットのファイルサイズ(バイト数) */
```

stat64_ms は stat の st_size, st_atime, st_mtime, st_ctime の宣言を以下に置き換えたものである。

```
off64_t         st_size;                /* 64ビットのファイルサイズ(バイト数) */
SYSTIM          st_atime;               /* 最終アクセス時刻(ミリ秒単位) */
SYSTIM          st_mtime;               /* 最終更新時刻(ミリ秒単位) */
SYSTIM          st_ctime;               /* 最終状態変更時刻(ミリ秒単位) */
```

stat64_us は stat の st_size, st_atime, st_mtime, st_ctime の宣言を以下に置き換えたものである。

```
off64_t         st_size;                /* 64ビットのファイルサイズ(バイト数) */
SYSTIM_U        st_atime_u;             /* 最終アクセス時刻(μ秒単位) */
SYSTIM_U        st_mtime_u;             /* 最終更新時刻(μ秒単位) */
SYSTIM_U        st_ctime_u;             /* 最終状態変更時刻(μ秒単位) */
```

dev_t　　st_dev　　デバイスID
デバイスIDはデバイス登録時に動的に割り当てられるものであるため、固定的な値ではない。

ino_t　　st_ino;　　ファイル通し番号
システム内でファイルを識別するID。
ファイルシステム実装部依存の値を持つ。

mode_t　　st_mode;　　ファイルモード
ファイル属性の項目で示された、ファイルタイプおよびアクセスモード。

nlink_t　　st_nlink;　　リンクの数
ハードリンクを持つファイルシステムの場合、ハードリンクの数を返す。
ハードリンクを持たないファイルシステムの場合、1 を返す。

uid_t　　st_uid;　　所有者ID
所有者IDを持つファイルシステムの場合、そのID値を返す。
所有者IDを持たないファイルシステムの場合、0 を返す。

gid_t　　st_gid;　　グループID
グループIDを持つファイルシステムの場合、そのID値を返す。
グループIDを持たないファイルシステムの場合、0 を返す。

dev_t　　st_rdev;　　デバイスタイプ
ファイルがキャラクタまたはブロックスペシャルデバイスの場合、
ファイルシステム実装部が、そのIDを持っていればその値を返す。
持っていない場合は 0 を返す。

off_t　　st_size;　　ファイルサイズ(バイト数)
off64_t　　st_size;　　64ビットのファイルサイズ(バイト数)
通常ファイルの場合、ファイルのサイズをバイト数で返す。
他のタイプのファイルの場合は、このフィールドは未定となる。

time_t　　st_atime;　　最終アクセス時刻
SYSTIM　　st_atime;　　最終アクセス時刻(ミリ秒単位)
SYSTIM_U　　st_atime;　　最終アクセス時刻(μ秒単位)
ファイルが最後にアクセスされた時刻を型に応じた形式で返す。
実際に返る値の解像度は、ファイルシステムに依存する。

time_t　　st_mtime;　　最終更新時刻
SYSTIM　　st_mtime;　　最終更新時刻(ミリ秒単位)
SYSTIM_U　　st_mtime;　　最終更新時刻(μ秒単位)
ファイルが最後に更新された時刻を型に応じた形式で返す。
実際に返る値の解像度は、ファイルシステムに依存する。

time_t　　st_ctime;　　最終状態変更時刻
SYSTIM　　st_ctime;　　最終状態変更時刻(ミリ秒単位)
SYSTIM_U　　st_ctime;　　最終状態変更時刻(μ秒単位)
ファイルが最後に状態変更された時刻を型に応じた形式で返す。
実際に返る値の解像度は、ファイルシステムに依存する。

st_atime, st_mtime, st_ctime は、ファイルのタイムスタンプと呼ばれるもので、
その厳密な意味と精度はファイルシステムに依存する。
例えば、FATの仕様で定められた時刻の解像度は、更新時刻 2秒、アクセス時刻 1日である。
最終状態変更時刻は存在せず、最終更新時刻と同じものとなる。

blksize_t　　st_blksize;　　I/Oブロックサイズ(バイト数)
システム固有の、望ましいI/Oブロックのサイズ。

ファイルシステムのタイプによってはファイルによって異なることがある。

blkcnt_t st_blocks; 割当ブロック数
このファイルを割り当てるのに必要なブロック数。

4.4.12 fs_rename - ファイルの名前・位置を変更

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_rename(const char* oldpath, const char* newpath);
```

【パラメータ】

const char*	oldpath	変更元のファイル名またはディレクトリ名
const char*	newpath	変更後のファイル名またはディレクトリ名

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_ACCES	必要な書込み許可属性が無い
EX_BUSY	oldpath または newpath が使用中
EX_EXIST	newpath のディレクトリが空でない
EX_INVAL	oldpath が newpath のパス名の途中のパスである どちらかの引数の最後の要素が ″.″ または ″..″
EX_IO	I/Oエラー
EX_ISDIR	newpath がディレクトリであるが oldpath がファイル
EX_NAMETOOLONG	ファイル名が長すぎる - oldpath または newpath に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX) - パス名全体の長さが長すぎる(最大PATH_MAX)
EX_NOENT	ファイルが存在しない - oldpath が存在しない - newpath のパス部分が存在しない - oldpath か newpath のどちらかが空の文字列
E_NOSPC	newpath を含むディレクトリが拡張できない
EX_NOTDIR	ディレクトリではない - パスのプレフィクス部にディレクトリではないものがある - oldpath はディレクトリだが newpath がディレクトリでない
EX_ROFS	読込み専用ファイルシステム
EX_XDEV	oldpath と newpath が異なるファイルシステム上にあり、異なるファイルシステム間の移動が未サポートである

【解説】

ファイルの名前を変更する。oldpath で指定されるファイルを newpath で指定される名前に変更する。

oldpath がファイルの場合、newpath はディレクトリであってはいけない。
newpath のファイルがすでに存在する場合、まずはそのファイルが削除される。

oldpath がディレクトリの場合、newpath はファイルであってはならない。
newpath のディレクトリがすでに存在する場合、まずはそのディレクトリが削除される。

いずれかの引数の最後の要素が ″.″ または ″..″ の場合、fs_rename() は失敗する。

oldpath は newpath のパス名中の途中のディレクトリであってはならない。

oldpath および newpath を含むディレクトリに対して書込み許可属性が必要である。

oldpath がディレクトリの場合、oldpath に対する書込み許可属性が必要であり、newpath が存在するならば newpath に対しても書込み許可属性が必要である。

正常終了時、fs_rename() は、各引数の親ディレクトリの最終更新時刻および最終状態変更時刻を更新する。

EX_IO 以外のエラーで fs_rename() が失敗した場合、newpath のファイルには影響は無い。

4.4.13 fs_unlink - ファイルの削除

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_unlink(const char* path);
```

【パラメータ】

const char*	path	削除するファイルのパス名

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_ACCES	親ディレクトリの書込み許可属性が無い
EX_BUSY	ファイルが使用中
EX_ISDIR	path がディレクトリである
EX_NAMETOOLONG	ファイル名が長すぎる - path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX) - path 全体の長さが長すぎる(最大PATH_MAX)
EX_NOENT	path に含まれるファイルが存在しないか path が空
EX_NOTDIR	path のプレフィクス部にディレクトリではないものがある
EX_ROFS	読込み専用ファイルシステム

【解説】

path で指定された名前のファイルを削除する。
ディレクトリを削除することはできない。

正常終了時には、親ディレクトリの最終更新時刻と最終状態変更時刻を更新する。

【関連項目】
fs_rmdir

4.4.14 fs_fsync, fs_fdatasync - ファイルの同期

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_fsync(int fd);
ER er = fs_fdatasync(int fd);
```

【パラメータ】

int	fd	ファイルディスクリプタ

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_BADF	fd が不正
EX_INTR	fs_break() よる中止 (fs_fsync()の場合)
EX_INVAL	fd が同期可能なデバイスに対するファイルディスクリプタではない
EX_IO	I/Oエラー

他に fs_read() および fs_write() で定義されたエラーが返る。

【解説】

fd で指定された、既にオープンされているファイルに関して、現在キューイングされている全ての入出力処理要求を完了させ、ディスクキャッシュと物理デバイスを同期させる。同期の完了を待ってから戻る。

fs_fsync() はファイルのデータとメタデータすべてを同期するが、fs_fdatasync() は、最終更新時刻などのファイルのデータと直接関係しないメタデータは同期しない。

ファイルシステム実装部によっては、fs_fdatasync() と fs_fsync() は全く同じ場合がある。

4.4.15 fs_chdir, fs_fchdir - カレントディレクトリの変更

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_chdir(const char* path);
ER er = fs_fchdir(int fd);
```

【パラメータ】

const char*	path	ディレクトリのパス名
int	fd	ディレクトリに対するファイルディスクリプタ

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_ACCES	fd のディレクトリに対する検索許可属性が無い
EX_NAMETOOLONG	ファイル名が長すぎる - path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX) - path 全体の長さが長すぎる(最大PATH_MAX)
EX_NOTDIR	fd がディレクトリでない
EX_NOENT	path のディレクトリが存在しないか path が空
EX_INTR	fs_break() による中止
EX_IO	I/Oエラー

【解説】

fs_chdir() は、path で指定したパス名のディレクトリをカレントディレクトリに設定する。

fs_fchdir() は、fd のファイルディスクリプタで指定したオープン中のディレクトリをカレントディレクトリに設定する。
ディレクトリは、oflags に O_RDONLY を指定した fs_open() によりオープンすることができる。

T2EX ではシステム全体でカレントディレクトリは一つしか存在しない。

【関連項目】

fs_getcwd

4.4.16 fs_getcwd - 現在のカレントディレクトリの取得

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_getcwd(char* buf, size_t size);
```

【パラメータ】

char*	buf	ディレクトリ名の格納領域

size_t	size	buf のバイトサイズ

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_INVAL	size が 0
EX_RANGE	size が正だが、文字列のバイト数+1 よりも小さい
EX_ACCES	カレントディレクトリに対する検索許可属性がないか、 その上位ディレクトリに対する読込みまたは検索許可属性がない
EX_NOMEM	メモリ不足

【解説】

カレントディレクトリの絶対パス名を buf で示す領域に返す。
パス名は ″.″ または ″..″ の要素を含まない。

buf が NULL の場合、動作は未定である。

【関連項目】

fs_chdir, fs_fchdir

4.4.17 fs_creat - ファイルの作成

【C言語インタフェース】

```
#include <t2ex/fs.h>

int fd = fs_creat(const char* pathname, mode_t mode);
```

【パラメータ】

const char*	pathname	ファイルのパス名
mode_t	mode	ファイル作成モード

【リターンパラメータ】

int	fd	ファイルディスクリプタ

【エラーコード】

fs_open() の項を参照

【解説】

ファイルの生成、オープンを行う。
fs_open(pathname, O_WRONLY|O_CREAT|O_TRUNC, mode) を実行するのと等価である。

詳しくは fs_open() の項を参照。

【関連項目】

fs_open

4.4.18 fs_chmod,fs_fchmod - ファイルのモードを変更

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_chmod(const char *path, mode_t mode);
ER er = fs_fchmod(int fd, mode_t mode);
```

【パラメータ】

const char*	path	モード変更するファイルのパス名
int	fd	モード変更するファイルディスクリプタ
mode_t	mode	ファイルモード

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_ACCES	path に含まれるディレクトリに検索許可属性の無いものがある
EX_BADF	fd が不正
EX_NAMETOOLONG	ファイル名が長すぎる - path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX) - path 全体の長さが長すぎる(最大PATH_MAX)
EX_NOENT	path に含まれるファイルが存在しないか path が空
EX_NOTDIR	path のプレフィクス部にディレクトリではないものがある
EX_ROFS	読込み専用ファイルシステム

【解説】

fs_chmod() は、path で指定したパス名のファイルのアクセス許可に関するモードを mode で指定した値に変更する。
ファイルモードのうち、アクセス許可に関するビットのみを変更する。

各ビットの意味とその効果は、ファイルモードの項の説明に従う。
FATファイルシステムの場合、ファイルの書込み許可/禁止のみを設定する。

fs_fchmod() は、fd のファイルディスクリプタで指定したオープン中のファイルに関するアクセス許可を変更する。

【関連項目】
fs_open, fs_stat

4.4.19 fs_utimes, fs_utimes_ms, fs_utimes us - ファイルの時刻の変更

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_utimes(const char* path, const struct timeval tim[2]);
ER er = fs_utimes_ms(const char* path, const SYSTIM tim_m[2]);
ER er = fs_utimes_us(const char* path, const SYSTIM_U tim_u[2]);
```

【パラメータ】

const char*	path	パス名
const struct timeval	tim[2]	POSIX 形式のμ秒単位のシステム時刻
const SYSTIM	tim_m[2]	ミリ秒単位のシステム時刻
const SYSTIM_U	tim_u[2]	μ秒単位のシステム時刻

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_ACCES	path に含まれるディレクトリに検索許可属性の無いものがある
EX_INVAL	ファイルシステムでサポートされない時刻を指定した
EX_NAMETOOLONG	ファイル名が長すぎる - path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX) - path 全体の長さが長すぎる(最大PATH_MAX)
EX_ROFS	読込み専用ファイルシステム

【解説】

path で指定したファイルのアクセスおよび更新時刻を指定した時刻に設定する。
tim_u[0], tim_m[0], tim[0] は、新しいアクセス時刻を指定する。
tim_u[1], tim_m[1], tim[1] は、新しい更新時刻を指定する。

構造体 timeval は、以下のように定義されている。

```
struct timeval {
        long    tv_sec;         /* 秒 */
        long    tv_usec;        /* マイクロ秒 */
};
```

tim, tim_m, tim_u が NULL の場合、ファイルのアクセスおよび更新時刻は、ファイルシステム実装部がサポートしている
現在時刻を越えない最大の値に設定する。

完了時には、ファイルの最終状態変更時刻を更新する。

【関連項目】

fs_stat, fs_fstat, fs_stat64, fs_fstat64

4.4.20 fs_ioctl - デバイスの制御

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_ioctl(int fd, int request, ... /* arg */);
```

【パラメータ】

int	fd	ファイルディスクリプタ
int	request	要求コマンド

【リターンパラメータ】

ER	er	要求コマンドに応じた結果、 またはエラーコード

【エラーコード】

E_OK	正常終了
EX_BADF	fd が不正
EX_INVAL	request またはその後の引数が不正
EX_NOTTY	fd がキャラクタ型のスペシャルデバイスを参照していない

上記に加え、ファイルシステム実装部が request の種類に応じて適切なエラーを返す。

【解説】

fs_ioctl() は、request に応じたファイルまたはデバイスに関する種々の制御機能を実行する。
fd は、デバイスを参照するオープンされたファイルディスクリプタである。
request およびオプションの第三引数以後の引数は、fd に対応するファイルシステム実装部に渡され処理される。

arg は、対象となるデバイスで request を実行するのに必要とされる付加的な情報である。
arg の型は request の種類に依存するが、int 型かデバイス固有なデータ構造へのポインタである。

request と arg のタイプを以下に示す。

FIONBIO　　const int*
ディスクリプタに対するI/O動作のブロッキング/ノンブロッキングモードを引数の int* が指す値で設定する。
*arg == 0 の場合は、ブロッキングモードとなる(O_NONBLOCK 状態フラグがクリアされる)。
*arg != 0 以外の場合は、ノンブロッキングモードとなる(O_NONBLOCK 状態フラグがセットされる)。

FIONREAD　　int*
直ちに読み込める準備ができているバイト数を引数の int* が指す領域に返す。

FIONWRITE　　int*
ディスクリプタの送信キューに入っているバイト数を引数の int* が指す領域に返す。
それらのバイトは、ディスクリプタに書き込まれたデータで処理を待っている状態にある。
それらがどのように処理されるかはデバイスに依存する。

FIONSPACE　　int*
ディスクリプタの送信キューの空き容量を引数の int* が指す領域に返す。
この値は、送信キューのサイズからキューに保持されているデータのサイズを引いたものである。

上記以外の request でシステムで予約されていない値は、対応するファイルシステム実装部に引数とともに転送されそこで処理される。
すなわち、アプリケーションとファイルシステム実装部との通信手段を提供する。

ファイルシステム実装部によっては、メインエラーコードが EC_ERRNO 以外のエラーを返すことがある。これはデバイスドライバ側のエラーコードをそのまま伝えたい場合に利用する。

4.4.21 fs_fcntl - ファイルディスクリプタの操作

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_fcntl(int fd, int cmd, ... /* arg */);
```

【パラメータ】

int	fd	ファイルディスクリプタ
int	cmd	操作コマンド
	arg	コマンドに応じて必要な引数(可変個)

【リターンパラメータ】

ER	er	コマンドに応じた0以上の結果 またはエラーコード

【エラーコード】

EX_BADF	fd が不正
EX_INVAL	cmd の値が不正

上記に加え、ファイルシステム実装部がコマンドに応じて適切なエラーコードを返す。

【解説】

ファイルディスクリプタ fd に対して、cmd に指定した以下の操作を行う。

F_GETFL
fd のファイルディスクリプタに対するファイル状態フラグとアクセスモードを返す。
ファイルアクセスモードは、戻値を O_ACCMODE でマスクすることにより得られる。

F_SETFL
fd のファイルディスクリプタに対するファイル状態フラグを第三引数 arg の対応するビットで設定する。
第三引数 arg は int 型とする。
ファイルアクセスモードおよびファイル生成フラグに対応するビットは無視されファイル状態フラグが設定される。
arg のそれ以外のビットを変更できるが、その結果は未定である。

ファイル状態フラグとアクセスモードは、個々のファイルディスクリプタに結びついたもので、F_SETFL の設定は、同じファイルを別にオープンした異なるファイルディスクリプタに影響を与えない。

上記以外のコマンドおよび arg は、ファイルシステム実装部に転送され処理されるものとする。

4.4.22 fs_truncate, fs_ftruncate, fs_truncate64, fs_ftruncate64 - ファイルの切り詰め/拡大

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_truncate(const char* path, off_t length);
ER er = fs_ftruncate(int fd, off_t length);
ER er = fs_truncate64(const char* path, off64_t length64);
ER er = fs_ftruncate64(int fd, off64_t length64);
```

【パラメータ】

const char*	path	ファイルのパス名
int	fd	ファイルディスクリプタ
off_t	length	ファイルの長さ
off64_t	length64	ファイルの長さ(64ビットオフセット表現)

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_BADF	fd が書込みオープンされていない
EX_INTR	fs_break() による中止
EX_INVAL	length, length64 が負、または fd が不正
EX_BIG	length, length64 がファイルの最大長を超えている
EX_IO	I/Oエラー
EX_ISDIR	path または fd がディレクトリである
EX_NAMETOOLONG	ファイル名が長すぎる - path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX) - path 全体の長さが長すぎる(最大 PATH_MAX)
EX_NOENT	path に含まれるファイルが存在しないか path が空
EX_ROFS	読込み専用ファイルシステム

【解説】

path という名前のファイルか fd で参照されるファイルを length または length64 で指定された長さに変更する。

fd は書込みオープンされたファイルディスクリプタでなければならない。

ファイルが length または length64 より大きい場合、length または length64 を超えるデータは捨てられる。
ファイルが length または length64 より小さい場合、ファイルのサイズは拡大され増えた分は 0 で埋められる。

ファイルがオープンされている場合、ファイルオフセットは変更されない。

サイズが変更された場合ファイルの最終更新時刻および最終状態変更時刻が更新される。

fs_truncate64, fs_ftruncate64 は、64ビットオフセットでサイズ指定できる。

【関連項目】

fs_open

4.4.23 fs_sync - ファイルシステムの同期

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_sync(void);
```

【パラメータ】

無し

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_INTR	fs_break() による中止
EX_IO	I/Oエラー

【解説】

システムに接続されている全てのファイルシステムに対して、現在キューイングされている全ての入出力処理要求を完了させ、全てのディスクキャッシュを物理デバイスを同期させる。同期の完了を待ってから戻る。

fs_sync() が正常終了した後、システムに接続されている全てのファイルシステムの物理デバイスとの同期が保証される。

【関連項目】

fs_fsync, fs_fdatasync

4.4.24 fs_statvfs, fs_fstatvfs - ファイルシステムの統計情報

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_statvfs(const char* path, struct statvfs* buf);
ER er = fs_fstatvfs(int fd, struct statvfs* buf);
```

【パラメータ】

const char*	path	パス名
struct statvfs*	buf	ファイル統計情報
int	fd	ファイルディスクリプタ

【リターンパラメータ】

ER	er	エラーコード
struct statvfs*	buf	ファイル統計情報

【エラーコード】

E_OK	正常終了
EX_BADF	fd が不正(fs_fstatvfs)
EX_IO	I/Oエラー
EX_INTR	fs_break() による中止
EX_OVERFLOW	buf の構造体で表現できない値がある
EX_NAMETOOLONG	ファイル名が長すぎる - path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX) - path 全体の長さが長すぎる(最大 PATH_MAX)
EX_NOENT	path に含まれるファイルが存在しないか path が空
EX_NOTDIR	path のプレフィクス部にディレクトリではないものがある

【解説】

fs_statvfs_は path のファイルを含んだファイルシステムに関する情報を buf に返す。
fs_fstatvfs は fd のファイルディスクリプタが参照しているファイルを含んだファイルシステムに関する情報を buf に返す。

statvfs 構造体は以下の定義である。

```
struct statvfs {
        unsigned long   f_bsize         /* ファイルシステムのブロックサイズ */
        unsigned long   f_frsize        /* フラグメントのブロックサイズ */
        fsblkcnt_t      f_blocks        /* f_frsize 単位のファイルシステムの総ブロック数 */
        fsblkcnt_t      f_bfree         /* 空きブロック数 */
        fsblkcnt_t      f_bavail        /* b_free に等しい */
```

```
        fsfilcnt_t      f_files         /* ファイル総数 */
        fsfilcnt_t      f_ffree         /* 空きファイル数 */
        fsfilcnt_t      f_favail        /* f_ffree に等しい */
        unsigned long   f_fsid          /* ファイルシステムID */
        unsigned long   f_flag          /* f_flag 値のビットマスク */
        unsigned long   f_namemax       /* 最大ファイル名長 */
};
```

f_flag 値には以下のものがある

ST_RDONLY	読込み専用ファイルシステム
ST_NOSUID	ST_ISUID および ST_ISGID ファイルモードビットの意味をサポートしない (通常は 1)
ST_REMOVABLE	取り外し可能ファイルシステム
ST_MEMORY	メモリファイルシステム
ST_NETWORK	ネットワークファイルシステム

4.4.25 fs_mkdir - ディレクトリの作成

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_mkdir(const char* path, mode_t mode);
```

【パラメータ】

const char*	path	作成するディレクトリのパス名
mode_t	mode	ディレクトリの許可属性

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_ACCES	親ディレクトリの書込み許可属性がない
EX_EXIST	その名前のファイルがすでに存在する
EX_NAMELOOLONG	path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX)
EX_NOENT	path に含まれるファイルが存在しないか path が空
EX_NOSPC	デバイスの容量不足
EX_NOTDIR	path のプレフィクス部にディレクトリではないものがある
EX_ROFS	親ディレクトリが読込み専用ファイルシステムにある

【解説】

path で指定したパス名の新規ディレクトリを作成する。

新規ディレクトリのアクセス許可は mode により設定される。

T2EX には所有者、グループ、他人の概念が無いため、いずれかのビットがセットされていればそのアクセス権は許可とみなす。

読込み、書込み、および実行について、アクセス権が作用するかどうかはファイルシステムに依存する。

【関連項目】

fs_rmdir

4.4.26 fs_rmdir - ディレクトリの削除

【C言語インタフェース】

```
#include <t2ex/fs.h>

ER er = fs_rmdir(const char* path);
```

【パラメータ】

const char*	path	ディレクトリのパス名

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_BUSY	ディレクトリが使用中(ルートディレクトリまたは接続ポイント)
EX_NOTEMPTY	path が空のディレクトリではない
EX_INVAL	path が不正
EX_IO	I/Oエラー
EX_NAMELOOLONG	path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX)
EX_NOENT	path が存在しない
EX_NOTDIR	path のプレフィクス部にディレクトリではないものがある
EX_ROFS	読込み専用ファイルシステム

【解説】

path で指定したパス名のディレクトリを削除する。
ディレクトリは空でなければならない。

path がルートディレクトリか接続ポイントの場合は削除に失敗する。

正常終了時には、削除した親ディレクトリの最終更新時刻および最終状態変更時刻を更新する。

【関連項目】

fs_mkdir, fs_rename

4.4.27 fs_getdents - ディレクトリエントリの読み込み

【C言語インタフェース】

```
#include <t2ex/fs.h>

int nb = fs_getdents(int fd, struct dirent* buf, size_t bufsz);
```

【パラメータ】

int	fd	ディレクトリのファイルディスクリプタ
struct dirent*	buf	ディレクトリエントリ構造体を読み込むメモリ領域へのポインタ
size_t	bufsz	buf で指定したメモリ領域のサイズ(バイト数)

【リターンパラメータ】

int	nb	読み込んだサイズ(バイト数) またはエラーコード
struct dirent*	buf	読み込んだディレクトリエントリ

【エラーコード】

EX_BADF	fd が不正
EX_FAULT	引数のアドレスが不正
EX_INVAL	bufsz が小さ過ぎてエントリが一つも読み込めない。
EX_NOENT	そのようなディレクトリは存在しない
EX_NOTDIR	ファイルディスクリプタ fd がディレクトリを参照していない

【解説】

fs_getdents() は、fd で指定したディレクトリからディレクトリエントリを読み込む。
fd は、fs_open() でオープンしたディレクトリのファイルディスクリプタである。
buf は、ディレクトリエントリを読み込むためにアプリケーションが確保したメモリ領域へのポインタである。
bufsz は、buf で指定したメモリ領域のサイズ(バイト数)である。
bufで指定したメモリ領域に、bufsz を超えない範囲で一つ以上のディレクトリエントリを読み込み、読み込んだサイズ(バイト数)を返す。ディレクトリの末尾に達している場合は、0 を返す。

ディレクトリエントリ構造体（dirent）は以下の定義である。

```
struct dirent {
        ino_t           d_ino;                  /* ファイルの内部番号 */
        unsigned short  d_reclen;               /* 本エントリのバイトサイズ */
        char            d_name[NAME_MAX+1];     /* エントリの名前 */
};
```

一つのディレクトリエントリは可変長であり、そのサイズは d_reclen で示される。
エントリの先頭アドレスに d_reclen を加えたアドレスが、次のエントリの先頭アドレスになる。
読み込んだディレクトリエントリの d_reclen の合計値が戻値となる。

※ 通常、ディレクトリエントリを読み込むには、この関数は使うべきではない。
　標準C互換ライブラリの readdir_r ライブラリ関数を使用すべきである。

4.4.28 fs_break - ファイル管理操作の中止

【C言語インタフェース】

```
#include <t2ex/fs.h>

int ntsk = fs_break(ID tskid);
```

【パラメータ】

ID	tskid	対象タスクID

【リターンパラメータ】

int	ntsk	待ちを解除したタスクの数 またはエラーコード

【エラーコード】

E_ID	タスクIDが不正(負または TMaxTskId を超えている)
E_NOEXS	タスクIDのタスクが存在しない

【解説】

ファイルマネージャの呼び出しに伴う tskid で指定したタスクの待ち状態を解除する。

対象タスクがファイルマネージャのAPIコールの実行の途中で待ち状態に入っていた場合、その待ち状態は即座に解除され、実行の途中であったAPIコールはEX_INTR を返す。

tskid が TSK_ALL(= (-1)) の場合、全てのタスクを対象として前述の処理を行う。

fs_break() は、正常終了した場合、待ちを解除したタスクの数を返す。

すべての対象タスクがファイルマネージャのAPIコールの実行の途中の待ち状態ではなかった場合、fs_break() は 0 を返す。

fs_break() によるタスクの待ち解除要求はキューイングされない。すなわち tskid で指定されたタスクが待ち状態でなかった場合、その後に待ち状態に入っても fs_break() の実行による影響はない。

4.5 ファイルシステム実装部

4.5.1 概要

T2EXでは、指定したデバイス上のファイルシステムを処理するプログラムコードをユーザが独自に定義できる機能を提供している。この機能により定義されたプログラムコードを「ファイルシステム実装部」と呼ぶ。ファイルマネージャのAPIコールの対象となったファイルのファイルシステム形式や、それに対する具体的な処理内容は、ファイルシステム実装部によって定められる。

T2EXには、最初から登録済みのファイルシステム実装部として、FATファイルシステム形式を扱う「基本FATファイルシステム実装部」と、標準入出力を扱う「基本コンソールファイルシステム実装部」がある。

ファイルシステム実装部を登録し、デバイス中のファイルシステムやファイルを操作する手順は以下のようになる:

1. 利用したいファイルシステム形式やデバイスに対応した各種の処理を行う関数群を、ファイルシステム実装部として作成する。これらの関数群は、T2EXのファイルマネージャから呼び出される。

2. fs_regist() により、ファイルシステム実装部の関数群のポインタ列を、ファイルシステム実装部の名称とともにシステムに登録する。
   このとき、ファイルシステム実装部の登録関数が呼び出される。

   ※ 基本FATファイルシステム実装部、および基本コンソールファイルシステム実装部に関しては、fs_main() の実行により、ここまでの処理が自動的に行われる。したがって、基本FATファイルシステム実装部と基本コンソールファイルシステム実装部のみを用いる場合には、上記の 1. と 2. の手順は不要である。

3. fs_attach() により、ファイルシステムの置かれたデバイスと、そのファイルシステムを扱うファイルシステム実装部との関連付けを行い、そのファイルシステムを指定した接続ポイントに接続する。
   このとき、ファイルシステム実装部のデバイス接続関数が呼び出される。

4. ファイルマネージャの各種 API コールの実行により、ファイルシステムやファイルを操作する。
   このとき、各種 API コールに対応したファイルシステム実装部の要求サービス関数が呼び出される。

デバイスを切り離して、ファイルシステム実装部の登録を解除する手順は以下のようになる:

1. そのデバイス上のすべてのオープン中のファイルをクローズする。
   fs_sync() を実行して、デバイスに関して全てのデータが書き込まれるようにする。

2. fs_detach() により、指定した接続ポイントに接続されているデバイスを切り離す。
   このとき、ファイルマネージャからファイルシステム実装部のデバイス切断関数が呼び出される。

3. そのファイルシステム実装部を用いて接続した全てのデバイスに関して 2. の操作を行う。

4. fs_unregist() により指定した名前のファイルシステム実装部の登録をシステムから解除する。
   このとき、ファイルシステム実装部の登録解除関数が呼び出される。

4.5.2 ファイルシステム実装部の構成

ファイルシステム実装部は、以下に示す関数群から構成される。

- 要求サービス関数 (reqfn)
- 登録関数 (registfn)
- 登録解除関数 (unregistfn)
- デバイス接続関数 (attachfn)
- デバイス切断関数 (detachfn)
- スタートアップ関数 (startupfn)
- クリーンアップ関数 (cleanupfn)
- ブレーク関数 (breakfn)

これらの関数はすべてファイルマネージャから呼び出され、処理に成功した場合は正常終了として E_OK を返し、失敗した場合はその原因や理由に応じて適切な T2EX 拡張エラーコード(EX_xxx)を返す。

これらの関数に対して、ファイルマネージャから渡されるパス名などの文字列の文字コードは UTF-8 形式であるため、デバイス上のファイルシステムで使用する文字コードが異なるときは、これらの関数内で文字コードの変換を行う必要がある。

ファイルマネージャはサブシステムとして実装されるため、これらの関数はファイルマネージャの API コールを実行したアプリケーションタスクのコンテキストで準タスク部として実行される。
ただし、fs_break()はタスク独立部からも実行可能であり、その場合、fs_break()の処理で呼び出されるブレーク関数はタスク独立部として実行される。

ファイルシステム実装部の各関数では、アプリケーションタスクのコンテキストで準タスク部として実行されるので、予期しない動作を回避するために、アプリケーションタスク内で使用される可能性のある以下の T-Kernel 2.0 API を使用してはいけない。

たとえば、アプリケーションタスクがファイルマネージャのAPIを実行している別のタスクを起床するために tk_wup_tsk() を呼び出したとき、ファイルマネージャーAPI内で tk_slp_tsk()を実行していると誤って起床してしまう。

```
tk_slp_tsk()
tk_slp_tsk_u()
tk_wup_tsk()
tk_can_wup()
```

スタートアップ関数、クリーンアップ関数、およびブレーク関数は省略できるが、それ以外の関数はいずれも必須である。

○要求サービス関数

```
ER      (*reqfn)(fimp_t *req);
```

本関数は、fs_main(), fs_regist(), fs_unregist(), fs_attach(), fs_detach(), fs_break() を除くすべてのファイルマネージャの API コールの処理で呼び出され、req で指定された各種のファイル操作を実行する。fimp_t データ型、および要求サービス関数の詳細については後述する。

ファイルマネージャの APIコールが複数のタスクから同時に実行された場合、要求サービス関数もそれぞれのタスクのコンテキストで準タスク部として同時に呼び出されるため、必要に応じて要求サービス関数内で排他制御を行う必要がある。

○登録関数

```
ER      (*registfn)(fimpinf_t *fimpinf, void *info);
```

本関数は、fs_regist() の処理で呼び出され、ファイルシステム実装部自体のデバイスに依存しない部分の初期化を行う。

fimpinf_t データ型については後述するが、fimpinf->fimpnm は fs_regist() で指定されたファイルシステム実装部名である。
fimpinf->fimpsd は、ファイルシステム実装部が自由に使えるデータであり、本関数内で適当な値を設定して、ファイルシステム実装部の他の関数で利用することができる。
通常は、ファイルシステム実装部が必要とする固有データを格納するための領域を獲得して、そのポインタを設定する。

fimpinf は、fs_regist()の実行のたびに新しく生成される。同じファイルシステム実装部を複数回 fs_regist() した場合、それぞれ fimpinf の値は異なる。

info は fs_regist() で指定された info そのものである。

○登録解除関数

```
ER      (*unregistfn)(fimpinf_t * fimpinf);
```

この関数は、fs_unregist() の処理で呼び出され、ファイルシステム実装部自体のデバイスに依存しない部分の終了処理を行う。

fimpinf は、登録関数に渡された fimpinf と同じ値であり、fimpinf->fimpsd に固有データの格納のためのメモリ領域を獲得していた場合は、その領域を解放する必要がある。

○デバイス接続関数

```
ER      (*attachfn)(coninf_t * coninf, void *info);
```

この関数は、fs_attach() の処理で呼び出され、デバイスの初期化およびデバイスの接続処理を行う。

coninf_t データ型については後述するが、coninf->fimpinf は登録関数に渡された fimpinf と同じ値である。

coninf->devnm は fs_attach() で指定されたデバイス名 devnm である。ファイルマネージャではデバイスに関してのチェックは行わないため、本関数内で指定されたデバイスに関して、その存在やファイルシステム形式などの適切なチェックを行う必要がある。

coninf->dflags には、fs_attach() で指定された flags の値が入るが、デバイス種別に関する追加情報があれば、本関数内で元の値との論理和で設定し、ファイルマネージャに通知する必要がある。

coninf->connm は fs_atttach() で指定された接続ポイント connm である。

coninf->consd は、ファイルシステム実装部が自由に使えるデータであり、本関数内で適当な値を設定して、ファイルシステム実装部の他の関数で利用することができる。
通常は、ファイルシステム実装部が必要とする接続ごとの固有データを格納するためのメモリ領域を獲得して、そのポインタを設定する。

coninf は、fs_attach()の実行のたびに新しく生成される。同じデバイス、ファイルシステム実装部を複数回 fs_attach() した場合、それぞれ coninf の値は異なる。

info は fs_attach() で指定された info そのものである。

○デバイス切断関数

```
ER      (*detachfn)(coninf_t *coninf);
```

本関数は、fs_detach() の処理で呼び出され、接続したデバイスの切断および終了処理を行う。

キャッシュにデバイスに書込むべきデータが残っている場合は書込み処理を行う。

coninf は、デバイス接続関数に渡された coninf と同じ値であり、coninf->consd に固有データの格納領域を獲得していた場合は、その領域を解放する必要がある。

○スタートアップ関数

```
ER      (*startupfn)(coninf_t *coninf, ID resid);
```

本関数は、T-Kernel2.0 の API コール tk_sta_ssy() の実行による、ファイルマネージャのスタートアップ関数の処理で呼び出され、resid で指定されたリソース管理ブロックの初期化処理を行う。
通常、T2EX システムではリソース管理ブロックは使用しないため、本関数は省略される。

○クリーンアップ関数

```
ER      (*cleanupfn)(coninf_t *coninf, ID resid);
```

本関数は、T-Kernel2.0 の API コール tk_cln_ssy()の実行による、ファイルマネージャのクリーンアップ関数の処理で呼び出され、resid で指定されたリソース管理ブロックの解放処理を行う。
通常、T2EX システムではリソース管理ブロックは使用しないため、本関数は省略される。

○ブレーク関数

```
ER      (*breakfn)(coninf_t *coninf, ID tskid, BOOL set);
```

本関数は、fs_break() の処理で呼び出される。

最初に set == TRUE として呼び出されるので、tskid で指定されたタスクの未完了の要求サービス関数の実行を中断して速やかに終了させる。中断された要求サービス関数が戻る時点で、再度 set == FALSE として呼び出されるので、tskid で指定されたタスクの内部中断状態を初期化する。

実行の途中で待ち状態に入っていた場合は、即座に待ちを解除し、中断された要求サービス関数は EX_INTR のエラーコードを返す。ただし、ファイルからの読込み/書込みの要求サービス関数で、すでに読込み/書込み済みのデータがある場合は、正常動作とみなし、読込んだ/書込んだバイト数を格納して E_OK を返す。

本関数は常に E_OK を返す。

coninf は、デバイス接続関数に渡された coninf と同じ値である。

本関数は、準タスク部だけでなく、タスク独立部からも呼び出されるため、どちらのコンテキストでも正しく実行できなくてはいけない。

ファイルシステム実装部のすべての要求サービス関数が短時間で終了し、長時間または不定の待ちに入らない場合は、本関数を省略してもよい。

4.5.3 データ型

□ファイルシステム実装部定義 (fs_fimp_t)

ファイルシステム実装部を定義する構造体であり、fs_regist() の引数として使用される。

```
/* ファイルシステム実装部定義 */
typedef struct {
        ER      (*reqfn)(fimp_t * req);                                 /* 要求サービス関数 */
        ER      (*registfn)(fimpinf_t * fimpinf, void *info);           /* 登録関数 */
```

```
        ER      (*unregistfn)(fimpinf_t * fimpinf);                     /* 登録解除関数 */
        ER      (*attachfn)(coninf_t * coninf, void *info);             /* デバイス接続関数 */
        ER      (*detachfn)(coninf_t * coninf);                         /* デバイス切断関数 */
        ER      (*startupfn)(coninf_t * coninf, ID resid);              /* スタートアップ関数 */
        ER      (*cleanupfn)(coninf_t * coninf, ID resid);              /* クリーンアップ関数 */
        ER      (*breakfn)(coninf_t * coninf, ID tskid, BOOL set);      /* ブレーク関数 */
        int     flags;                                                  /* ファイルシステム実装部種別
*/
        int     priority;                                               /* 優先度 */
} fs_fimp_t;
```

○要求サービス関数（reqfn）
○登録関数（registfn）
○登録解除関数（unregistfn）
○デバイス接続関数（attachfn）
○デバイス切断関数（detachfn）
○スタートアップ関数（startupfn）
○クリーンアップ関数（cleanupfn）
○ブレーク関数（breakfn）

ファイルシステム実装部の各関数へのポインタを指定する。
スタートアップ関数、クリーンアップ関数、およびブレーク関数を省略する場合は NULL を指定する。

○ファイルシステム実装部種別（flags）

ファイルシステム実装部の種別を以下のビットフラグの論理和で指定する。

FIMP_FLAG_READONLY	読込み専用ファイルシステム
FIMP_FLAG_MEMORY	メモリファイルシステム
FIMP_FLAG_NETWORK	ネットワークファイルシステム
FIMP_FLAG_64BIT	64bitファイルサイズ対応ファイルシステム
FIMP_FLAG_USEABORT	fs_break() 対応ファイルシステム
0	その他

FIMP_FLAG_USEABORT を指定した場合は、ブレーク関数を省略することはできない。

○優先度（priority）

ファイルシステム実装部の優先度を8段階（最高：１～ 最低：8）で指定する。
ファイルマネージャは、この優先度順にファイルシステム実装部のスタートアップ関数を呼び出す。
スタートアップ関数が NULL に設定された場合は、この値は参照されない。

□ファイルシステム実装部情報（fimpinf_t）

ファイルシステム実装部が動作に必要な情報を管理するための構造体であり、ファイルシステム実装部のすべての関数に渡される。

```
/* ファイルシステム実装部情報 */
typedef struct {
        char    fimpnm[L_FIMPNM+1];     /* ファイルシステム実装部名 */
        void *  fimpsd;                 /* ファイルシステム実装部固有データ */
} fimpinf_t;
```

○ファイルシステム実装部名（fimpnm）

fs_regist() で指定されたファイルシステム実装部名。ファイルシステム実装部はこの値を変更してはいけない。

○ファイルシステム実装部固有データ（fimpsd）

ファイルシステム実装部が自由に使えるデータであり、ファイルマネージャではこの値は使用しない。

□接続情報（coninf_t）

ファイルシステム実装部が接続ごとに必要な情報を管理するための構造体であり、ファイルシステム実装部の初期化関数、および終了関数を除くすべての関数に渡される。

```
/* 接続情報 */
typedef struct {
        fimpinf_t *     fimpinf;                /* ファイルシステム実装部情報 */
        UB              devnm[L_DEVNM+1];       /* デバイス名 */
        int             dflags;                 /* デバイス種別 */
```

```
        char            connm[L_CONNM+1];       /* 接続ポイント名 */
        void *          consd;                  /* 接続固有データ */
} coninf_t;
```

○ファイルシステム実装部情報(fimpinf)

fs_regist()で生成されたファイルシステム実装部情報へのポインタ。

○デバイス名(devnm)

fs_attach()で指定されたデバイス名。ファイルシステム実装部はこの値を変更してはいけない。

○デバイス種別(dflags)

デバイスの種別を表す以下のビットフラグの論理和である。

DEV_FLAG_READONLY	読込み専用デバイス
DEV_FLAG_REMOVABLE	リムーバブルデバイス
DEV_FLAG_MEMORY	メモリデバイス
0	その他

fs_attach() で指定された flags の値が設定されるが、デバイス接続関数の処理で追加設定される場合がある。

○接続ポイント名(connm)

fs_attach() で指定された接続ポイント名。ファイルシステム実装部 はこの値を変更してはいけない。

○接続固有データ(consd)

coninf->consd は、ファイルシステム実装部が自由に使えるデータであり、ファイルマネージャではこの値は使用しない。

□ファイルオープン識別子 (fid_t)

ファイルシステム実装部においてオープンしたファイル/ディレクトリを識別するための数値データ型。

ファイルオープン識別子の値は、ファイルシステム実装部の実装方法に依存するが、同じファイル/ディレクトリを多重にオープンした場合、それぞれのファイルオープン識別子はすべて同じ値でなければいけない。

4.5.4 要求サービス関数

要求サービス関数は、fs_regist(), fs_unregist(), fs_attach(), fs_detach(), fs_break() を除くすべてのファイル管理の APIコールの処理で呼び出され、指定された各種のファイル操作を実行する。

```
        ER      (*reqfn)(fimp_t *req);
```

fimp_t は、要求ファイル操作の内容を示す以下の共用体である。

```
/* ファイル処理要求共用体 */
union fimp {
        struct {                                /* 要求サービス共通構造 */
                int             r_code;         /* 要求サービス機能コード */
                coninf_t *      coninf;         /* 接続情報へのポインタ */
        } com;
                                                /* 要求サービス機能コード別パラメータ */
        struct fimp_open        r_open;                 /* FIMP_OPEN 用 */
        struct fimp_close       r_close;                /* FIMP_CLOSE 用 */
        struct fimp_read64      r_read64;               /* FIMP_READ64 用 */
        struct fimp_write64     r_write64;              /* FIMP_WRITE64 用 */
        struct fimp_ioctl       r_ioctl;                /* FIMP_IOCTL 用 */
        struct fimp_fsync       r_fsync;                /* FIMP_FSYNC 用 */
        struct fimp_truncate64  r_truncate64;           /* FIMP_TRUNCATE64 用 */
        struct fimp_ftruncate64 r_ftruncate64;          /* FIMP_FTRUNCATE64 用 */
        struct fimp_unlink      r_unlink;               /* FIMP_UNLINK 用 */
        struct fimp_rename      r_rename;               /* FIMP_RENAME 用 */
        struct fimp_chmod       r_chmod;                /* FIMP_CHMOD 用 */
        struct fimp_fchmod      r_fchmod;               /* FIMP_FCHMOD 用 */
        struct fimp_mkdir       r_mkdir;                /* FIMP_MKDIR 用 */
        struct fimp_rmdir       r_rmdir;                /* FIMP_RMDIR 用 */
        struct fimp_chdir       r_chdir;                /* FIMP_CHDIR 用 */
        struct fimp_fchdir      r_fchdir;               /* FIMP_FCHDIR 用 */
```

```
        struct fimp_getdents    r_getdents;             /* FIMP_GETDENTS 用 */
        struct fimp_fstatvfs    r_fstatvfs;             /* FIMP_FSTATVFS 用 */
        struct fimp_statvfs     r_statvfs;              /* FIMP_STATVFS 用 */
        struct fimp_sync        r_sync;                 /* FIMP_SYNC 用 */
        struct fimp_utimes_us   r_utimes_us;            /* FIMP_UTIMES_US 用 */
        struct fimp_fcntl64     r_fcntl64;              /* FIMP_FCNTL64 用 */
        struct fimp_stat64_us   r_stat64_us;            /* FIMP_STAT64_US 用 */
        struct fimp_fstat64_us  r_fstat64_us;           /* FIMP_FSTAT64_US 用 */
};
typedef union fimp fimp_t;
```

ファイル処理要求共用体は、ファイルマネージャの実装に依存して、上記以外のメンバを含む場合があるが、それらのメンバはファイルマネージャで使用するものであり、ファイルシステム実装部で参照および変更してはいけない。

要求サービス機能コード(r_code) は、ファイルマネージャの APIコールに対応した、ファイル操作を区別するコードであり、FIMP_OPEN ～ FIMP_STAT64_US のいずれかの値が入る。

以下に、各要求サービス機能コードごとのパラメータおよび要求サービス関数の動作について記述する。

・要求サービス関数に渡されるファイル/ディレクトリのパス名は、すべて ″/″ から始まる絶対パス名であり、″.″ や ″..″ は含まれない。
・要求サービス関数の説明において、あるポインタ p が指す領域の内容を *p と記述する。
・パラメータのコメントでは以下の表記を用いる。ポインタの場合はポインタが指す先が対象。
　(in)　　入力　: 参照のみで変更されないパラメータ。
　(out)　 出力　: 変更(設定)のみで参照されないパラメータ。
　(in/out) 入出力: 参照かつ変更(設定)される/される場合があるパラメータ。

□FIMP_OPEN

```
struct fimp_open {
        int             r_code;         /* (in) = FIMP_OPEN */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        const char *    path;           /* (in) ファイルパス名 */
        int             oflags;         /* (in) オープンフラグ */
        mode_t          mode;           /* (in) ファイルモード */
        fid_t *         fid;            /* (out) ファイルオープン識別子へのポインタ */
};
```

path で指定されたファイルまたはディレトリをオープンする。
path で指定されたファイルが存在せず、oflags に O_CREAT が指定された場合は、指定されたパス名のファイルを新規に生成する。

oflags には fs_open() の oflags で指定された値が入る。oflags にはファイルアクセスモードの O_RDONLY, O_WRONLY, O_RDWR の何れかを含むほか、ファイル生成フラグの O_CREAT, O_EXCL, O_TRUNC や、ファイル状態フラグの O_APPEND 等を含む場合もある。

mode には fs_open() の mode で指定された値が入る。mode は、指定したファイルが存在せず、oflags に O_CREAT が指定されている場合のみ有効であり、新規に生成するファイルのアクセス許可フラグとなる。

所有者やグループおよび権限のないファイルシステムの場合、mode の各ビットの解釈はファイルシステム実装部に依存する。
例えば、所有者の機能が無い場合に S_IWUSR が指定された場合、ファイルに書込み権限を付与するか、エラーとするかは、ファイルシステム実装部に依存する。
また、書込み可能なファイルに対して、FIMP_STAT64_US, FIMP_FSTAT64_US が返す st_mode に S_IWUSR, S_IWGRP, S_IWOTH のどの属性を設定するかはファイルシステム実装部に依存する。

オープンに成功した場合、オープンしたファイルを識別するためのファイルオープン識別子を *fid に格納する。
ファイルオープン識別子は、以降の要求サービス関数でオープンしたファイルを指定するパラメータとして使用される。

□ FIMP_CLOSE

```
struct fimp_close {
        int             r_code;         /* (in) = FIMP_CLOSE */
        coninf_t*       coninf;         /* (in/out) 接続情報へのポインタ */
        fid_t           fid;            /* (in) ファイルオープン識別子 */
        int             oflags;         /* (in) オープンフラグ */
};
```

fid で指定されたオープン中のファイル/ディレクトリをクローズする。

□ FIMP_READ64

```
struct fimp_read64 {
        int             r_code;         /* (in) = FIMP_READ64 */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        fid_t           fid;            /* (in) ファイルオープン識別子 */
        int             oflags;         /* (in) オープンフラグ */
        void *          buf;            /* (out) 読込みデータバッファへのポインタ */
        size_t *        len;            /* (in/out) 読込みバイト数/読込んだバイト数 へのポインタ */
        off64_t *       off;            /* (in) 読込み前ファイルオフセットへのポインタ */
        off64_t *       retoff;         /* (out) 読込み後ファイルオフセットへのポインタ */
};
```

fid で指定されたオープン中のファイルの *off で指定されたファイルオフセットから *len で指定されたバイト数のデータを *buf に読み込む。

実際に読込んだバイト数を *len に格納し、読込み後のファイルオフセットを *retoff に格納して戻る。
off と retoff が、同じ値のポインタであっても正しく動作しなくてはいけない。

ファイル終端に達している場合は、*len に 0 を格納して、E_OK を返す。
ファイル終端に達していない場合で、1バイトも読み込めなかった場合は、適切なエラーコードを返す。

ファイルがディレクトリの場合、ディレクトリの内容を読み込む。また、接続ポイントである場合も同様に、接続されたデバイスのルートディレクトリの内容を読み込む。

4GB までのファイルサイズしか扱えないファイルシステム実装部において、*off に 4GB 以上の値が設定された場合、EX_OVERFLOW のエラーコードを返す。

oflags は FIMP_OPEN が呼び出されたときの値から変更されている場合がある。FIONBIO を指定した fs_fcntl() が実行された場合には、oflags の O_NONBLOCK ビットが変更されることになる。oflags をパラメータとする他の要求サービス関数でも同様である。

□ FIMP_WRITE64

```
struct fimp_write64 {
        int             r_code;         /* (in) = FIMP_WRITE64 */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        fid_t           fid;            /* (in) ファイルオープン識別子 */
        int             oflags;         /* (in) オープンフラグ */
        void *          buf;            /* (in) 書込みデータバッファへのポインタ */
        size_t *        len;            /* (in/out) 書込みバイト数/書込んだバイト数 へのポインタ */
        off64_t *       off;            /* (in) 書込み前ファイルオフセットへのポインタ */
        off64_t *       retoff;         /* (out) 書込み後ファイルオフセットへのポインタ */
};
```

id で指定されたオープン中のファイルの *off で指定されたファイルオフセットから *len で指定されたバイト数のデータを *buf から書き込む。

実際に書込んだバイト数を *len に格納し、書込み後のファイルオフセットを *retoff に格納して戻る。
off と retoff が、同じ値のポインタであっても正しく動作しなくてはいけない。

ファイルに1バイトも書込むことができなかった場合は、適切なエラーコードを返す。

4GB までのファイルサイズしか扱えないファイルシステム実装部において、*off に4GB 以上の値が設定された場合、EX_OVERFLOW のエラーコードを返す。

oflags に O_APPEND が指定されている場合は、*off の値は無視され、常にファイルの末尾に追加書込みを行う。

□ FIMP_IOCTL

```
struct fimp_ioctl {
        int             r_code;         /* (in) = FIMP_IOCTL */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        fid_t           fid;            /* (in) ファイルオープン識別子 */
        int             oflags;         /* (in) オープンフラグ */
        int             dcmd;           /* (in) デバイス制御コマンド番号 */
        void *          arg;            /* (in/out) デバイス制御パラメータ */
        ER *            retval;         /* (out) 戻値へのポインタ */
};
```

fid で指定されたオープン中のファイルに対して、ファイルシステム実装部固有のデバイス制御を行う。

dcmd は、fs_ioctl()の第二引数であり、ファイルシステム実装部固有のデバイス制御コマンド番号である。
指定されたコマンド番号をサポートしていない場合は EX_NOSYS エラーを返す。

arg は、fs_ioctl()の第三引数であり、dcmd に依存した内容である。

*retval には、fs_ioctl() で返す値を格納する。

□ FIMP_FSYNC

```
struct fimp_fsync {
        int             r_code;         /* (in) = FIMP_FSYNC */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        fid_t           fid;            /* (in) ファイルオープン識別子 */
        int             type;           /* (in) TYPE_FSYNC または TYPE_FDATASYNC */
        int             oflags;         /* (in) オープンフラグ */
};
```

fid で指定されたオープン中のファイルに関して、まだデバイスに反映していないキャッシュデータをデバイスに反映する。

type が TYPE_FSYNC の場合、ファイルのデータおよびファイルの管理情報すべてをデバイスに反映し、
TYPE_FDATASYNC の場合、ファイルのデータと直接関係しない管理情報(アクセス時刻や更新時刻など)は反映しない。

ファイルシステム実装部によっては、TYPE_FDATASYNC の動作は TYPE_FSYNC の動作と同じことがある。

すでに対象のキャッシュデータがデバイスに反映されている場合は、何もせずに E_OK を返す。

□ FIMP_TRUNCATE64

```
struct fimp_truncate64 {
        int             r_code;         /* (in) = FIMP_TRUNCATE64 */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        const char *    path;           /* (in) ファイルパス名 */
        off64_t         len;            /* (in) ファイルサイズ */
};
```

path で指定されたファイルを len で指定されたサイズに切り詰める、または拡大する。
拡大した場合、拡大した領域には 0 が詰められる。

4GB までのファイルサイズしか扱えないファイルシステム実装部において、len に4GB 以上の値が設定された場合、EX_OVERFLOW のエラーコードを返す。

□ FIMP_FTRUNCATE64

```
struct fimp_ftruncate64 {
        int             r_code;         /* (in) = FIMP_FTRUNCATE64 */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        fid_t           fid;            /* (in) ファイルオープン識別子 */
        off64_t         len;            /* (in) ファイルサイズ */
};
```

fid で指定されたオープン中のファイルを len で指定された長さに切り詰める、または拡大する。
拡大した場合、拡大した領域には 0 が詰められる。

指定されたファイルは書込みオープンされていなければいけない。

4GB までのファイルサイズしか扱えないファイルシステム実装部において、len に 4GB 以上の値が設定された場合、EX_OVERFLOW エラーコードを返す。

□ FIMP_UNLINK

```
struct fimp_unlink {
        int             r_code;         /* (in) = FIMP_UNLINK */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        const char *    path;           /* (in) ファイルパス名 */
```

```
};
```

path で指定されたファイルのリンクを削除する。

リンクに対応していないファイルシステム実装部においては、ファイル自体を削除する。

□ FIMP_RENAME

```
struct fimp_rename {
        int             r_code;         /* (in) = FIMP_RENAME */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        const char *    oldpath;        /* (in) 変更前パス名 */
        const char *    newpath;        /* (in) 変更後パス名 */
};
```

oldpath で指定されたファイルのパス名を newpath で指定されたパス名に変更する。
oldpath と newpath は、同一のファイルシステム内、すなわち、同一の接続ポイント下になければいけない。

□ FIMP_CHMOD

```
struct fimp_chmod {
        int             r_code;         /* (in) = FIMP_CHMOD */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        const char *    path;           /* (in) ファイルパス名 */
        mode_t          mode;           /* (in) ファイルモード */
};
```

path で指定されたファイルのアクセス許可モードを mode で指定された値に設定する。

mode に関しては、FIMP_OPEN 項目を参照のこと。

□ FIMP_FCHMOD

```
struct fimp_fchmod {
        int             r_code;         /* (in) = FIMP_FCHMOD */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        fid_t           fid;            /* (in) ファイルオープン識別子 */
        mode_t          mode;           /* (in) ファイルモード */
};
```

fid で指定されたオープン中のファイルのアクセス許可モードを mode で指定された値に設定する。

mode に関しては、FIMP_OPEN 項目を参照のこと。

□ FIMP_MKDIR

```
struct fimp_mkdir {
        int             r_code;         /* (in) = FIMP_MKDIR */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        const char *    path;           /* (in) ディレクトリパス名 */
        mode_t          mode;           /* (in) ファイルモード */
};
```

path で指定された新しいディレクトリを生成し、mode で指定されたアクセスモードを設定する。

mode の解釈に関しては、FIMP_OPEN 項目を参照のこと。

□ FIMP_RMDIR

```
struct fimp_rmdir {
        int             r_code;         /* (in) FIMP_RMDIR */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        const char *    path;           /* (in) ディレクトリパス名 */
};
```

path で指定されたディレクトリを削除する。

□ FIMP_CHDIR

```
struct fimp_chdir {
        int             r_code;         /* (in) FIMP_CHDIR */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        const char *    path;           /* (in) ディレクトリパス名 */
};
```

path で指定されたディレクトリにカレントディレクトリを変更する。

カレントディレクトリはファイルマネージャで管理され、ファイルシステム実装部には常に絶対パスが渡されるため、ファイルシステム実装部でカレントディレクトリを管理する必要はないが、カレントディレクトリを保持することにより処理が高速化できる可能性がある。

□ FIMP_FCHDIR

```
struct fimp_chdir {
        int             r_code;         /* (in) = FIMP_CHDIR */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        fid_t           fid;            /* (in) ファイルオープン識別子 */
        char *          buf;            /* (out) ディレクトリパス名の格納領域へのポインタ */
        int             len;            /* (in) buf の領域のバイト数 */
};
```

fid で指定されたオープン中のディレクトリにカレントディレクトリを変更し、その絶対パス名を *buf に格納する。
len は buf の領域のバイト数を指定する。len が小さくて絶対パス名を格納できない場合は、エラーコード(EX_NAMETOOLONG)を返す。

*buf に格納されたカレントディレクトリの絶対パスは、ファイルマネージャ内で保持され、相対パスから絶対パスへの変換に使用される。

□ FIMP_GETDENTS

```
struct fimp_getdents {
        int             r_code;         /* (in) = FIMP_GETDENTS */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        fid_t           fid;            /* (in) ファイルオープン識別子 */
        int             oflags;         /* (in) オープンフラグ */
        struct dirent * buf;            /* (out) ディレクトリエントリ読み込み領域 */
        size_t *        len;            /* (in/out) サイズ/処理サイズへのポインタ */
        off64_t *       off;            /* (in) 処理前ディレクトリオフセットへのポインタ */
        off64_t *       retoff;         /* (out) 処理後ディレクトリオフセットへのポインタ */
};
```

fid で指定されたオープン中のディレクトリの *off で指定されたファイルオフセットから一つ以上のディレクトリエントリを *buf に読み込む。

実際に読込んだバイト数を *len に格納し、読込み後のディレクトリオフセットを *retoff に格納して戻る。
off と retoff が、同じ値のポインタであっても正しく動作しなくてはいけない。

ディレクトリ末尾に達している場合は、*len に 0 を格納し、E_OK を返す。

□ FIMP_FSTATVFS

```
struct fimp_fstatvfs {
        int             r_code;         /* (in) = FIMP_FSTATVFS */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        fid_t           fid;            /* (in) ファイルオープン識別子 */
        struct statvfs * buf;           /* (out) ファイルシステム統計情報取得領域へのポインタ */
};
```

fid で指定されたオープン中のファイルが含まれるファイルシステムに関する統計情報(statvs 構造体)を *buf に格納する。

□ FIMP_STATVFS

```
struct fimp_statvfs {
        int             r_code;         /* (in) = FIMP_STATVFS */
```

```
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        const char *    path;           /* (in) ファイルパス名 */
        struct statvfs * buf;           /* (out) ファイルシステム統計情報取得領域へのポインタ */
};
```

path で指定されたファイルが含まれるファイルシステムに関する統計情報(statvfs 構造体)を *buf に格納する。

□ FIMP_SYNC

```
struct fimp_sync {
        int             r_code;         /* (in) = FIMP_SYNC */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
};
```

ファイルシステム実装部が対応している接続済みのすべてのファイルシステムに関して、まだデバイスに反映されていないすべてのキャッシュデータをデバイスに反映する。

すでにすべてのキャッシュデータがデバイスに反映されている場合は、何もせずに E_OK を返す。

□ FIMP_UTIMES_US

```
struct fimp_utimes_us {
        int             r_code;         /* (in) = FIMP_UTIMES_US */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        const char *    path;           /* (in) ファイルパス名 */
        SYSTIM_U *      times_u;        /* (in) times_u[0] アクセス時刻、times_u[1] 更新時刻 */
};
```

path で指定されたファイルのアクセスおよび更新時間を *times_u で指定されたアクセス時刻、更新時刻に変更する。

□ FIMP_FCNTL64

```
struct fimp_fcntl64 {
        int             r_code;         /* (in) = FIMP_FCNTL64 */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        fid_t           fid;            /* (in) ファイルオープン識別子 */
        int *           oflags;         /* (in) オープンフラグへのポインタ */
        int             fcmd;           /* (in) ファイル制御コマンド番号 */
        off64_t *       off;            /* (in/out) 現在のファイルオフセットへのポインタ */
        void *          arg;            /* (in/out) ファイル制御パラメータ */
        ER *            retval;         /* (out) 戻値へのポインタ */
};
```

fid で指定されたオープン中のファイルに対して、ファイルシステム実装部固有のファイル制御を行う。

oflags は、fs_open()で指定されたオープンフラグへのポインタであり、変更することができる。

fcmd は、fs_fcntl()の第二引数であり、ファイルシステム実装部固有のファイル制御コマンド番号である。
指定されたコマンド番号をサポートしていない場合は EX_NOSYS エラーを返す。

arg は、fs_fcntl()の第三引数であり、cmd に依存した内容である。

*retval には、fs_fcntl() で返すエラーコード値を格納する。

□ FIMP_FSTAT64_US

```
struct fimp_stat64_us {
        int             r_code;         /* (in) = FIMP_STAT64_US */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        fid_t           fid;            /* (in) ファイルオープン識別子 */
        struct stat64_us * buf;         /* (out) ファイル情報取得領域へのポインタ */
};
```

fid で指定されたオープン中のファイルに関する情報(stat64_us 構造体) を *buf に格納する。

□ FIMP_STAT64_US

```
struct fimp_stat64_us {
        int             r_code;         /* (in) = FIMP_STAT64_US */
        coninf_t *      coninf;         /* (in/out) 接続情報へのポインタ */
        const char *    path;           /* (in) ファイルパス名 */
        struct stat64_us * buf;         /* (out) ファイル情報取得領域へのポインタ */
};
```

path で指定されたファイルに関する情報(stat64_us 構造体)を *buf に格納する。

第5章　ネットワーク通信機能

5.1 概要

ネットワーク通信機能は、TCP/IP、UDP/IPを含む通信のソケットインタフェースを提供する機能である。API 名称のプレフィクスとして ″so_″ (socket) を持つ。また、ネットワーク通信機能を提供する実体をネットワーク通信マネージャと呼ぶ。

ネットワーク通信機能は、POSIX 仕様におけるネットワーキングサービスに近いAPIを持つ。POSIX仕様との大きな違いは、APIコールの戻値が負の場合はT-Kernel仕様のエラーコードになっている点である。

タイムアウト時間を必要とするAPIでは、POSIX 仕様の時間データ形式の他に T-Kernel 2.0 のタイムアウト時間形式(TMO, TMO_U)を使える API を追加している。

5.2 用語

5.2.1 ソケット

ソケットは、本章で説明するネットワーク通信機能が使用する端点である。ソケットは、後述のソケットタイプを指定して生成され、特定のプロトコルに関連付けられる。

5.2.2 ソケットディスクリプタ

ソケットを識別するための0以上の整数であり、ソケットを生成時に新規に割り当てられる。ソケットに対する操作を行うときにソケットディスクリプタを使用する。

5.2.3 ストリーム

ストリームは、ファイルの読み書きとネットワーク通信で使用される抽象化であり、順序性のある連続したアクセスが可能である。

5.2.4 データグラム

コネクションレス型サービスにおいて、1つの端点から別の端点へ転送されるデータの単位である。

5.2.5 ソケットタイプ

ソケットタイプはソケットの特性を表す。ソケットはソケットタイプを指定して生成され、そのソケットタイプは通信セマンティクスを定義し、ソケットタイプに従い適切な通信プロトコルが選択される。

T2EX では、SOCK_STREAM、 SOCK_DGRAM、SOCK_RAW の 3 種類のソケットタイプが利用できる。

SOCK_STREAM
　　接続された2つのソケット間で、信頼性と順序性がある全二重接続のストリームを提供する。
　　SOCK_STREAM 型ソケットは通信相手と接続してからデータを送受信する。SOCK_STREAM 型ソケットには以下の特徴がある。
　　- SOCK_STREAM 型ソケットはレコード境界を保持しない。SOCK_STREAM 型ソケットにデータを送信した場合、そのデータよりも小さな、または、大きなサイズを指定して入力APIコールを用いても、欠落することなくデータを受信できる。
　　- SOCK_STREAM 型ソケットはデータをバッファリングする。
　　そのため、出力APIコールが正常終了しても送信先に配送されたことを保証しない。一定時間以内にデータを正常に送信できなかった場合は接続が切断されたとみなし、それ以降のそのソケットに対する操作は失敗する。

SOCK_DGRAM
　　信頼性のないコネクションレスのデータ配送を提供する。出力操作時に(マルチキャストまたはブロードキャストを含めて)アドレスを指定してデータグラムを送信する。また、複数の送信元からのデータグラムも受信できる。各データグラムの送信元アドレスは、データグラム受信時に知ることができる。
　　あるいは、アプリケーションは通信相手アドレスを事前に指定することもできる。この場合、通信相手アドレスを指定せずに出力APIコールを呼び出すと、事前に指定した通信相手アドレスにデータグラムが送信される。事前に指定する代わりに通信相手を指定して入力APIコールを呼び出すと、指定した通信相手からのデータグラムのみを受信する。
　　SOCK_DGRAM 型ソケットの場合、データグラムを一度の出力操作で送信しなければならない。
　　同様に、データグラムを一度の入力操作で受信しなければならない。
　　また、SOCK_DGRAM 型ソケットには以下の特徴がある。
　　- データグラムの最大サイズはプロトコルに依存する。
　　- 出力されるデータグラムをシステム内で一時的に保存する。
　　そのため、出力APIコールが正常終了してもデータグラムの送信成功を保証しない。

SOCK_RAW
　　パケットヘッダを含むデータグラムの送受信を提供する。
　　- アドレスファミリがAF_INETの場合、アプリケーションはIPヘッダを含むパケットを送受信する。た

だし、so_setsockopt() で IP_HDRINCL に 0 を設定している場合、送信パケットにはIPヘッダが自動的に付与される。このとき、ソケットに設定された宛先アドレスとプロトコル番号がIPヘッダに設定される。IP_HDRINCL に 0 以外の値を設定している場合、送信パケットにIPヘッダが自動的に付与されることはなくアプリケーションがIPヘッダを設定しなければならない。
- アドレスファミリがAF_ROUTEの場合、ルーティングテーブルを操作する機能を提供する。
- それ以外のアドレスファミリはSOCK_RAWをサポートしていない。

5.2.6 アドレスファミリ

アドレスファミリは、特定のネットワーク環境内で機能するプロトコルを実装するために必要な基本サービスを提供する。このサービスには、パケットの断片化・再構築、ルーティング、アドレッシング、基本的なデータ転送が含まれる。アドレスファミリは、通常、複数のプロトコルから構成され、1つのソケットタイプに対して 1つのプロトコルが存在する。各プロトコルはソケットタイプに応じた機能を提供する。

アドレスファミリはすべてのソケットタイプをサポートするとは限らない。また、1 つのアドレスファミリの中に、同じソケットタイプをサポートするプロトコルが複数存在してもよい。

5.2.7 ネットワークアドレス

ネットワークアドレスとは、ネットワーク上の端点を指定する識別子である。ただし、IPのアドレッシングの場合は、サブネットを指定する識別子となる。ホスト自身やホスト内の特定の端点などがネットワークアドレスを持つ。

5.2.8 ソケットアドレス

ソケットに関連付けられるアドレスであり、アドレスファミリ識別子とそのアドレスファミリ固有のアドレッシング情報を持つ。また、このアドレスは、ホストのネットワークアドレス、特定の端点のアドレスなどの情報を持つこともある。

5.2.9 アドレッシング

ソケットアドレスの形式はアドレスファミリにより定義される。ネットワーク通信機能のAPI の中でネットワークアドレスは、 sockaddr 構造体と呼ばれる汎用的なデータ構造を用いて表現される。しかし、各アドレスファミリを記述するためには、それぞれに固有のデータ構造が必要である。そのため、通常、アドレスファミリ固有のフィールドをもつデータ構造を定義する(例: IPv4の場合は sockaddr_in 構造体(第8章 netinet/in.h参照)。

5.2.10 プロトコル

プロトコルは、情報を交換するための意味的、統語的な規約である。プロトコルを通信プロトコル、ネットワークプロトコルと呼ぶこともあるが、本書では曖昧さが生じない限りプロトコルとのみ記述する。

プロトコルは 1 つのソケットタイプをサポートする。so_socket() に引数として渡すアドレスファミリ、ソケットタイプ、プロトコル番号を指定することで、プロトコルを選択する。
プロトコルはそのソケットタイプに応じた機能を提供する。また、基本的な機能に加えて、標準的ではない機能または拡張機能を提供することもある。

5.2.11 ルーティング

ソケットはパケットのルーティング機能を提供する。この機能はパケット送信時に適切なネットワークインタフェースを選択し、この時に利用されるルーティング情報データベースを管理する。

5.2.12 メッセージ

本章ではメッセージをソケット間で通信される情報を指す用語として使用する。

5.2.13 ネットワークインタフェース

システム内の各ネットワークインタフェースは、メッセージが送受信される経路に対応する。通常、ネットワークインタフェースはハードウェアに関連付けられている。しかし、ループバックインタフェースなど一部のインタフェースは、ハードウェアに関連付けられていない。

5.2.14 ソケットI/Oモード

ソケットの I/O モードは、O_NONBLOCK状態フラグにより記述される。ソケット生成時はこのフラグを設定しない。so_fcntl() で F_SETFL コマンドを用いることにより、このフラグを設定・解除できる。

基本的にネットワーク通信機能のAPIコールを呼び出したタスクは処理が完了するまで待つ。O_NONBLOCK 状態フラグをソケットディスクリプタに設定すると、タスクは待たずに直ちにAPIコールから戻る。この状態をノンブロッキングモードと呼ぶ。

O_NONBLOCK を設定した場合の動作

so_bind() は、ソケットアドレスを直ちに割当てることができない場合でも待ちに入らず、

ソケットアドレス割当ての初期化を行った後にエラー EX_INPROGRESS を返す。
この場合、ソケットアドレスの割当て処理は開始しているが、完了していない。

so_connect() は、接続を直ちに確立できない場合でも待ちに入らず、接続を初期化し後に
エラー EX_INPROGRESS を返す。この場合、接続処理は開始しているが、完了していない。

データ転送操作 (so_read(), so_write(), so_send(), so_recv()) は、可能な量だけ転送して、
待ちに入らずに正常終了する。または、データ転送を直ちに開始できなかったことを示す
エラー E_AGAIN を返す。

5.2.15 ソケットの所有者

T2EX のソケットには所有者の概念はない。

5.2.16 ソケットキューの制限

ソケットの送受信キューのバッファサイズはソケット生成時に設定される。ソケット生成時におけるデフォルトのバッファサイズはプロトコルに依存する。so_setsockopt() の SO_SNDBUF、SO_RCVBUF を用いてこれらのサイズを変更できる。

各プロトコルのデフォルトの送受信キューのバッファサイズは、システム構成情報として定義される。

- SOCK_STREAM型ソケット(AF_INET): TCP/IP
  - SoTcpTxBufSz: 送信バッファサイズ
  - SoTcpRxBufSz: 受信バッファサイズ
- SOCK_DGRAM型ソケット(AF_INET): UDP/IP
  - SoUdpTxBufSz: 送信バッファサイズ
  - SoUdpRxBufSz: 受信バッファサイズ
- SOCK_RAW型ソケット(AF_INET)
  - SoRawIPTxBufSz: 送信バッファサイズ
  - SoRawIPRxBufSz: 受信バッファサイズ
- SOCK_RAW型ソケット(AF_ROUTE)
  - SoRawTxBufSz: 送信バッファサイズ
  - SoRawRxBufSz: 受信バッファサイズ

5.2.17 保留中のエラー

エラーは非同期的に発生する可能性があり、プロトコルで発生したエラーがソケットに通知される。アプリケーションに通知すべきエラーは、ソケットに保持されて一時的に保留される。エラー発生後のソケットに対して so_send(), so_recv(), so_getsockopt() を呼び出すと、これらのAPIコールは保留されていたエラーを返し、そのエラーはソケットから削除される。

5.2.18 ソケット受信キュー

ソケットは、システムがデータを受信した時にデータを保持する受信キューを持つ。保持されたデータは受信APIコールの呼び出しにより削除される。ソケットタイプに応じて、通常データに関連するアドレス情報や他のプロトコルデータなどの補助データが受信キューに保持される。また、帯域外データまたは優先データも受信キューに保持される。

5.2.19 帯域外データ

帯域外データは通常データとは論理的に独立した通信路で配送されるデータであり、通常データとは独立してユーザーに配送される。
帯域外データ受信時の扱い方は、通常データの受信キューに保持する方法と別のキューに保持する2通りの方法がある。

通常データの受信キューに保持する場合
帯域外データは、キューの最後またはすべての通常データの前に配置される。
この場合、アプリケーションは通常の受信呼び出しを用いて帯域外データを取得する。

通常データとは別のキューに保持する場合
アプリケーションは、通常の受信呼び出しではなく帯域外データを要求する受信呼び出しを用いて
帯域外データを取得する。
帯域外データに先行する通常データとそれに続く通常データの境界を示す帯域外データマークは、
帯域外データ受信時に受信キューに保持される。帯域外データマークは論理的に空のデータセグメント
であり、キュー中の他のデータと統合されることはない。帯域外データマークは入力操作では取得できず、
so_sockatmark() を用いて帯域外データマークが受信キューの先頭に位置するか調べる必要がある。
帯域外データマークが受信キューの先頭に位置して、かつ MSG_PEEK フラグを指定せずに
入力APIコールを呼び出した場合、帯域外データマークは取り除かれて、これが存在しなかったように
(もしあれば)続くデータが処理される。

5.2.20 接続待ちキュー

接続要求を受け付けるソケットは、未処理の接続要求を保持するキューを持つ。このキューは、アプリケーションが受け付けるまで待機している接続要求のリストである。

5.2.21 非同期エラー

ソケットに対して次の条件が非同期的に生じると、対応する以下のエラーコードがソケットに保持される。

EX_CONNABORTED	ローカルで接続が中断された
EX_CONNREFUSED	コネクション型ソケットの場合、ノンブロッキングモードの接続処理中に強制的に接続が拒否された コネクションレス型ソケットの場合、データグラムの配送が強制的に拒否された
EX_CONNRESET	通信相手が接続を中断した
EX_HOSTDOWN	宛先ホストがダウンしている、あるいは宛先ホストが接続を切断した
EX_HOSTUNREACH	宛先ホストに到達できない
EX_MSGSIZE	コネクションレス型ソケットに対して、前に送信したデータグラムのサイズが不正のために配送できない。
EX_NETDOWN	ローカルネットワークが稼働していない
EX_NETRESET	リセットによりネットワーク接続が中断された
EX_NETUNREACH	宛先ネットワークに到達できない

5.2.22 ソケットオプション

ソケットオプションを用いると、ソケットの振舞いを制御でき、また関連する情報を取得できる。このオプションはプロトコルレベル毎に提供されていて、一番上位のソケットレベルはどのソケットに対しても常に存在する。ソケットオプションを操作する場合は so_getsockopt(), so_setsockopt() の 2 つのAPIコールを用いる。
プロトコルレベルには、以下を指定できる。

IPPROTO_IP	IPレベル
IPPROTO_TCP	TCPレベル
SOL_SOCKET	ソケットレベル

以下のオプションの説明中の R-、RW は、それぞれ取得(so_getsockopt())のみできることと取得・設定(so_getsockopt(), so_setsockopt())の両方ができることを意味する。

IPレベルの次のオプションには、以下を指定できる。

IP_OPTIONS	RW	送信される各パケットのIPヘッダに埋め込まれるIPオプション 以前に指定されたオプションを無効にするには、長さ 0 のバッファを指定する。 IPオプションの詳細については、RFC-791 を参照。 デフォルト時、IPヘッダに埋め込むIPオプションは設定されていない。
IP_HDRINCL	RW	アプリケーションによる送信データへIPヘッダ付与の有効・無効 オプションの型: int オプションの値: - 0 以外の場合、アプリケーションは送信データにIPヘッダを設定して送信する。 - 0 の場合、アプリケーションは送信データにIPヘッダを設定せず、システムがIPヘッダを付与する。 このオプションはSOCK_RAW型ソケットに対してのみ有効である。 デフォルト時、システムがIPヘッダを付与する。

TCPレベルの次のオプションには、以下を指定できる。

TCP_NODELAY	RW	データを直ちに送信することの許可・禁止 オプションの型: int オプションの値: - 0 以外の場合、Nagleのアルゴリズムが無効である。 - 0 の場合、Nagleのアルゴリズムが有効である。 TCPは、ネットワークを有効に利用するために、Nagleのアルゴリズムを用いて、 小さなパケットを頻繁に送らずに送信データを溜めてから送信する。Nagleのアルゴリズムが無効の場合、小さなパケットも直ちに送信する。 デフォルト時、Nagleのアルゴリズムは有効である。
TCP_MAXSEG	RW	最大セグメント長 オプションの型: int デフォルト時、536バイトである。

ソケットレベルの次のオプションには、以下を指定できる。

SO_DEBUG RW 下位層プロトコルのデバッグ情報の有効・無効
オプションの型: int
オプションの値:
- 0 以外の場合、デバッグ情報が有効である。
- 0 の場合、デバッグ情報が無効である。
デフォルト時、デバッグ情報は無効である。

SO_ACCEPTCONN R- so_listen() による接続待ち状態の確認
オプションの型: int
オプションの値:
- 0 以外の場合、接続待ち状態である
- 0 の場合、接続待ち状態ではない

SO_REUSEADDR RW ローカルアドレスの再利用の有効・無効
オプションの型: int
オプションの値:
- 0 以外の場合、ローカルアドレスを再利用する。
- 0 の場合、ローカルアドレスを再利用しない。
デフォルト時、ローカルアドレスの再利用は無効である。

SO_KEEPALIVE RW 定期的なキープアライブメッセージ送信の有効・無効
オプションの型: int
オプションの値:
- 0 以外の場合、定期的なキープアライブメッセージを送信する。
- 0 の場合、定期的なキープアライブメッセージを送信しない。
デフォルト時、キープアライブメッセージを送信しない。

SO_DONTROUTE RW 標準のルーティング機能をバイパスしたメッセージ送信の有効・無効
オプションの型: int
オプションの値:
- 0 以外の場合、このオプションは有効である。
- 0 の場合、このオプションは無効である。
このオプションが有効な場合、ネットワークに直接接続された宛先へ送信しなければならず、宛先アドレスに従い適切なインタフェースでメッセージが送信される。無効な場合、標準のルーティング機能を使用してメッセージが送信される。
デフォルト時、標準のルーティング機能を使用してメッセージが送信される。

SO_BROADCAST RW ブロードキャストメッセージ送信の有効・無効
オプションの型: int
オプションの値:
- 0 以外の場合、ブロードキャストメッセージを送信できる。
- 0 の場合、ブロードキャストメッセージを送信できない。
デフォルト時、ブロードキャストメッセージ送信は無効である。

SO_USELOOPBACK RW ハードウェアをバイパスして通信する機能の有効・無効
オプションの型: int
オプションの値:
- 0 以外の時、可能であればハードウェアをバイパスして通信する。
- 0 の時、ハードウェアをバイパスした通信はしない。
デフォルト時、ハードウェアをバイパスした通信はしない。

SO_LINGER RW 未送信のデータがある場合にソケットを閉じた時の動作
オプションの型: struct linger
オプションの値:
- l_onoffの値が 0 以外、かつ l_linger の値が正の場合、未送信のデータが送信完了するか、または l_linger 秒経過してタイムアウトするまで待つ。
- l_onoffの値が 0 、または l_lingerの値が 0 の場合、未送信データの有無に関わらず可能な限り速やかに so_close() の処理を完了させる。この場合、待ちに入らない。
デフォルト時、l_onoff と l_linger の双方に 0 が設定されている。

SO_OOBINLINE RW 帯域外データを通常の受信キューへ挿入する機能の有効・無効
オプションの型: int
オプションの値:
- 0 以外の場合、帯域外データを通常の受信キューへ挿入して、MSG_OOB フラグを設定せずに so_read(), so_recv() などから取得できる。
- 0 の場合、帯域外データを通常の受信キューへ挿入しない。
デフォルト時、帯域外データを通常の受信キューへ挿入しない。

SO_REUSEPORT RW ローカルアドレスとポートの再利用の有効・無効
オプションの型: int

オプションの値:
 - 0 以外の場合、ローカルアドレスとポートを再利用する。
 - 0 の場合、ローカルアドレスとポートを再利用しない。
デフォルト時、ローカルアドレスとポートを再利用しない。

SO_TIMESTAMP RW 受信したデータグラムへのタイムスタンプ追加の有効・無効
オプションの型: int
オプションの値:
 - 0 以外の場合、受信したデータグラムにタイムスタンプを追加する。
 - 0 の場合、受信したデータグラムにタイムスタンプを追加しない。
このオプションが有効な場合、タイムスタンプは補助データに格納され、cmsg_len, cmsg_level, cmsg_type はそれぞれ timeval 構造体のサイズ(バイト数)、SOL_SOCKET、SCM_TIMESTAMP となる。
デフォルト時、受信したデータグラムにタイムスタンプを追加しない。

SO_SNDBUF RW 送信バッファのサイズ（バイト数）
オプションの型: int
デフォルトの送信バッファサイズはプロトコルに依存する。
 - TCP/IP: 32768バイト
 - UDP/IP: 9216バイト
 - SOCK_RAW型ソケット(AF_INET): 8192バイト
 - SOCK_RAW型ソケット(AF_ROUTE): 8192バイト

SO_RCVBUF RW 受信バッファのサイズ（バイト数）
オプションの型: int
デフォルトの受信バッファサイズはプロトコルに依存する。
 - TCP/IP: 32768バイト
 - UDP/IP: 41600バイト
 - SOCK_RAW型ソケット(AF_INET): 8192バイト
 - SOCK_RAW型ソケット(AF_ROUTE): 8192バイト

SO_SNDLOWAT RW ブロックせずに書込み可能な送信バッファの最小の空きサイズ（バイト数）
オプションの型: int
デフォルト時、2048バイトである。

SO_RCVLOWAT RW ブロックせずに読み取り可能な受信バッファの最小の受信データサイズ（バイト数）
オプションの型: int
デフォルト時、1バイトである。

SO_SNDTIMEO RW 送信タイムアウト
オプションの型: struct timeval
オプションの値:
 - optvalの値が 0 秒以外の値を示す場合、送信APIコールは指定された時間が経過するまで待つ。タイムアウトした場合、何もデータを書き込んでいない場合はエラー EX_AGAIN を返し、1 バイト以上のデータを書き込んだ場合はそのデータサイズ(バイト数)を返す。
 - 0 秒の値を示す場合、送信APIコールに設定されていたタイムアウト時間を解除する。
デフォルト時、タイムアウト時間は設定されていない。

SO_RCVTIMEO RW 受信タイムアウト
オプションの型: struct timeval
オプションの値:
 - optvalの値が 0 秒以外の値を示す場合、受信APIコールは指定された時間が経過するまで待つ。タイムアウトした場合、何もデータを読込んでいない場合はエラー EX_AGAIN を返し、1 バイト以上のデータを読み込んだ場合はそのデータサイズ(バイト数)を返す。
 - optvalの値が 0 秒の値を示す場合、受信APIコールに設定されていたタイムアウト時間を解除する。
デフォルト時、タイムアウト時間は設定されていない。

SO_ERROR R- ソケットに対する保留中のエラー
オプションの型: int
オプションの値:
 - 0 以外の場合、非同期エラーを示す。
 - 0 の場合、保留中のエラーはない。

SO_TYPE R- ソケットタイプ
オプションの型: int

5.2.23 IPを使用するソケットの利用

so_socket() は、プロトコルに 0 の値を指定してソケットを生成した場合、指定したソケットタイプに従いデフォルトのプロトコルをソケットに設定する。各ソケットタイプに対して、次のプロトコルがデフォルトプロトコルとして設定される。

SOCK_STREAM	IPPROTO_TCP
SOCK_DGRAM	IPPROTO_UDP
SOCK_RAW	IPPROTO_RAW

SOCK_STREAM 型ソケットと SOCK_DGRAM 型ソケットのデフォルトのプロトコルは、それぞれ TCP と UDP である。

SOCK_RAW 型ソケットは、IPを直接操作できるインタフェースを提供する。SOCK_RAW 型ソケットのデフォルトプロトコルは IPPROTO_RAW の値を含む IP ヘッダで識別される。アプリケーションは SOCK_RAW 型ソケット生成時にデフォルトプロトコルではなく特定のプロトコルを指定すべきである。例えば、ICMP 制御プロトコルにアクセスするソケットを生成する場合は、IPPROTO_ICMP をプロトコルに指定する。

5.2.24 ホスト名テーブル

アドレス、ホスト名、ホストの別名を定義しているテーブルである。PCなどに実装されているTCP/IPプロトコルスタックにおける設定ファイル /etc/hosts に相当する。
ホスト名テーブルに対する設定はシステム共通である。

ホスト名テーブルの各エントリは、少なくともアドレスとホスト名から構成される。また、ホスト名に加えて複数の別名も定義できる。

5.2.25 名前解決

ドメイン名からネットワークアドレスへの変換すること。

T2EXでは、ホスト名テーブルと名前解決サーバによる名前解決機能を提供する。

5.2.26 ルーティングテーブル

ルーティングテーブルは、宛先への経路を探索するときに利用されるルーティング情報データベースである。このテーブルは経路情報から構成されており、経路情報は宛先毎に使用すべきネットワークインタフェースと、宛先へ到達するためにホストが最初に送信する相手であるゲートウェイなどの情報を保持している。

宛先は特定のホストまたはネットワークに属する全てのホストである。ネットワークに属する全てのホストは、ネットワークのアドレスとネットマスクを用いて表現される。また、すべてがゼロのネットマスクをもつ特殊なアドレスは、宛先が特定のホストまたはネットワークのアドレスに当てはまらない場合に使用される。このときに使用されるゲートウェイをデフォルトゲートウェイと呼ぶ。

5.2.27 ルーティングソケット

ルーティングソケットはルーティングテーブルの操作を行うための特殊なソケットである。

ルーティングソケットに対する操作は 5.6 に記述してある。

5.3 サポートしない機能

T2EX のネットワーク通信機能では、POSIX仕様におけるネットワーキングサービスのサブセットを提供しているが、以下の点については機能の差異がある。

(1) T2EXではプロセスが無く、ユーザという概念もないため、ソケットの所有者やアクセスできるユーザのグループは意味がなく、これに関連した機能も提供しない。

(2) T2EXのネットワーク通信機能は、ファイル管理機能とは独立している。そのため、POSIX 仕様とは異なり、同一のT2EXのAPIコールでネットワーク通信とファイル管理の両方を処理することはない。ネットワーク通信機能のAPIコールは so_XXXX()、ファイル管理機能のAPIコールはfs_XXXX() の名称となっており、両者はすべて別のAPIコールになっている。

(3) 以下の機能は提供しない。ただし、将来的に T2EX の機能としてサポートする可能性はある。

- IPv6
- IPsec
- マルチキャスト
- UNIXドメインソケット
- DHCP と DNS を除く、トランスポート層より上位のプロトコル

5.4 データ型の定義

5.4.1 アドレスファミリ

```
#define AF_UNSPEC       0               /* 未指定 */
```

```
#define AF_INET         2               /* IPv4プロトコル */
#define AF_ROUTE        17              /* 内部ルーティングプロトコル */
#define AF_LINK         18              /* データリンク層プロトコル */
```

5.4.2 プロトコルファミリ

```
#define PF_UNSPEC       0               /* 未指定 */
#define PF_INET         2               /* IPv4プロトコル */
#define PF_ROUTE        17              /* 内部ルーティングプロトコル */
#define PF_LINK         18              /* データリンク層プロトコル */
```

5.4.3 ソケットタイプ

```
#define SOCK_STREAM     1               /* ストリームソケット */
#define SOCK_DGRAM      2               /* データグラムソケット */
#define SOCK_RAW        3               /* 生ソケット */
```

5.4.4 プロトコル

```
#define IPPROTO_IP      0               /* IP */
#define IPPROTO_ICMP    1               /* ICMP */
#define IPPROTO_TCP     6               /* TCP */
#define IPPROTO_UDP     17              /* UDP */
#define IPPROTO_RAW     255             /* SOCK_RAW型ソケットのデフォルトプロトコル */
```

5.4.5 アドレスを表現する汎用的な構造体

struct sockaddr はソケットアドレスを表す汎用的な構造体である。ソケットアドレスを扱うAPIで使用される。

```
struct sockaddr {
        unsigned char   sa_len;         /* アドレスのサイズ(バイト数) */
        unsigned char   sa_family;      /* アドレスファミリ識別子 */
        char            sa_data[14];    /* アドレッシング情報 */
};
```

アドレスファミリを示す sa_family フィールドは sa_dataフィールドの形式を規定し、sa_dataフィールドはアドレスファミリ固有の領域である。

アプリケーションが特定のアドレスファミリに属するアドレスを使用する場合は、そのアドレスを表す構造体(例:IPv4の場合は sockaddr_in 構造体(8.14 netinet/in.h参照))から sockaddr 構造体へのキャスト、または逆向きのキャストを行う。

また、アドレスを sockaddr 構造体に格納する場合、sa_data 領域のサイズは 14 バイトであることに注意する必要がある。T2EXが提供するアドレスファミリのアドレスはこのサイズを超えることはないが、独自に機能拡張したアドレスを使用する場合はこのサイズを超える可能性がある。その場合、sockaddr_storage 構造体を使用すると任意のサイズのアドレスを格納できる。

5.4.6 任意のアドレスを格納可能な構造体

struct sockaddr_storage は任意のアドレスファミリのアドレスを格納できる十分なサイズをもつ構造体である。sockaddr構造体で格納できないサイズのアドレスを扱う場合に使用する。

```
#define _SS_MAXSIZE     128
#define _SS_ALIGNSIZE   (sizeof(int64_t))
#define _SS_PAD1SIZE    (_SS_ALIGNSIZE - 2)
#define _SS_PAD2SIZE    (_SS_MAXSIZE - 2 - _SS_PAD1SIZE - _SS_ALIGNSIZE)

struct sockaddr_storage {
        uint8_t         ss_len;                         /* アドレスのサイズ(バイト数) */
        sa_family_t     ss_family;                      /* アドレスファミリ識別子 */
        char            __ss_pad1[_SS_PAD1SIZE];        /* 6バイトのパディング
                                                           (__ss_alignを8バイト境界に配置するため) */
        int64_t         __ss_align;                     /* 構造体を8バイト境界に配置するための64ビット
整数 */
        char            __ss_pad2[_SS_PAD2SIZE];        /* 112バイトのパディング
                                                           (構造体全体で128バイトのサイズにするため) */
};
```

5.4.7 データリンク層アドレス構造体

```
struct sockaddr_dl {
        uint8_t         sdl_len;                /* アドレスのサイズ(バイト長) */
        sa_family_t     sdl_family;             /* アドレスファミリ(常にAF_LINK) */
```

```
        uint16_t        sdl_index;              /* インタフェースのインデックス */
        uint8_t         sdl_type;               /* インタフェースのタイプ */
        uint8_t         sdl_nlen;               /* インタフェース名のサイズ(バイト長) */
        uint8_t         sdl_alen;               /* データリンク層アドレスのサイズ(バイト長) */
        uint8_t         sdl_slen;               /* データリンク層セレクタのサイズ(バイト長) */
        char            sdl_data[12];           /* インタフェース名とデータリンク層アドレスを格納する
バッファ */
};
```

5.4.8 統計情報とネットワークインタフェース情報を格納する構造体

```
struct if_data {
        /* インタフェース情報 */
        unsigned char   ifi_type;               /* インタフェースのタイプ */
        unsigned char   ifi_addrlen;            /* メディアのアドレスサイズ(バイト数) */
        unsigned char   ifi_hdrlen;             /* メディアのヘッダサイズ(バイト数) */
        int             ifi_link_state;         /* リンク状態 */
        uint64_t        ifi_mtu;                /* MTU */
        uint64_t        ifi_metric;             /* ルーティングメトリック */
        uint64_t        ifi_baudrate;           /* ボーレート */
        /* 統計情報 */
        uint64_t        ifi_ipackets;           /* 受信パケット数 */
        uint64_t        ifi_ierrors;            /* 受信エラー数 */
        uint64_t        ifi_opackets;           /* 送信パケット数 */
        uint64_t        ifi_oerrors;            /* 送信エラー数 */
        uint64_t        ifi_collisions;         /* CSMAインタフェースのコリジョン発生回数 */
        uint64_t        ifi_ibytes;             /* 全受信サイズ(オクテット数) */
        uint64_t        ifi_obytes;             /* 全送信サイズ(オクテット数) */
        uint64_t        ifi_imcasts;            /* マルチキャストの受信パケット数 */
        uint64_t        ifi_omcasts;            /* マルチキャストの送信パケット数 */
        uint64_t        ifi_iqdrops;            /* 破棄された受信パケット数 */
        uint64_t        ifi_noproto;            /* サポートされていないプロトコルを使用した回数 */
        struct timeval  ifi_lastchange;         /* 最後に動作状態が変更された時刻 */
};
```

5.4.9 Scatter/Gather構造体

```
struct iovec {
        void*           iov_base;       /* ベースアドレス */
        size_t          iov_len;        /* サイズ(バイト数) */
};
```

5.4.10 so_recvmsg(), so_sendmsg() 用メッセージヘッダ

```
struct msghdr {
        void*           msg_name;               /* 送信元アドレス */
        socklen_t       msg_namelen;            /* 送信元アドレスのサイズ(バイト数) */
        struct iovec*   msg_iov;                /* Scatter/Gather構造体の配列 */
        int             msg_iovlen;             /* msg_iov の配列の要素数 */
        void*           msg_control;            /* 補助データ */
        socklen_t       msg_controllen;         /* 補助データのサイズ(バイト数) */
        int             msg_flags;              /* 受信メッセージのフラグ */
};
```

5.4.11 so_recvmsg(), so_sendmsg() の補助データ用構造体

```
struct cmsghdr {
        socklen_t       cmsg_len;               /* ヘッダ(cmsghdr)を含むデータのサイズ(バイト数) */
        int             cmsg_level;             /* プロトコルレベル */
        int             cmsg_type;              /* プロトコル固有のタイプ */
};
```

補助データは、cmsghdr 構造体に続いた領域に格納される。

5.4.12 SO_LINGER オプション用構造体

```
struct linger {
        int     l_onoff;        /* オプションのオン/オフ */
        int     l_linger;       /* リンガ間隔(秒) */
};
```

5.4.13 ソケットディスクリプタ集合

```
#define FD_SETSIZE      256

typedef struct fd_set {
        int     fds_bits[((FD_SETSIZE + (sizeof(int)*8-1)) / sizeof(int)*8)];
} fd_set;
```

5.4.14 so_getaddrinfo() のアドレス情報構造体

```
struct addrinfo {
        int                     ai_flags;               /* 入力フラグ */
        int                     ai_family;              /* アドレスファミリ */
        int                     ai_socktype;            /* ソケットタイプ */
        int                     ai_protocol;            /* プロトコル */
        socklen_t               ai_addrlen;             /* ソケットアドレスのサイズ(バイト数) */
        struct sockaddr*        ai_addr;                /* ソケットアドレス */
        char*                   ai_canonname;           /* サービスロケーションの標準的な名称 */
        struct addrinfo*        ai_next;                /* 次のリストへのポインタ */
};
```

5.4.15 ネットワークインタフェース操作構造体

```
struct ifreq {
#define IFNAMSIZ        16
        char    ifr_name[IFNAMSIZ];     /* デバイス名 */
        union {
                struct sockaddr         ifru_addr;              /* ホストアドレス */
                struct sockaddr         ifru_dstaddr;           /* 宛先アドレス */
                struct sockaddr         ifru_broadaddr;         /* ブロードキャストアドレス */
                short                   ifru_flags;             /* フラグ */
                short                   ifru_metric;            /* メトリック */
                void*                   ifru_data;              /* インタフェースが使用する領域 */
        } ifr_ifru;
#define ifr_addr                ifr_ifru.ifru_addr
#define ifr_dstaddr             ifr_ifru.ifru_dstaddr
#define ifr_broadaddr           ifr_ifru.ifru_broadaddr
#define ifr_flags               ifr_ifru.ifru_flags
#define ifr_metric              ifr_ifru.ifru_metric
#define ifr_data                ifr_ifru.ifru_data
};
```

5.4.16 so_ioctl() の SIOCAIFADDR、SIOCDIFADDR 用構造体

```
struct ifaliasreq {
        char                    ifrac_name[IFNAMSIZ];   /* デバイス名 */
        struct sockaddr         ifra_addr;              /* ホストアドレス */
        struct sockaddr         ifra_dstaddr;           /* 宛先アドレス */
#define ifra_broadaddr          ifra_dstaddr            /* ブロードキャストアドレス */
        struct sockaddr         ifra_mask;              /* ネットワークマスク */
};
```

5.4.17 ネットワークインタフェース

```
struct ifaddrs {
        struct ifaddrs*         ifa_next;               /* Pointer to next struct */
        char*                   ifa_name;               /* インタフェース名 */
        unsigned int            ifa_flags;              /* フラグ */
        struct sockaddr*        ifa_addr;               /* アドレス */
        struct sockaddr*        ifa_netmask;            /* ネットマスク */
        struct sockaddr*        ifa_broadaddr;          /* ブロードキャストアドレス */
        struct sockaddr*        ifa_dstaddr;            /* P2Pインタフェースの宛先アドレス */
        void*                   ifa_data;               /* アドレスファミリ特有データ */
};
```

5.4.18 ホスト名テーブル構造体

```
struct hosttable {
        struct sockaddr*        addr;           /* アドレス */
        char*                   host;           /* ホスト名 */
        char*                   aliases;        /* ホストの別名 */
};
```

5.4.19 ルーティングメトリック構造体

```
struct rt_metrics {
        unsigned long   rmx_locks;      /* ネットワーク通信マネージャに変更させないルーティングメトリ
ックを指定するフラグ */
        unsigned long   rmx_mtu;        /* 経路のMTU */
        unsigned long   rmx_hopcount;   /* 最大ホップ数 */
        unsigned long   rmx_expire;     /* 経路の存続期間（単位: 秒）(ARPエントリが削除されるまでの時
間) */
        unsigned long   rmx_recvpipe;   /* 受信ソケットバッファのサイズ (inbound delay-bandwidth
product) */
        unsigned long   rmx_sendpipe;   /* 送信ソケットバッファのサイズ (outbound delay-bandwidth
product) */
        unsigned long   rmx_ssthresh;   /* ゲートウェイのバッファサイズ (outbound gateway buffer
limit) */
        unsigned long   rmx_rtt;        /* ラウンドトリップタイム（ミリ秒）*/
        unsigned long   rmx_rttvar;     /* ラウンドトリップタイムの分散（ミリ秒）*/
        unsigned long   rmx_pksent;     /* 経路を用いて送信されたパケット数 */
};
```

5.4.20 ルーティングメッセージヘッダ構造体

```
struct rt_msghdr {
        unsigned short  rtm_msglen;     /* メッセージサイズ(バイト長) */
        unsigned char   rtm_version;    /* バージョン */
        unsigned char   rtm_type;       /* メッセージタイプ */
        unsigned short  rtm_index;      /* インタフェースのインデックス */
        int             rtm_flags;      /* フラグ */
        int             rtm_addrs;      /* メッセージに含まれるアドレスを示すビットマスク */
        ID              rtm_tid;        /* 送信タスクのタスクID */
        int             rtm_seq;        /* シーケンス番号 */
        int             rtm_errno;      /* メッセージ処理中に発生したエラー */
        int             rtm_use;        /* 対応する経路を用いて送信した回数 */
        unsigned long   rtm_inits;      /* 初期化するルーティングメトリックを指定するフラグ */
        struct rt_metrics rtm_rmx;      /* ルーティングメトリック */
};
```

5.4.21 ルーティングメッセージヘッダ構造体(RTM_IFINFO用)

```
struct if_msghdr {
        unsigned short  ifm_msglen;     /* メッセージサイズ（バイト長）*/
        unsigned char   ifm_version;    /* バージョン */
        unsigned char   ifm_type;       /* メッセージタイプ */
        int             ifm_addrs;      /* メッセージに含まれるアドレスを示すビットマスク */
        int             ifm_flags;      /* フラグ */
        unsigned short  ifm_index;      /* インタフェースのインデックス */
        struct if_data  ifm_data;       /* 統計情報とインタフェースに関する他の情報 */
};
```

RTM_IFINFO メッセージはこの構造体をヘッダに用いる。

5.4.22 ルーティングメッセージヘッダ構造体(RTM_NEWADDR/RTM_DELADDR用)

```
struct ifa_msghdr {
        unsigned short  ifam_msglen;    /* メッセージサイズ（バイト長）*/
        unsigned char   ifam_version;   /* バージョン */
        unsigned char   ifam_type;      /* メッセージサイズ */
        int             ifam_addrs;     /* メッセージに含まれるアドレスを示すビットマスク */
        int             ifam_flags;     /* フラグ */
        unsigned short  ifam_index;     /* インタフェースのインデックス */
        int             ifam_metric;    /* インタフェースの通信コスト */
};
```

RTM_NEWADDR, RTM_DELADDR メッセージはこの構造体をヘッダに用いる。

5.4.23 ルーティングメッセージヘッダ構造体(RTM_IFANNOUNCE用)

```
struct if_announcemsghdr {
        unsigned short  ifan_msglen;            /* メッセージサイズ(バイト長) */
        unsigned char   ifan_version;           /* バージョン */
        unsigned char   ifan_type;              /* メッセージタイプ */
        unsigned short  ifan_index;             /* インタフェースのインデックス */
        char            ifan_name[IFNAMSIZ];    /* インタフェース名 */
        unsigned short  ifan_what;              /* 通知の種類 */
};
```

RTM_IFANNOUNCE メッセージはこの構造体をヘッダに用いる。

5.5 API

5.5.1 so_main - ソケットシステムサービスの初期化・終了

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_main(INT ac, UB* arg[]);
```

【パラメータ】

INT	ac	arg[] の要素数または負の値
UB*	arg[]	引数文字列の配列

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
EX_INVAL	不正なパラメータ

【解説】

T2EXのネットワーク通信マネージャの初期化(ac >= 0)、または終了(ac < 0)を行う。

初期化時は、arg[] に ac 個の文字列を引数として渡すことができる。引数の内容は実装依存とする。

5.5.2 so_socket - 通信の端点を生成

【C言語インタフェース】

```
#include <t2ex/socket.h>

int sd = so_socket(int domain, int type, int protocol);
```

【パラメータ】

int	domain	通信を行うドメイン
int	type	ソケットタイプ
int	protocol	プロトコル

【リターンパラメータ】

int	sd	ソケットディスクリプタ、 またはエラーコード

【エラーコード】

EX_AFNOSUPPORT	指定されたアドレスファミリは実装されていない
EX_NFILE	システムで使用可能なディスクリプタ数を超えた
EX_PROTONOSUPPORT	プロトコルがサポートされていない - 指定されたアドレスファミリは指定されたプロトコルをサポートしていない - 指定されたプロトコルは実装されていない
EX_PROTOTYPE	指定されたプロトコルは指定されたソケットタイプをサポートしていない
EX_NOBUFS	操作を実行するために必要なシステムの資源が足りない

【解説】

名前付けされていないソケットを通信ドメインに生成する。

戻値のソケットディスクリプタを以後のソケット操作時のAPIコール呼出しで利用する。

protocol にソケットで利用するプロトコルを指定する。protocol に 0 を指定すると、ソケットタイプに適切な

デフォルトのプロトコルを選択する。

domain に通信ドメインで使用するアドレスファミリを指定する。次のアドレスファミリを利用できる。

AF_INET　　IPv4プロトコル
AF_ROUTE　　ルーティング情報データベース操作用

type にソケットタイプを指定する。ソケットタイプは生成するソケットで使用される通信の種類を指定する。次のソケットタイプを利用できる。

SOCK_STREAM
　　信頼性と順序性のある、全二重接続のコネクション型バイトストリームを提供する。
　　また、帯域外データを送信できる。

SOCK_DGRAM
　　最大のメッセージ長が固定長の、信頼性のない、コネクションレス型データグラムを提供する

SOCK_RAW
　　ヘッダを含む生のパケットを直接操作するインタフェースを提供する。

protocol の値が 0 ではない場合、アドレスファミリがサポートするプロトコルを指定しなければならない。
protocol の値が 0 の場合、アドレスファミリとソケットタイプに対応するデフォルトプロトコルが使用される。

5.5.3 so_close - ソケットを閉じる

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_close(int sd);
```

【パラメータ】

int	sd	ソケットディスクリプタ

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
EX_BADF	sd が不正

【解説】

sd で示されるソケットをクローズする。

ソケットディスクリプタ sd を解放し、その後の so_socket() あるいは他のAPIコール呼び出しで再利用できるようにする。ソケットに割当てられた名前情報とキュー内のデータを破棄する。

sd がコネクション型ソケットかつ so_setsockopt() に SO_LINGER を指定してリンガ間隔と呼ばれるタイムアウト時間を設定した場合、未送信メッセージをすべて送信するか、あるいはタイムアウトが発生するまで待つ。この場合、ソケットディスクリプタ sd をノンブロッキングモードに設定していても待ちに入る。

so_break() により中止された場合、so_close() は正常終了して待ちを解除し、ソケットは遅延して閉じられる。さらにリンガ間隔を設定していた場合、タイムアウトが発生した時と同様に処理を行う。

【関連項目】

so_accept, so_getsockopt, so_socket, so_socketpair, so_setsockopt

5.5.4 so_accept - ソケットへの接続の受け付け

【C言語インタフェース】

```
#include <t2ex/socket.h>

int rsd = so_accept(int sd, struct sockaddr* addr, socklen_t* addrlen);
```

【パラメータ】

int	sd	ソケットディスクリプタ
struct sockaddr*	addr	接続相手アドレス
socklen_t*	addrlen	接続相手アドレスのサイズ(バイト数)

【リターンパラメータ】

int	rsd	新規生成されたソケットディスクリプタ、 またはエラーコード
struct sockaddr*	addr	接続相手のアドレス
socklen_t*	addrlen	実際に書き込まれた接続相手のアドレスのサイズ(バイト数)

【エラーコード】

EX_AGAIN、または EX_WOULDBLOCK	sd がノンブロッキングモードに設定されていて、受付可能な接続が存在しない
EX_BADF	sd が不正
EX_CONNABORTED	接続処理が中止された
EX_FAULT	addr が書込み可能なアドレス空間にない
EX_INTR	so_break() による中止
EX_INVAL	sd は接続を受け付けない
EX_NFILE	システムで使用可能なディスクリプタ数を超えた
EX_OPNOTSUPP	sd のソケットタイプが SOCK_STREAM でない

【解説】

sd に対する新しい接続を受け付ける。

保留状態の接続要求が入っているキューから先頭の接続要求を取り出す。その後、ソケットタイプ、プロトコル、アドレスファミリが sd と等しい新しいソケットを生成し、そのソケットに対して新しいソケットディスクリプタを割当てる。なお、本APIコールは SOCK_STREAM 型ソケットに対してのみ利用できる。

sd は、so_socket() で生成され、so_bind() でアドレスが割当てられ、so_listen() で接続待ちソケットに設定されたソケットである。

addr は、NULL ポインタ、あるいは sockaddr 構造体へのポインタである。 addr が sockaddr 構造体へのポインタの場合、接続したソケットのアドレスが addr に書き込まれる。

addrlen は、socklen_t データ型へのポインタであり、addrlenの指す領域はパラメータとしてもリターンパラメータとしても利用される。so_accept() を呼び出す場合は、addr で指定した接続相手アドレスのsockaddr 構造体のサイズを *addrlen に指定する。so_accept() の実行により、addr に設定された接続相手アドレスのサイズが *addrlen に設定される。

addr が NULL ポインタではない場合、接続したソケットのアドレスが addr が指す sockaddr 構造体に格納され、そのアドレスの長さが addrlen が指す領域に格納される。

接続相手のアドレスのサイズがパラメータの addrlen より大きい場合は、 addr に格納するアドレスを切り詰める。

保留中の接続要求がキューに存在しない場合、sd の状態フラグに従い次の処理を行う。

sd に O_NONBLOCK が設定されていない場合
so_accept() を呼び出したタスクは待ちに入らない。

sd に O_NONBLOCK が設定されている場合（ノンブロッキングモード)
so_accept()は失敗してエラー EX_AGAIN を返す。
この場合、so_select() を用いてソケットが読込み準備完了になることを監視することにより、接続要求が受付可能になることを知ることができる。

接続が成立して新規生成したソケットを別の接続受付に再利用できない。オリジナルのソケット sd はオープンしたままであり、再び接続を受け付けることができる。

【関連項目】

so_bind, so_connect, so_listen, so_select, so_socket

5.5.5 so_bind - ソケットに名前をつける

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_bind(int sd, const struct sockaddr* addr, socklen_t addrlen);
```

【パラメータ】

int	sd	ソケットディスクリプタ
const struct sockaddr*	addr	アドレス
socklen_t	addrlen	アドレスのサイズ(バイト数)

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
EX_ADDRINUSE	指定されたアドレスは使用中である
EX_ADDRNOTAVAIL	指定されたアドレスはローカルマシンから利用できない
EX_BADF	sd が不正
EX_INVAL	不正なパラメータ - sd には既にアドレスが割当てられていて、それが使用するプロトコルは新しいアドレスを名前付けできない - ソケットがシャットダウンされている(全二重接続の一部が閉じられている) - addrlen はアドレスファミリに対して不正
EX_FAULT	addr が有効なアドレス空間にない

【解説】

so_socket() で生成したソケットは生成時に名前がつけられていない。そのような名前をつけられていない sd に対してローカルなソケットアドレス addr を割当てる。

sd には名前を付けるソケットディスクリプタを指定する。

addr はソケットに割当てるローカルアドレスを格納している sockaddr 構造体へのポインタである。アドレスの長さと形式はソケットのアドレスファミリに依存する。

addrlen は addr が指す sockaddr 構造体の大きさを指定する。

【関連項目】

so_connect, so_getsockname, so_listen, so_socket

5.5.6 so_connect - ソケットの接続

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_connect(int sd, const struct sockaddr* addr, socklen_t addrlen);
```

【パラメータ】

int	sd	ソケットディスクリプタ
const struct sockddr*	addr	アドレス
socklen_t	addrlen	アドレスのサイズ(バイト数)

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
EX_ADDRNOTAVAIL	指定されたアドレスはローカルマシンから利用できない
EX_AFNOSUPPORT	指定されたアドレスは指定されたソケットのアドレスファミリで利用できない
EX_ALREADY	指定されたソケットに対する接続要求は既に処理中である
EX_BADF	sd が不正
EX_CONNREFUSED	通信相手が接続要求を受け付けていない、あるいは、接続要求を拒絶した
EX_INPROGRESS	ソケットディスクリプタがノンブロッキングモードに設定されており、

	接続をすぐには確立できない（この場合、接続は非同期的に確立される）
EX_INTR	so_break() による中止(接続は非同期的に確立される)
EX_ISCONN	指定されたソケットはコネクション型ソケットであり、既に接続されている
EX_NETUNREACH	ネットワークへの経路がない
EX_TIMEDOUT	接続が確立するまえにタイムアウトした
EX_ADDRINUSE	既に使用中のアドレスを使用して接続を確立しようとした
EX_FAULT	addr が有効なアドレス空間にない
EX_INVAL	不正なパラメータ

【解説】

コネクション型ソケットの場合は接続を確立し、コネクションレス型ソケットの場合は通信相手アドレスを設定する。

sd に接続するソケットのソケットディスクリプタを指定する。

addr は通信相手のアドレスを格納している sockaddr 構造体へのポインタである。アドレスの長さと形式はソケットのアドレスファミリに依存する。

addrlen に addr が指す sockaddr 構造体のサイズを指定する。

ソケットにローカルアドレスが割り当てられていない場合、so_connect() はソケットに未使用のローカルアドレスを割当てる。

ソケットのタイプが SOCK_DGRAM の場合

so_connect() は通信相手のソケットを設定するが接続は確立しない。その後の so_send() は通信相手アドレスにデータグラムを送信し、また、so_recv() は通信相手アドレスからのデータグラムのみを受信する。以下のいずれかの条件が成立すると、ソケットの通信相手アドレスをリセットする。

- sockaddr の sa_family フィールドに AF_UNSPEC を設定した場合
- addr に NULL などの不正なアドレスを設定した場合

ソケットのタイプが SOCK_STREAM の場合

so_connect()は addr で指定されたアドレスに接続を確立しようとする。
もし直ちに接続を確立できない場合、sd の状態フラグに従い以下の処理を行う。

sd に O_NONBLOCK が設定されていない場合

接続が確立されるまで待つ。接続を確立する前に一定時間が経過してタイムアウトが発生した場合、so_connect() は失敗して接続処理は中断される。待ち状態のタスクに対して so_break() が実行されると、so_connect() は失敗して EX_INTR を返す。
この場合、接続処理は中断されず、非同期的に接続が確立される。

sd に O_NONBLOCK が設定されている場合（ノンブロッキングモード)

so_connect() は失敗して EX_INPROGRESS を返す。しかし、接続処理は中断されず、非同期的に接続が確立される。接続が確立される前に同じソケットに対して so_connect() を呼び出すと、so_connect() は失敗して EX_ALREADY を返す。

接続が非同期的に確立される場合、so_select() を用いてソケットディスクリプタが書込み可能になることを監視することにより、接続確立を知ることができる。

【関連項目】

so_accept, so_bind, so_close, so_getsockname, so_send, so_shutdown, so_socket

5.5.7 so_listen - ソケット上の接続を待つ

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_listen(int sd, int backlog);
```

【パラメータ】

int	sd	ソケットディスクリプタ
int	backlog	最大接続要求待ちキュー数

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
EX_BADF	sd が不正
EX_OPNOTSUPP	ソケットのプロトコルは so_listen() 操作をサポートしていない
EX_INVAL	不正なパラメータ

【解説】

コネクション型ソケット sd を接続待ちソケット(listen状態)に設定する。

backlog に接続待ちキューに溜められる保留中の接続要求の最大数を指定する。backlog に指定できる最大値は SOMAXCONN で定義されている。 backlog に SOMAXCONN を超えた値を指定した場合、本APIコールは backlog に SOMAXCONN を指定したとみなす。接続要求待ちキューに空きがない状態で接続要求が到達した場合、クライアントは EX_CONNREFUSED を示すエラーを受信する。あるいは、下位層のプロトコルが再送信をサポートしているならば、のちの再送信処理が成功できるようにその要求は無視される。

backlog に負の値を設定した場合、本APIコールは backlog に 0 を設定した場合と同様に処理を行う。

【関連項目】

so_accept, so_connect, so_socket

5.5.8 so_select, so_select_ms, so_select_us - 多重化I/Oの同期をとる

【C言語インタフェース】

```
#include <t2ex/socket.h>

int nfd = so_select(int nfds, fd_set* readfds, fd_set* writefds, fd_set* exceptfds, struct timeval*
tv);
int nfd = so_select_ms(int nfds, fd_set* readfds, fd_set* writefds, fd_set* exceptfds, TMO tmout);
int nfd = so_select_us(int nfds, fd_set* readfds, fd_set* writefds, fd_set* exceptfds, TMO_U tmout_u);
```

【パラメータ】

int	nfds	ソケットディスクリプタの範囲
fd_set*	readfds	読込み準備完了を調べるソケットディスクリプタの集合
fd_set*	writefds	書込み準備完了を調べるソケットディスクリプタの集合
fd_set*	exceptfds	例外発生を調べるソケットディスクリプタの集合
struct timeval*	tv	タイムアウト指定 (timeval 形式)
TMO	tmout	タイムアウト指定 (ミリ秒)
TMO_U	tmout_u	タイムアウト指定 (マイクロ秒)

【リターンパラメータ】

int	nfd	条件を満たしたディスクリプタの個数、 またはエラーコード

【エラーコード】

EX_BADF	1 つ以上のソケットディスクリプタ集合が不正なソケットディスクリプタを 指定している
EX_INTR	so_break() による中止
EX_INVAL	不正なパラメータ - タイムアウト時間が不正 - nfds が 0 以下または FD_SETSIZE より大きい
EX_FAULT	readfds, writefds, exceptfds の 1 つ以上が有効なアドレス空間を指していない

【解説】

readfds、writefds、exceptfds で指定したソケットディスクリプタ集合を監視し、各集合の中のいずれかのソケットディスクリプタが、読込み準備完了、書込み準備完了、保留中の例外の取得準備完了になるまで待つ。

nfds は評価すべきディスクリプタの範囲を示す。 so_select() は、各集合 readfds、writefds、exceptfds のソケットディスクリプタの中で、 0 から nfds-1 のディスクリプタを検査する。すなわち、3 つの集合に含まれるソケットディスクリプタの最大値に対して、1 を加えた値を nfds に設定する。

readfds が NULL ではない場合、ソケットディスクリプタの集合を保持する fd_set 型のオブジェクトを指す。readfds は、本APIコールへ渡す場合は読込み準備完了を調べるソケットディスクリプタの集合を指定し、本APIコールから返ってくる場合は読込み準備完了になったソケットディスクリプタの集合を示す。

writefds が NULL ではない場合、ソケットディスクリプタの集合を保持する fd_set 型のオブジェクトを指す。writefds は、本APIコールへ渡す場合は書込み準備完了を調べるソケットディスクリプタの集合を指定し、本APIコールから返ってくる場合は書込み準備完了になったソケットディスクリプタの集合を示す。

exceptfds が NULL ポインタではない場合、ソケットディスクリプタの集合を保持する fd_set 型のオブジェクトを指す。exceptfds は、本APIコールへ渡す場合は例外発生を調べるソケットディスクリプタの集合を指定し、本APIコールから返ってくる場合は例外が発生したソケットディスクリプタの集合を示す。

正常終了すると、これらのAPIコールは readfds, writefds, exceptfds が指すオブジェクトを、それぞれ読込み準備完了、書込み準備完了、保留中の例外の取得準備完了になったディスクリプタ集合に変更し、準備完了になったディスクリプタの合計数を返す。入力時に指定して、かつそのディスクリプタの条件が成立した場合、 nfds より小さな値であれば、そのディスクリプタに対応するビットを各集合に設定する。

要求した操作に対して準備完了したディスクリプタがない場合、以下のいずれかの条件を満たすまで待つ。
- 少なくとも 1 つの要求した操作が準備完了になる。
- タイムアウトが発生する。
- so_break() で待ちが解除される。

tv, tmout, tmout_u にソケットディスクリプタが要求を満たすまでのタイムアウト時間を指定し、それぞれ so_select(), so_select_ms(), so_select_us() で使用する。タイムアウトするまでの相対時間を、tv は timeout 構造体を使用して、tmout はミリ秒単位で、 tmout_u はマイクロ秒単位で指定する。

tv, tmout, tmout_u に 0 秒より大きな時間を設定した場合、ソケットディスクリプタが準備完了になるまでの最大待ち時間を示す。要求された操作が準備完了になる前に指定された時間が経過した場合、これらのAPIコールはタスクの待ちを解除して処理を戻す。 tmout, tmout_u に TMO_FEVR 、または tv に NULL を指定した場合、これらのAPIコールは少なくとも 1 つのソケットディスクリプタが要求を満たすまで無期限にタスクを待たせる。ポーリングする場合は、 tmout, tmout_u に TMO_POL、 または tv に 0 の値を指定する。

so_select() は tv の値を変更せず、その後のAPIコール呼び出しで再利用できる。ただし、POSIX仕様では tv の値を変更してもしなくてもどちらでも良いと規定している(例、FreeBSD、NetBSD は timeout を変更しないが、Linux は timeout を変更する)。アプリケーションの移植性を考慮した場合、ユーザープログラムは tv の値を so_select() の呼び出し毎に再初期化することをすすめる。

ソケットディスクリプタを読込み準備完了とみなす条件

so_recvmsg() に渡したソケット
パラメータにより通常データと補助データを受信することができ、いずれかの種類のデータを受信した場合、そのソケットを読込み可能であるとみなす。また、ソケットオプション SO_OOBINLINE が有効な場合は、帯域外データを通常データと同じキューに挿入する。この場合、帯域外データ受信時もソケットを読込み可能であるとみなす。

so_accept() に渡したソケット
接続要求を受信した場合、そのソケットを読込み可能であるとみなす。

ソケットディスクリプタを書込み準備完了とみなす条件

so_sendmsg() に渡したソケット
最小単位の通常データを送信可能になった場合、そのソケットを書込み可能であるとみなす。

so_connect() に渡したソケット
接続に成功した場合、そのソケットを書込み可能であるとみなす。また、保留中のエラーを残して接続失敗した場合、そのソケットも書込み可能であるとみなす。

ソケットディスクリプタに例外条件が保留されているとみなす条件

- ノンブロッキングモードに設定していないソケットディスクリプタに対して MSG_OOB フラグを設定して受信操作
  をしたときにブロックせずに帯域外データを返せる場合（帯域外データを読み込むために MSG_OOB フラグを設定する必要があるかどうかはプロトコルに依存する）
- 受信キューの中に帯域外データマークが存在する場合
- その他、プロトコルに依存した例外条件がある。

readfds, writefds, exceptfds のすべてに NULL を設定し、かつ tmout_u, tmout, tv に 0 以上の時間を設定した場合、so_select() は指定された時間が経過するまで、あるいは so_break() で処理が中止されるまでタスクを待たせる。

readfds, writefds, exceptfds のすべてに NULL を設定し、かつ tmout_u, tmout, tv に無制限の待ち時間を設定した場合(tmout_u, tmout は TMO_FEVR、 tv は NULL の場合)、so_select() は so_break() で処理が中止さ

れるまでタスクを待たせる。

これらのAPIコールが失敗した場合は、 readfds, writefds, exceptfds が指すオブジェクトを変更しない。ソケットディスクリプタが指定された条件を満たすことなくタイムアウト時間が経過した場合、 readfds, writefds, exceptfds が指すオブジェクトのビットをすべて 0 に設定する。

fd_set 型のソケットディスクリプタマスクの初期化や評価に、マクロ FD_CLR(), FD_ISSET(), FD_SET(), FD_ZERO() を用いる。

FD_CLR(fd, fdsetp)
　　fdsetp が指す集合からソケットディスクリプタ fd を取り除く。
　　fd がこの集合に含まれていない場合、このマクロは集合に対して何も操作せず、
　　また、エラーも返さない。

FD_ISSET(fd, fdsetp)
　　ソケットディスクリプタ fd が fdsetp が指す集合に含まれているか評価して、含まれている場合は
　　非ゼロを返し、そうでなければゼロを返す。

FD_SET(fd, fdsetp)
　　ソケットディスクリプタ fd を fdsetp が指す集合に付け加える。
　　ソケットディスクリプタ fd が既に集合に含まれているならば、このマクロは集合に対して何も操作せず、
　　また、エラーも返さない。

FD_ZERO(fdsetp)
　　fdsetp が指すディスクリプタ集合を空集合に初期化する。

以下のいずれかの条件を満たす場合、これらのマクロの動作は未定義である。
　　- fd の値が 0 以下またはFD_SETSIZE 以上の場合
　　- fd が有効なソケットディスクリプタではない場合
　　- 引数のいずれかが副作用を含む式の場合

5.5.9 so_read - ソケットからの読込み

【C言語インタフェース】

```
#include <t2ex/socket.h>

int nb = so_read(int sd, void* buf, size_t count);
```

【パラメータ】

int	sd	ソケットディスクリプタ
void*	buf	受信バッファ
size_t	count	受信バッファのサイズ(バイト数)

【リターンパラメータ】

int	nb	読み込んだバイト数、 またはエラーコード
void*	buf	読み込まれたデータ

【エラーコード】

EX_AGAIN、または EX_WOULDBLOCK	ソケットディスクリプタがノンブロッキングモードに設定されていて、 読込み可能なデータがない
EX_BADF	sd が不正
EX_INTR	so_break() による中止 (データは受信されていない)
EX_IO	ソケットからの読込み時にI/Oエラー
EX_FAULT	buf が有効なアドレス空間にない
EX_INVAL	不正なパラメータ
EX_NOTCONN	接続が確立されていないコネクション型ソケットで受信しようとした

【解説】

ソケットディスクリプタ sd から count バイトのデータを buf で指定されたバッファに読み込む。

count が 0 の場合は何も処理を行わない。パラメータエラーがなければ正常終了し、エラーがあれば対応するエラーコードを返す。

count が SSIZE_MAX より大きい場合、EX_INVAL のエラーを返す。

現在利用可能なデータがない場合、sd の状態フラグに従い次の処理を行う。

sd に O_NONBLOCK が設定されていない場合
データが読込み可能になるまで、呼び出したタスクを待たせる。

sd に O_NONBLOCK が設定されている場合（ノンブロッキングモード)
EX_AGAIN のエラーを返す。

読込み可能なデータがある場合、 O_NONBLOCK フラグは何も影響を与えない。

count が 0 より大きい場合に正常終了すると、so_read()は読み込んだバイト数を返す。この値は count より大きくなることはない。読込み可能なデータが count バイトより少ない場合、戻値は count よりも小さな値になる。

データを読み込む前に so_break() により中止されると、EX_INTR のエラーを返す。1 バイト以上のデータを読み込んだ後に中止された場合は、それまでに読み込んだバイト数を返す。

so_read() は、so_recv()にフラグを何も指定しない時と等しい処理を行う。

【関連項目】

so_fcntl, so_ioctl, so_recv, so_select, so_socket, so_sockpair

5.5.10 so_recv, so_recvfrom, so_recvmsg - ソケットからのメッセージ受信

【C言語インタフェース】

```
#include <t2ex/socket.h>

int nb = so_recv(int sd, void* buf, size_t len, int flags);
int nb = so_recvfrom(int sd, void* buf, size_t len, int flags, struct sockaddr* src_addr, socklen_t*
addrlen);
int nb = so_recvmsg(int sd, struct msghdr* msg, int flags);
```

【パラメータ】

int	sd	ソケットディスクリプタ
void*	buf	受信バッファ
size_t	len	受信バッファのサイズ(バイト数)
struct msghdr*	msg	メッセージヘッダ
int	flags	フラグ
struct sockaddr*	src_addr	送信元アドレス
socklen_t*	addrlen	送信元アドレスのサイズ(バイト数)

【リターンパラメータ】

int	nb	読み込んだバイト数、 またはエラーコード
void*	buf	読み込んだデータ

【エラーコード】

EX_AGAIN、または EX_WOULDBLOCK	ソケットディスクリプタがノンブロッキングモードに設定されていて、 読込み可能なデータがない。あるいは、MSG_OOB が設定されていて読込み可能な 帯域外データがなく、かつソケットディスクリプタがノンブロッキングモードに設定 されているかソケットが帯域外データのブロックによる受信待ちをサポートしていない
EX_BADF	sd が不正
EX_INTR	so_break() による中止（データは受信されていない)
EX_INVAL	不正なパラメータ - iov_lenの合計値が SSIZE_MAX より大きい - MSG_OOB が設定されているが利用可能な帯域外データがない
EX_NOTCONN	接続が確立されていないコネクション型ソケットで受信しようとした
EX_OPNOTSUPP	指定されたフラグは指定されたソケットのソケットタイプまたはプロトコルで サポートされていない
EX_FAULT	受信バッファが有効なアドレス空間にない
so_recvmsg()のみ	
EX_MSGSIZE	msg 構造体のメンバ msg_iovlen が 0 以下の値である

あるいは IOV_MAX より大きな値である

【解説】

ソケットディスクリプタ sd から引数で指定したバッファにデータを受信する。

so_recvfrom(), so_recvmsg()は、コネクション型ソケットとコネクションレス型ソケットに関係なくメッセージを受信できる。アプリケーションは受信データの送信元アドレスを取得できるため、これらのAPIコールをコネクションレス型ソケット使用時に利用することが多い。

so_recv()は、通常、コネクション型ソケットとともに利用する。コネクションレス型ソケットからもメッセージを受信できるが、受信データの送信元アドレスは取得できない。

sd にソケットディスクリプタを指定する。

buf はメッセージを格納するバッファへのポインタである。

len に buf が指すバッファのサイズをバイト数で指定する。

msg は msghdr 構造体へのポインタである。この構造体は、送信元のアドレスと受信メッセージをそれぞれ格納するバッファを持つ。アドレスのサイズと形式は、ソケットのアドレスファミリに依存する。

so_connect() で sd に通信相手が設定されていない場合、msghdr 構造体のメンバ msg_name と msg_namelen に送信元アドレスを指定する。so_connect() で sd に通信相手が設定されている場合、メンバ msg_name と msg_namelen は無視される。送信元アドレスが必要ない場合、メンバ msg_name に NULL を指定する。
msghdr 構造体のメンバ msg_flags の値は、本APIコールへ渡す時は無視されるが、本APIコールから返ってくる時に意味のあるデータが格納される。so_recvmsg()の処理が正常終了すると、受信メッセージの状態が以下のフラグの論理和が msg_flags に保持される。

MSG_EOR	レコード終端(End-of-record)を受信した(プロトコルがサポートする場合)。
MSG_OOB	帯域外データを受信した。
MSG_TRUNC	通常データが切り詰められた。
MSG_CTRUNC	制御データが切り詰められた。

msghdr 構造体のメンバ msg_iov と msg_iovlen は受信データを保持する場所を指定する。
msg_iov に iovec 構造体の配列を指定し、msg_iovlen にこの配列の要素数を設定する。

msghdr 構造体のメンバ msg_iov が指す各 iovec 構造体は、フィールド iov_base に保存領域を指定し、
フィールド iov_len にその領域のサイズのバイト数を指定する。すべての受信データを保存するまで、あるいはすべての領域を満たさすまで、msg_iov が指す各保存領域へ順番に受信データを保存する。

msghdr 構造体の msg_control にプロトコルの補助データを書き込む cmsghdr 構造体へのポインタを指定し、msg_controllen に msg_control のサイズをバイト数で指定する。

flags にメッセージ受信時のオプションを指定する。次の値の論理和、または 0 を指定する。

MSG_PEEK
受信したメッセージに何の変更も加えず、ただ内容を読み取る。この時、データを未読として扱い、次の so_recv(), so_recvfrom(), so_recvmsg() 呼び出し時に読み込むことができる。

MSG_OOB
帯域外データを要求する。帯域外データの意味や動作はプロトコル固有である。

MSG_WAITALL
SOCK_STREAM 型ソケットの場合、このフラグを指定すると、すべてのデータを受信するまで待つ。
以下のいずれか条件を満たした場合、受信できたデータサイズが指定したサイズに満たないことがある。
- SOCK_DGRAM 型のようなメッセージベースのソケットの場合
- so_break()により中止された場合
- 接続が閉じられた場合
- MSG_PEEK が指定されている場合
- ソケットに対する保留中のエラーがある場合

src_addr に NULL ポインタ、あるいは sockaddr 構造体へのポインタを指定する。NULL ポインタ以外の場合、メッセージの送信元のアドレスを格納する。アドレスの長さと形式はソケットのアドレスファミリに依存する。

addrlen に src_addr が指す sockaddr 構造体のサイズをバイト数で指定する。

so_recv()とso_recvfrom()は、 buf に書き込んだメッセージの長さを返し、so_recvmsg() は、メッセージ全体の長さを返す。SOCK_DGRAM 型のようなメッセージベースのソケットの場合、一つの操作でメッセージ全体を読み込

む。メッセージが長過ぎてバッファに収まらず、かつ MSG_PEEK を flags に設定していない場合、超過したデータは破棄する。さらに、so_recvmsg()の場合は msghdr 構造体のメンバ msg_flags に MSG_TRUNC を設定する。SOCK_STREAM 型のようなストリームベースのソケットの場合、メッセージ境界を無視する。この場合、読込み可能になったデータをすぐにユーザーに返し、データを破棄することはない。

MSG_WAITALL フラグを設定していないと、最初のメッセージ分しかデータを読込まない。

ソケット sd に読込み可能なメッセージがない場合、sd の状態フラグに従い次の処理を行う。

sd に O_NONBLOCK が設定されていない場合
メッセージが到着するまで待つ。

sd に O_NONBLOCK が設定されている場合（ノンブロッキングモード)
これらのAPIコールは失敗し EX_AGAIN を返す。

すべてのプロトコルがメッセージの送信元アドレスを提供するわけではない。so_recvfrom() は、 src_addr が NULL ではなく、かつプロトコルがメッセージの送信元アドレスを提供する場合のみ、メッセージの送信元アドレスを src_addr が指す sockaddr 構造体に格納し、このアドレスのサイズを addrlen が指すオブジェクトに保持する。

実際のアドレスのサイズが本APIコールに与えられた sockaddr 構造体の長さより大きい場合、保持するアドレスを切り詰める。

src_addr が NULL ではなく、かつプロトコルがメッセージの送信元アドレスを提供しない場合、 src_addr が指すオブジェクトに格納される値は不定である。

5.5.11 so_write - ソケットへの書込み

【C言語インタフェース】

```
#include <t2ex/socket.h>

int nb = so_write(int sd, const void* buf, size_t count);
```

【パラメータ】

int	sd	ソケットディスクリプタ
const void*	buf	書込みバッファ
size_t	count	書込みサイズ(バイト数)

【リターンパラメータ】

int	nb	書き込んだバイト数、 またはエラーコード

【エラーコード】

EX_AGAIN、または EX_WOULDBLOCK	ソケットディスクリプタがノンブロッキングモードに設定されていて、 要求された操作が直ちに完了せず遅延される
EX_BADF	sd が不正
EX_INTR	so_break() による中止
EX_FAULT	buf で指定された領域が有効なアドレス空間にない
EX_HOSTUNREACH	メッセージが宛先に到達できない
EX_HOSTDOWN	宛先がローカルのサブネットに存在し、arp に対して反応しない
EX_INVAL	不正なパラメータ
EX_NOTCONN	接続が確立されていないコネクション型ソケットで送信しようとした

【解説】

sd で示されたソケットに、buf で指定されたバッファから count バイト書き込む。

count が 0 の場合、何も送信せずに正常終了する。

書込み開始前に so_break() により中止された場合は、so_write() は EX_INTR を返す。

1 バイト以上のデータを正常に書き込んだ後に so_break() により中止された場合、so_write() は書き込んだバイト数を返す。

count が SSIZE_MAX より大きい場合、EX_INVAL のエラーを返す。

データを直ちに書き込めない場合、sd の状態フラグに従い次の処理を行う。

sd に O_NONBLOCK が設定されていない場合
データを受け付けられるようになるまで呼び出したタスクを待たせる。

sd に O_NONBLOCK が設定されている場合（ノンブロッキングモード)
EX_AGAIN のエラーを返す。

so_write() は so_send() にフラグを何も指定しない時と等しい処理を行う。

【関連項目】

so_read, so_select

5.5.12 so_send, so_sendto, so_sendmsg - ソケットへのメッセージ送信

【C言語インタフェース】

```
#include <t2ex/socket.h>

int nb = so_send(int sd, const void* buf, size_t len, int flags);
int nb = so_sendto(int sd, const void* buf, size_t len, int flags, const struct sockaddr* dest_addr,
socklen_t addrlen);
int nb = so_sendmsg(int sd, const struct msghdr* msg, int flags);
```

【パラメータ】

int	sd	ソケットディスクリプタ
const void*	buf	送信バッファ
size_t	len	送信データのサイズ(バイト数)
const struct msghdr*	msg	メッセージヘッダ
int	flags	フラグ
const struct sockaddr*	dest_addr	宛先アドレス
socklen_t	addrlen	宛先アドレスのサイズ(バイト数)

【リターンパラメータ】

int	nb	読み込んだバイト数、 またはエラーコード

【エラーコード】

EX_AGAIN、または EX_WOULDBLOCK	ソケットディスクリプタがノンブロッキングモードに設定されていて、 要求された操作が直ちに完了せず遅延される
EX_BADF	sd が不正
EX_INTR	so_break() による中止
EX_INVAL	不正なパラメータ - iov_lenの合計値が SSIZE_MAX より大きい
EX_NOBUFS	操作を実行するために必要なシステムの資源が足りない
EX_FAULT	引数で指定された領域が有効なアドレス空間にない
EX_DESTADDRREQ	ソケットはコネクション型ではなく、通信アドレスが設定されていない
EX_HOSTUNREACH	メッセージが宛先に到達できない
EX_HOSTDOWN	宛先がローカルのサブネットに存在し、arp に対して反応しない
EX_AFNOSUPPORT	指定されたアドレスファミリのアドレスはこのソケットで利用できない。
EX_NOTCONN	接続が確立されていないコネクション型ソケットで送信しようとした
so_sendto()のみ	
EX_ISCONN	宛先アドレスが指定されていて、ソケットは既に接続されている
so_sendmsg()のみ	
EX_MSGSIZE	msghdr 構造体のメンバ msg_iovlen が 0 以下の値である、 あるいは IOV_MAX より大きな値である

【解説】

ソケットディスクリプタ sd に引数で指定したデータを送信する。

so_send() は、指定したソケットからその通信相手へメッセージ送信を開始する。so_send() は、ソケットが接続している場合のみメッセージを送信できる。ソケットがコネクションレス型の場合、事前に設定された通信相手にメッセージを送信する。

so_sendto() は、コネクション型ソケットとコネクションレス型ソケットに関係なくメッセージを送信できる。ソケットがコネクションレス型の場合、事前に通信相手を設定していなければ dest_addr が示すアドレスにメッセージを送信する。事前に通信相手アドレスを設定した場合、エラー EX_ISCONN を返す。ソケットがコネクション型の場合、dest_addr を無視する。

so_sendmsg() は、コネクション型ソケットとコネクションレス型ソケットに関係なくメッセージを送信できる。ソケットがコネクションレス型の場合、事前に通信相手を設定していなければ msghdr で指定したアドレスにメッセージを送信する。通信相手アドレスを事前に設定した場合、msghdr で指定したアドレスにメッセージを送信する(事前に設定されたアドレスを上書きする)。ソケットがコネクション型の場合、msghdr 内の宛先アドレスを無視する。

sd にソケットディスクリプタを指定する。

buf は送信するメッセージを格納しているバッファへのポインタである。

len に送信するメッセージのサイズをバイト数で指定する。

msg は、宛先アドレスと送信メッセージのバッファを格納している msghdr 構造体へのポインタである。アドレスのサイズと形式はソケットのアドレスファミリに依存する。メンバ msg_flags は使用しない。

msg の msg_iov と msg_iovlen フィールドは、送信されるデータを格納している 0以上のバッファを示す。
msg_iov は iovec 構造体の配列へのポインタである。msg_iovlen にこの配列の要素数を設定する。
各 iovec の iov_base フィールドは保存領域を指し示し、iov_len はそのサイズのバイト数を示す。
これらのサイズのいくつかはゼロでもよい。msg_iov が指す保存領域からデータを順に送信する。

flags は、送信メッセージのタイプを指定する。flags には、次のフラグの論理和、または 0 を指定する。

MSG_EOR
(プロトコルがサポートするならば)レコードを終端する。

MSG_OOB
帯域外通信をサポートするソケットから帯域外データを送信する。
帯域外データの意味や動作はプロトコルに依存する。

dest_addr は宛先アドレスを格納している sockaddr 構造体を指す。アドレスの長さと形式はプロトコルに依存する。

addrlen に dest_addr が指す sockaddr 構造体のサイズをバイト数で指定する。

送信するメッセージの長さを len で指定する。メッセージサイズが大き過ぎて下層のプロトコルへ渡せない場合、so_send() は失敗して何もデータを送信しない。

so_send(), so_sendto(), so_sendmsg() が正常終了してもメッセージの配送を保証することはない。ローカルにエラーを検出した場合のみエラーを返す。

送信ソケットの送信メッセージの保存領域を直ちに利用できない場合、sd の状態フラグに従い次の処理を行う。

sd に O_NONBLOCK が設定されていない場合
so_send(), so_sendto(), so_sendmsg() は領域が利用可能になるまでタスクをブロックする。

sd に O_NONBLOCK が設定されている場合 (ノンブロッキングモード)
so_send(), so_sendto(), so_sendmsg() は失敗して E_AGAIN を返す。
so_select() で、データを送信できるかどうか調べることができる。

ソケットプロトコルがブロードキャスト送信をサポートし、かつ指定したアドレスが利用可能なブロードキャストアドレスの場合、so_sendto(), so_sendmsg() は SO_BROADCAST がソケットに設定されていなければ失敗する。

5.5.13 so_getpeername - 相手ソケットの名前取得

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_getpeername(int sd, struct sockaddr* addr, socklen_t* addrlen);
```

【パラメータ】

int	sd	ソケットディスクリプタ
struct sockaddr*	addr	アドレス

socklen_t*	addrlen	アドレスのサイズ(バイト数)

【リターンパラメータ】

ER	ercd	エラーコード
struct sockaddr*	addr	相手ソケットのアドレス
socklen_t*	addrlen	実際に書き込まれた相手ソケットのアドレスのサイズ(バイト数)

【エラーコード】

E_OK	正常終了
EX_BADF	sd が不正
EX_NOTCONN	指定されたソケットは接続を確立していない、あるいは事前に通信相手を設定していない
EX_NOBUFS	操作を実行するために必要なシステムの資源が足りない
EX_FAULT	addr が有効なアドレス空間にない
EX_INVAL	不正なパラメータ

【解説】

ソケット sd に接続されている通信相手のアドレスを取得する。

取得したアドレスを addr が指す sockaddr 構造体に格納し、アドレスのサイズを addrlen に格納する。

sd に通信相手のアドレスを調べるソケットのソケットディスクリプタを指定する。

addr は、NULL ポインタ、あるいは sockaddr 構造体へのポインタである。 addr が sockaddr 構造体へのポインタの場合、接続したソケットのアドレスを addr に書き込む。

addrlen は、socklen_t データ型へのポインタであり、本APIコールを呼び出す時と本APIコールから戻る時の双方で利用される。APIコールを呼び出す時、addrlen はAPIコールに渡した sockaddr 構造体のサイズを指定する。APIコールから戻る時、addrlen には書き込まれたアドレスのサイズを格納する。

取得された実際のアドレスのサイズが addr よりも大きい場合、保持できない領域を切り詰めたアドレスを addr に格納する。

【関連項目】

so_accept, so_bind, so_getsockname, so_socket

5.5.14 so_getsockname - ソケットの名前取得

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_getsockname(int sd, struct sockaddr* addr, socklen_t* addrlen);
```

【パラメータ】

int	sd	ソケットディスクリプタ
struct sockaddr*	addr	アドレス
socklen_t*	addrlen	アドレスのサイズ(バイト数)

【リターンパラメータ】

ER	ercd	エラーコード
struct sockaddr*	addr	ローカルに割当てたアドレス
socklen_t*	addrlen	実際に書き込まれたアドレスのサイズ(バイト数)

【エラーコード】

E_OK	正常終了
EX_BADF	sd が不正
EX_INVAL	指定されたソケットがシャットダウンされた
EX_NOBUFS	操作を実行するために必要なシステムの資源が足りない
EX_FAULT	addr が有効なアドレス空間にない

【解説】

sd に対してローカルに割当てた名前を取得する。

ローカルな名前を addr が指す sockaddr 構造体に格納して、名前のサイズを addrlen に格納する。

sd にローカルな名前を調べるソケットのソケットディスクリプタを指定する。

addr は、NULL ポインタ、あるいは sockaddr 構造体へのポインタである。 addr が sockaddr 構造体へのポインタの場合、そこに接続したソケットのアドレスを書き込む。

addrlen は、socklen_t データ型へのポインタであり、本APIコールを呼び出す時と本APIコールから戻る時の双方で利用される。APIコールを呼び出す時、addrlen はAPIコールに渡した sockaddr 構造体の長さを指定する。APIコールから戻る時、addrlen に書き込まれたアドレスのサイズを格納する。

取得された実際の名前のサイズが addr よりも大きい場合、保持できない領域を切り詰めた名前を addr に格納する。

sd にローカルな名前を割り当てていない場合、addr に保持される値は不定である。

【関連項目】

so_accept, so_bind, so_getpeername, so_socket

5.5.15 so_getsockopt - ソケットオプションの取得

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_getsockopt(int sd, int level, int optname, void* optval, socklen_t* optlen);
```

【パラメータ】

int	sd	ソケットディスクリプタ
int	level	レベル
int	optname	オプション
void*	optval	オプションの値
socklen_t*	optlen	オプションのサイズ(バイト数)

【リターンパラメータ】

ER	ercd	エラーコード
void*	optval	取得されたオプションの値
socklen_t*	optlen	実際に書き込まれたオプションの値のサイズ(バイト数)

【エラーコード】

E_OK	正常終了
EX_BADF	sd が不正
EX_INVAL	不正なパラメータ - 指定されたオプションは指定されたソケットレベルに対して不正である - 指定されたソケットがシャットダウンされた
EX_NOPROTOOPT	指定されたソケットのプロトコルは指定されたオプションをサポートしない
EX_FAULT	optval, または optlen が有効なアドレス空間にない

【解説】

ソケットディスクリプタ sd の optname で指定したオプションの値を取得する。

オプションの値のサイズが optlen よりも大きい場合、その値を切り詰めて optval に格納する。それ以外の場合、実際に格納された値のサイズを示すように optlen の値を変更する。

level にオプションが属するプロトコルレベルを指定する。ソケットレベルのオプションを取得する場合、SOL_SOCKET を level に指定する。他のレベルのオプションを取得する場合、オプションを管理しているプロトコルに対応する適切なレベル識別子を与える。例えば、TCP が解釈するオプションを示すためには、IPPROTO_TCP を level に指定する。以下のレベル識別子が利用できる。

IPPROTO_IP	IPレベル
IPPROTO_TCP	TCPレベル
SOL_SOCKET	ソケットレベル

optname に値を取得するオプションを指定する。

level に IPPROTO_IP を指定した場合、optname には以下のオプションを指定できる。

IP_OPTIONS	送信する各パケットのIPヘッダに埋め込むIPオプション
IP_HDRINCL	アプリケーションによる送信データへIPヘッダ付与の有効・無効

level に IPPROTO_TCP を指定した場合、optname には以下のオプションを指定できる。

TCP_NODELAY	データを直ちに送信することの許可・禁止
TCP_MAXSEG	最大セグメント長

level に SOL_SOCKET を指定した場合、optname には以下のオプションを指定できる。

SO_DEBUG	下位層プロトコルのデバッグ情報の有効・無効
SO_ACCEPTCONN	so_listen() による接続待ち状態の確認
SO_REUSEADDR	ローカルアドレスの再利用の有効・無効
SO_KEEPALIVE	定期的なキープアライブメッセージ送信の有効・無効
SO_DONTROUTE	標準のルーティング機能をバイパスしたメッセージ送信の有効・無効
SO_BROADCAST	ブロードキャストメッセージ送信の有効・無効
SO_USELOOPBACK	ハードウェアをバイパスして通信する機能の有効・無効
SO_LINGER	未送信のデータがある場合にソケットを閉じた時の動作
SO_OOBINLINE	帯域外データを通常の受信キューへ挿入する機能の有効・無効
SO_REUSEPORT	ローカルアドレスとポートの再利用の有効・無効
SO_TIMESTAMP	受信したデータグラムへのタイムスタンプ追加の有効・無効
SO_SNDBUF	送信バッファのサイズ（バイト数）
SO_RCVBUF	受信バッファのサイズ（バイト数）
SO_SNDLOWAT	ブロックせずに書込み可能な送信バッファの最小の空きサイズ（バイト数）
SO_RCVLOWAT	ブロックせずに読み取り可能な受信バッファの最小の受信データサイズ（バイト数）
SO_SNDTIMEO	送信タイムアウト
SO_RCVTIMEO	受信タイムアウト
SO_ERROR	非同期に発生した保留中のエラー
SO_TYPE	ソケットタイプ

【関連項目】

so_ioctl, so_select, so_socket

5.5.16 so_setsockopt - ソケットオプションの設定

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_setsockopt(int sd, int level, int optname, const void* optval, socklen_t optlen);
```

【パラメータ】

int	sd	ソケットディスクリプタ
int	level	レベル
int	optname	オプション
const void*	optval	オプションの値
socklen_t	optlen	オプションのサイズ(バイト数)

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
EX_BADF	sd が不正
EX_INVAL	不正なパラメータ - 指定されたオプションは指定されたソケットレベルに対して不正である - 指定されたソケットがシャットダウンされた
EX_NOPROTOOPT	指定されたソケットのプロトコルは指定されたオプションをサポートしない
EX_FAULT	optval, または optlen が有効なアドレス空間にない

【解説】

ソケットディスクリプタ sd に対して optname で指定したオプションに optval が指す値を設定する。

level にオプションが属するプロトコルレベルを指定する。ソケットレベルのオプションを設定する場合、

SOL_SOCKET を level に指定する。他のレベルのオプションを設定する場合、オプションを管理しているプロトコルに対応する適切なレベル識別子を与える。例えば、TCP が解釈するオプションを示すためには、IPPROTO_TCP を level に設定する。以下のレベル識別子が利用できる。

IPPROTO_IP	IPレベル
IPPROTO_TCP	TCPレベル
SOL_SOCKET	ソケットレベル

optname は値を設定するオプションを指定する。

level に IPPROTO_IP を指定した場合、optname には以下のオプションを指定できる。

IP_OPTIONS	送信する各パケットのIPヘッダに埋め込むIPオプション
IP_HDRINCL	アプリケーションによる送信データへIPヘッダ付与の有効・無効

level に IPPROTO_TCP を指定した場合、optname には以下のオプションを指定できる。

TCP_NODELAY	データを直ちに送信することの許可・禁止
TCP_MAXSEG	最大セグメント長

level に SOL_SOCKET を指定した場合、optname には以下のオプションを指定できる。

SO_DEBUG	下位層プロトコルのデバッグ情報の有効・無効
SO_REUSEADDR	ローカルアドレスの再利用の有効・無効
SO_KEEPALIVE	定期的なキープアライブメッセージ送信の有効・無効
SO_DONTROUTE	標準のルーティング機能をバイパスしたメッセージ送信の有効・無効
SO_BROADCAST	ブロードキャストメッセージ送信の有効・無効
SO_USELOOPBACK	ハードウェアをバイパスして通信する機能の有効・無効
SO_LINGER	未送信のデータがある場合にソケットを閉じた時の動作
SO_OOBINLINE	帯域外データを通常の受信キューへ挿入する機能の有効・無効
SO_REUSEPORT	ローカルアドレスとポートの再利用の有効・無効
SO_TIMESTAMP	受信したデータグラムへのタイムスタンプ追加の有効・無効
SO_SNDBUF	送信バッファのサイズ（バイト数）
SO_RCVBUF	受信バッファのサイズ（バイト数）
SO_SNDLOWAT	ブロックせずに書込み可能な送信バッファの最小の空きサイズ（バイト数）
SO_RCVLOWAT	ブロックせずに読み取り可能な受信バッファの最小の受信データサイズ（バイト数）
SO_SNDTIMEO	送信タイムアウト
SO_RCVTIMEO	受信タイムアウト

5.5.17 so_gethostname - ホスト名を取得

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_gethostname(char* name, size_t len);
```

【パラメータ】

char*	name	ホスト名を格納する領域
size_t	len	ホスト名を格納する領域のサイズ(バイト数)

【リターンパラメータ】

ER	ercd	エラーコード
char*	name	ホスト名

【エラーコード】

E_OK	正常終了
EX_FAULT	name で指定された領域が有効なアドレス空間にない
EX_INVAL	不正なパラメータ

【解説】

現在のマシンに設定されている標準のホスト名を返す。

len に name が指す配列のサイズを指定する。取得される名前はヌル文字で終端する文字列である。ただし、 len に指定したサイズがホスト名のサイズより小さい場合、取得される名前を切り詰める。この時、その文字列がヌル文字で終端するかどうかは不定である。

ホスト名の最大長は HOST_NAME_MAX である。

【関連項目】

so_sethostname

5.5.18 so_sethostname - ホスト名を設定

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_sethostname(const char* name, size_t len);
```

【パラメータ】

const char*	name	ホスト名
size_t	len	ホスト名のサイズ(バイト数)

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
EX_FAULT	name で指定された領域が有効なアドレス空間にない
EX_INVAL	不正なパラメータ

【解説】

name で指定されたホスト名を現在のマシンに設定する。

len に name が指す配列のサイズを指定する。本APIコールは、通常、起動直後に使用される。

【関連項目】

so_gethostname

5.5.19 so_getaddrinfo, so_getaddrinfo_ms, so_getaddrinfo_us - ネットワークのアドレスとサービスの変換

【C言語インタフェース】

```
#include <t2ex/socket.h>

int len = so_getaddrinfo(const char* node, const char* service, const struct addrinfo* hints, struct
addrinfo** res, void* buf, size_t bufsz, struct timeval* timeout);
int len = so_getaddrinfo_ms(const char* node, const char* service, const struct addrinfo* hints,
struct addrinfo** res, void* buf, size_t bufsz, TMO tmout);
int len = so_getaddrinfo_us(const char* node, const char* service, const struct addrinfo* hints,
struct addrinfo** res, void* buf, size_t bufsz, TMO_U tmout_u);
```

【パラメータ】

const char*	node	ホスト名などのサービスロケーション
const char*	service	サービス名
const struct addrinfo*	hints	名前解決時に使用されるヒント
struct addrinfo**	res	名前解決結果のリンクリストの先頭へのポインタ
void*	buf	名前解決結果を格納するバッファ
size_t	bufsz	名前解決結果格納用バッファのサイズ(バイト数)
struct timeval*	timeout	タイムアウト指定 (timeval 形式)
TMO	tmout	タイムアウト指定 (ミリ秒)
TMO_U	tmout_u	タイムアウト指定 (マイクロ秒)

【リターンパラメータ】

int	len	名前解決結果を格納するために必要なバッファサイズ (バイト数) 、

		またはエラーコード
struct addrinfo**	res	名前解決した結果のリンクリストの先頭へのポインタ
void*	buf	名前解決した結果のリンクリスト

【エラーコード】

EX_TIMEDOUT	処理が完了する前にタイムアウトが発生した
EX_INTR	so_break() による中止
EX_INVAL	不正なパラメータ
EX_AI_AGAIN	名前解決が一時的に失敗した(名前解決を再試行すると成功するかもしれない)
EX_AI_BADFLAGS	ai_flags の値が不正
EX_AI_FAIL	名前解決時に回復不可能なエラーが発生した
EX_AI_FAMILY	サポートされていないアドレスファミリ
EX_AI_NONAME	名前が渡されていない (名前は node と service の少なくとも一方の変数で名前を渡す必要ある)
EX_AI_SERVICE	指定されたソケットのソケットタイプは service で渡されたサービスを サポートしていない
EX_AI_SOCKTYPE	ソケットタイプはサポートされていない

【解説】

ホスト名などのサービスロケーション名とサービス名の両方または片方を変換して、指定したサービスに対応するソケットアドレスとそれに関連する情報を返す。取得した情報をソケット生成時に使用する。

node と service は、NULL、あるいは、ヌル文字で終端する文字列へのポインタである。アプリケーションは、少なくとも片方の引数に NULL 以外の値を設定しなければいけない。

node と service に指定できる有効な名前の形式は、アドレスファミリに依存する。特定のアドレスファミリを指定せず、かつ名前が複数のアドレスファミリで有効な場合、サポートされるすべてのアドレスファミリにおいて名前解決を行い、エラーが発生しなければ 1 つ以上の結果を返す。

node が NULL ではない場合、サービスロケーションを表す文字列またはアドレスを示す文字列を指定する。指定したアドレスファミリが AF_INET または AF_UNSPEC の場合、 node にホスト名、もしくは IPv4 のドット区切りの数字で表現された文字列を指定する。

node が NULL の場合、解決すべきサービスロケーションは呼び出し元のローカルホストである。それ以外の場合、解決すべきサービスロケーションは node で示されるホストである。

service が NULL の場合、指定した node に対するネットワークレベルのアドレスを返す。 service が NULL ではない場合、サービス名を示すヌル文字で終端する文字列を指定する。指定したアドレスファミリが AF_INET または AF_UNSPEC の場合、 service は10進数のポート番号を示す文字列、または、対応するサービス名である。

hints が NULL ではない場合、名前解決処理を制御するオプションを設定し、特定のソケットタイプ、アドレスファミリ、プロトコルを指定して解決結果を制限する。hints の ai_flags, ai_family, ai_socktype, ai_protocol を除くすべてのメンバに 0 または NULL を設定しなければならない。 hints が NULL の場合、ai_flags, ai_socktype, ai_protocol に 0 を設定して、ai_family フィールドに AF_UNSPEC を設定した場合と同じ処理を行う。

ai_family
　　ai_family フィールドにサービスが求めるアドレスファミリを指定する。

　　ai_family フィールドに AF_UNSPEC を指定すると、呼び出し元はいかなるアドレスファミリも結果として受け付ける。ai_family フィールドに AF_UNSPEC を設定した場合、node と service を両方指定していれば両方が、そうでなければ片方が使用できる 1 つ以上のアドレスファミリに対するアドレスをすべて返す。それ以外の場合は、指定したアドレスファミリでのみ利用可能なアドレスを返す。ai_family が AF_UNSPECではなく、かつ ai_protocol が 0 でない場合、指定したアドレスファミリとプロトコルで利用可能なアドレスを返す。

ai_socktype
　　ai_socktype フィールドにサービスが求めるソケットタイプを指定する。
　　指定可能な値は、0 あるいは SOCK_STREAM, SOCK_DGRAM, SOCK_RAW のいずれかである。

　　ai_socktype フィールドに 0 を設定すると、呼び出し元はいかなるソケットタイプも結果として受け付ける。特定のソケットタイプを与えず(例、値が 0)、かつサービス名が複数のソケットタイプで有効である場合、サポートされているすべてのソケットタイプに対してサービス名の解決処理を行い、エラーが発生しなければすべての結果を返す。ソケットタイプの値が 0 ではない場合、結果として得られる情報は指定したソケットタイプに制限される。

ai_protocol
　　ai_protocol フィールドにサービスが求めるプロトコルを指定する。
　　指定可能な値は、IPPROTO_UDP または IPPROTO_TCP である。

　　ai_protocol フィールドに0を設定すると、呼び出し元はいかなるプロトコルも結果として受け付ける。

ai_family フィールドに 0 以外の値を設定し、かつ、ai_protocol にも 0 以外の値を設定した場合、指定されたアドレスファミリとプロトコルに対して有効なアドレスのみを返す。

ai_flags
ai_flags フィールドに 0 、あるいは AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST, AI_NUMERICSERV から 1 つ以上の論理和をとった値を指定する。

AI_CANONNAME
AI_CANONNAME フラグを設定して、かつ node が NULL ではない場合、node に対応する標準の名前を返そうとする。

AI_NUMERICHOST
AI_NUMERICHOST フラグを設定すると、 node に NULL ではない数値表現のアドレス文字列を指定しなければならない。それ以外の場合は EX_AI_NONAME を返す。このフラグを指定することにより名前解決サービス(例、DNS)の利用を回避できる。

AI_NUMERICSERV
AI_NUMERICSERV フラグを設定すると、 service に NULL ではない数値表現のポート番号を表す文字列を指定しなければならない。それ以外の場合は EX_AI_NONAME を返す。このフラグを指定することによりサービス解決(例、NIS+)の利用を回避できる。

AI_PASSIVE
AI_PASSIVE フラグを設定すると、指定したサービスに対して、接続の要求受付ソケットに使用できるアドレス情報を返す。このアドレス情報は名前付け(so_bind())にも使用できる。
さらに node が NULL の場合、ソケットアドレス構造体の IP アドレスの部分に INADDR_ANY を設定する。 node が NULL ではない場合、AI_PASSIVE フラグを無視する。

AI_PASSIVE フラグを設定しない場合、コネクション型プロトコルに対しては so_connect() の呼び出しが、コネクションレス型プロトコルに対しては so_connect(), so_sendto(), so_sendmsg() 呼び出しが可能なソケットを生成できるアドレス情報を返す。node が NULL の場合、ソケットアドレス構造の IP アドレス部分にループバックアドレスを設定する。

buf は結果を格納するバッファ領域へのポインタであり、 bufsz にそのサイズをバイト数で指定する。POSIX仕様の getaddrinfo() は、メモリ領域を動的に確保して結果をそこに格納し、そのポインタを res に設定する。それに対して so_getaddrinfo(), so_getaddrinfo_ms(), so_getaddrinfo_us() は、APIコール内の動的なメモリ割当てを回避するために、 buf で与えた領域に結果を格納する。正常に名前解決を実行できた場合の戻値として、結果のすべてのエントリを格納するために必要なバッファサイズを返す。

結果を格納するために必要なメモリサイズが bufsz よりも大きい場合、格納可能な数のエントリが buf に格納される。buf に 1 つのエントリも格納できない場合は res に NULL が設定される。

また、上記のようにAPIコール内でメモリ領域を動的に確保しないので、so_getaddrinfo() の結果を格納したメモリ領域を解放する必要はない。そのため、POSIX仕様とは異なり、結果のメモリ領域を解放する freeaddrinfo() に相当するAPIコールは存在しない。

timeout, tmout, tmout_u に名前解決処理が完了するまでのタイムアウト時間を指定し、それぞれ so_getaddrinfo(), so_getaddrinfo_ms(), so_getaddrinfo_us() で使用する。タイムアウトするまでの相対時間は、timout には timeout 構造体を使用して、tmout にはミリ秒単位で、tmout_u にはマイクロ秒単位で指定する。

timeout, tmout, tmout_u が 0 秒より大きな時間を示し、かつ、その指定した時間内に名前解決が完了しなかった場合、名前解決処理を中止して EX_TIMEDOUT を返す。この場合、名前解決処理を継続せず、再び同じ名前を解決するときは最初から処理を開始する。 tmout_u, tmout に TMO_FEVR を指定するか、timeout に NULL を設定した場合、結果用のメモリ領域割当てを除いて、POSIX仕様の getaddrinfo() と等しい挙動を示す。

so_getaddrinfo()が正常終了すると、 res に addrinfo 構造体のリンクリストを参照するポインタを格納する。各 addrinfo 構造体は、ソケット生成時に使用するソケットアドレスとそれに関連する情報を保持する。リンクリストは 1 つ以上の addrinfo 構造体から構成される。各構造体の ai_next フィールドは、リンクリストの次の構造体へのポインタである。ただし、リストの最後の構造体の場合、この値は NULL である。リンクリストの各構造体は、 so_socket() 呼び出し時に渡す値と so_connect() 呼び出し時に渡すソケットアドレス、さらに AI_PASSIVE を設定した場合は so_bind() 呼び出し時に渡すソケットアドレスを格納している。ai_family, ai_socktype, と ai_protocol は、so_socket() へ渡す引数として利用可能でき、名前解決結果のアドレスをもつソケットを生成する。ai_addr と ai_addrlen は、AI_PASSIVEに応じて、so_connect()またはso_bind()に渡す引数として利用可能である。

node が NULL ではなく、かつ AI_CANONNAME フラグを設定した場合、結果として得られるリンクリストの先頭の addrinfo 構造体の ai_canonname フィールドは、入力 node に対応する標準的な名前を示すヌル文字で終端する文字列を指す。標準的な名前が利用できない場合、ai_canonname は node と同じ内容の文字列を参照する。結果中の ai_flags フィールドの値は不定である。

so_getaddrinfo()の処理が完了する前に、本APIコールの処理完了待ちをしているタスクに対して so_break() が発行されると、本APIコールは名前解決処理を中止して EX_INTR を返す。この場合、名前解決処理を継続せず、再

び同じ名前を解決するときは最初から処理を開始する。

なお、POSIX 仕様で定義されている gethostbyname, gethostbyaddr にはスレッドセーフではない問題があるほか、本機能で十分代替できるので、T2EX はこれらに相当するAPIコールを提供しない。

【関連項目】

so_bind, so_connect, so_getnameinfo, so_socket

5.5.20 so_getnameinfo, so_getnameinfo_ms, so_getnameinfo_us - アドレスから名前への変換

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_getnameinfo(const struct sockaddr* sa, socklen_t salen, char* host, size_t hostlen, char*
serv, size_t servlen, int flags, struct timeval* timeout);
ER ercd = so_getnameinfo_ms(const struct sockaddr* sa, socklen_t salen, char* host, size_t hostlen,
char* serv, size_t servlen, int flags, TMO tmout);
ER ercd = so_getnameinfo_us(const struct sockaddr* sa, socklen_t salen, char* host, size_t hostlen,
char* serv, size_t servlen, int flags, TMO_U tmout_u);
```

【パラメータ】

const struct sockaddr*	sa	名前へ変換するアドレス
socklen_t	salen	名前へ変換するアドレスのサイズ(バイト数)
char*	host	ホスト名を格納するバッファ
size_t	hostlen	ホスト名を格納するバッファのサイズ(バイト数)
char*	serv	サービス名を格納するバッファ
size_t	servlen	サービス名を格納するバッファのサイズ(バイト数)
int	flags	オプションを指定するフラグ
struct timeval*	timeout	タイムアウト指定 (timeval 形式)
TMO	tmout	タイムアウト指定 (ミリ秒)
TMO_U	tmout_u	タイムアウト指定 (マイクロ秒)

【リターンパラメータ】

ER	ercd	エラーコード
char*	host	ホスト名
char*	serv	サービス名

【エラーコード】

E_OK	正常終了
EX_TIMEDOUT	処理が完了する前にタイムアウトが発生した
EX_INTR	so_break() による中止
EX_INVAL	不正なパラメータ
EX_AI_AGAIN	名前解決が一時的に失敗した(名前解決を再試行すると成功するかもしれない)
EX_AI_BADFLAGS	ai_flags の値が不正
EX_AI_FAIL	名前解決時に回復不可能なエラーが発生した
EX_AI_FAMILY	不正なアドレスファミリ - サポートされていないアドレスファミリ - 指定されたアドレスファミリに対して無効なアドレス長
EX_AI_NONAME	名前解決ができない - NI_NAMEREQD が設定されていてホスト名が見つからない - host と serv の両方が NULL
EX_AI_OVERFLOW	オーバーフロー - host あるいは serv が指すバッファサイズが小さすぎる
EX_AI_SYSTEM	内部エラー - アドレスから文字列への変換が失敗した
EX_AI_SOCKTYPE	ソケットタイプはサポートされていない

【解説】

ソケットアドレスを変換して、ホスト名などのサービスロケーション名とサービス名の両方または片方を返す。

sa は、変換すべきソケットアドレス構造体へのポインタである。

host が NULL ではなく、かつ hostlen が 0 ではない場合、 host に最大でサイズが hostlen のアドレスを格納

できるバッファを指定する。そのバッファにヌル文字で終端する文字列で表現された host の名前を格納する。host が NULL であるか、あるいは hostlen が 0 である場合、host の名前を取得しない。host の名前を見つけられなかった場合、名前を表す文字列の代わりに、sa が指すソケットアドレス構造体に含まれている数値表現のアドレスを返す。

serv が NULL ではなく、かつ servlen が 0 ではない場合、 serv は最大でサイズが servlen のアドレスを格納できるバッファを指定する。そのバッファにヌル文字で終端する文字列で表現された serv の名前を格納する。serv が NULL であるか、あるいは servlen が 0 である場合、serv の名前を取得しない。serv の名前を見つけられなかった場合、名前を表す文字列の代わりに、ポート番号のような数値表現のサービスアドレスを返す。

flags の値に従い、本APIコールのデフォルトの振舞いを変更する。デフォルトでは、指定されたホストに対する完全修飾ドメイン名(FQDN: Fully-Qualified Domain Name)を取得する。 flags には 0 、または以下の値の論理和を指定する。

NI_NOFQDN
: flags に NI_NOFQDN を設定すると、ローカルホストに対しては FQDN の一部分のみを返す。

NI_NUMERICHOST
: flags に NI_NUMERICHOST を設定すると、名前を表す文字列の代わりに、sa が指すソケットアドレス構造体に含まれている数値表現のアドレスを返す。

NI_NAMEREQD
: flags に NI_NAMEREQD を設定すると、host の名前が見つからなかった場合にエラーを返す。

NI_NUMERICSERV
: flags に NI_NUMERICSERV を設定されると、名前を表す文字列の代わりに、ポート番号のような数値表現のサービスアドレスを返す。

NI_DGRAM
: flags に NI_DGRAM を設定すると、サービスがデータグラムサービス(SOCK_DGRAM)であることを示す。デフォルトの振舞いでは、サービスをストリームサービスであるとみなす(SOCK_STREAM)。NI_DGRAM フラグは、ポート番号 512, 514 のサービスのように UDP と TCP で異なるサービスを表す場合に必要とされる。

timeout, tmout, tmout_u に名前解決処理が完了するまでのタイムアウト時間を指定し、それぞれ so_getnameinfo(), so_getnameinfo_ms(), so_getnameinfo_us() で使用する。タイムアウトするまでの相対時間は、timeout には timeout 構造体を使用して、tmout にはミリ秒単位で、tmout_u にはマイクロ秒単位で指定する。

timeout, tmout, tmout_u が 0 秒より大きな時間を示し、かつ、その指定した時間内に名前解決が完了しなかった場合、名前解決処理を中止して EX_TIMEDOUT を返す。この場合、名前解決処理を継続せず、再び同じ名前を解決するときは最初から処理を開始する。 tmout, tmout_u に TMO_FEVR に設定するか、timeout に NULL を設定した場合、結果用のメモリ領域割当てを除いて、POSIX仕様の getnameinfo() と等しい挙動を示す。

so_getnameinfo()の処理が完了する前に、本APIコールの処理完了待ちをしているタスクに対して so_break() が発行されると、本APIコールは名前解決処理を中止して EX_INTR を返す。この場合、名前解決処理を継続せず、再び同じ名前を解決するときは最初から処理を開始する。

なお、POSIX 仕様で定義されている getservbyname, getservbyport にはスレッドセーフではない問題があるほか、本機能で十分代替できるので、T2EX はこれらに相当するAPIコールを提供しない。

5.5.21 so_resctl - 名前解決に関する操作

【C言語インタフェース】

```
#include <t2ex/socket.h>

int len = so_resctl(int cmd, void* buf, size_t bufsz);
```

【パラメータ】

int	cmd	コマンド
void*	buf	データ格納用バッファ
int	bufsz	バッファサイズ

【リターンパラメータ】

int	len	データを格納するために必要なバッファサイズ（バイト数）、またはエラーコード
void*	buf	操作結果

【エラーコード】

EX_INVAL	不正なパラメータ

【解説】

cmd で指定された名前解決に関する操作を行う。

cmd には以下のいずれかのコマンドを指定する。

SO_RES_ADD_TABLE	ホスト名テーブルエントリの追加
SO_RES_DEL_TABLE	ホスト名テーブルエントリの削除
SO_RES_GET_TABLES	ホスト名テーブルの取得
SO_RES_FLUSH_TABLES	ホスト名テーブルのエントリ全削除
SO_RES_ADD_SERVER	名前解決サーバの追加
SO_RES_DEL_SERVER	名前解決サーバの削除
SO_RES_GET_SERVERS	名前解決サーバリストの取得
SO_RES_FLUSH_SERVERS	名前解決サーバーリストの全削除
SO_RES_ADD_DOMAIN	検索ドメインの追加
SO_RES_DEL_DOMAIN	検索ドメインの削除
SO_RES_GET_DOMAINS	検索ドメインリストの取得
SO_RES_FLUSH_DOMAINS	検索ドメインリストの全削除

buf にはデータ格納バッファを指定して、bufsz にはそのサイズ(バイト数)を指定する。buf は、アプリケーションから本関数にデータ渡す場合と、本関数からアプリケーションにデータを返す場合の両方で使用する。アプリケーションから本関数にデータを渡す場合は、buf の先頭アドレスから bufsz バイトだけデータを格納する。本関数からアプリケーションにデータを返す場合は、buf の先頭アドレスから戻値 len バイトまでを使用する。また、buf にNULL を指定すると、データを格納するために必要なバッファサイズを戻値として得ることができる。

本APIコールは、名前を解決する機能は提供しない。名前を解決する機能は so_getaddrinfo() が提供し、本APIコールによる設定内容に基づいた名前解決を行う。

SO_RES_ADD_TABLE
bufの型: const struct hosttable*

buf にはホスト名テーブルに追加する内容を設定する。
bufszにはbufのサイズを指定する。
hosttable 構造体の addr にアドレスを格納している sockaddr 構造体へのポインタを指定して、
host にヌル文字で終端するホスト名へのポインタを設定する。
aliases にはヌル文字で終端するスペース(' ')で区切られたホストの別名へのポインタを設定する。
メンバ aliases には NULL を指定してもよい。

SO_RES_DEL_TABLE
bufの型: const struct hosttable*

buf にはホスト名テーブルから削除する内容を設定する。
bufszにはbufのサイズを指定する。

SO_RES_GET_TABLES
bufの型: struct hosttable*

buf にはホスト名テーブルを格納するためのバッファを設定する。
bufszにはbufのサイズを指定する。
buf には hosttable 構造体の配列形式でホスト名テーブルが格納され、addr, host, aliases の
各メンバに必要な領域も buf から利用される。最後の要素の addr には NULL が設定される。
buf が指すアドレスは hosttable 構造体を格納できるような適切にアラインメントされたメモリ領域でなければならない。
結果を格納するために必要なメモリサイズが bufsz よりも大きい場合、格納可能な数のホスト名テーブルのエントリが buf に格納される。また、配列の最後の要素の addr には NULL が設定されるので、bufsz は最小でも 1 つのポインタを格納できるサイズが必要である。bufsz が小さくて 1 つのポインタも格納できない場合はエラー EX_INVAL を返す。

```
/* 擬似コード */
union {
        struct hosttable top;
        UB c[256];
} buf;
struct hosttable *res;
int len, i;

len = so_resctl(SO_RES_GET_TABLES, &buf.top, sizeof(buf));
res = &buf.top;
for( i=0; res[i].addr != NULL; i++ ) {
```

```
                /*
                 * res に対する処理
                 *   struct sockaddr *addr = res[i].addr;
                 *   char *host            = res[i].host;
                 *   char *aliases         = res[i].aliases;
                 */
        }
```

SO_RES_FLUSH_TABLES
bufの型: void*

buf には NULL を指定し、bufsz には 0 を指定する。
ホスト名テーブルのエントリを全て削除する。

SO_RES_ADD_SERVER
bufの型: const struct sockaddr*

buf には追加する名前解決サーバのアドレスを設定する。
bufsz には buf のサイズを指定する。
最大で MAXNS 個の名前解決サーバを設定できる。
複数の名前解決サーバを設定した場合、追加順に名前解決サーバへクエリを送信する。

名前解決のアルゴリズム
・最初の名前解決サーバにクエリを送信して、タイムアウトした場合、次の名前解決サーバに対して
  クエリを送信する。これをすべての名前解決サーバに対して行う。
・上記のすべての名前解決サーバに対する問い合わせを、最大試行回数に到達するまで繰り返す。

```
#define MAXNS   (3)     /* 登録できる名前解決サーバの最大個数 */
```

SO_RES_DEL_SERVER
bufの型: const struct sockaddr*

buf には削除する名前解決サーバのアドレスを設定する。
bufsz には buf のサイズを指定する。

SO_RES_GET_SERVERS
bufの型: struct sockaddr**

buf には名前解決サーバリストを格納するためのバッファを設定する。
bufsz には buf のサイズを指定する。
buf には sockaddr 構造体へのポインタの配列形式で名前解決サーバリストが格納され、各 sockaddr 構造体を格納するために必要な領域も buf から利用される。各 sockaddr 構造体のサイズは、
sockaddr構造体のメンバsa_len が示す。ポインタの配列の最後の要素には NULL が設定される。
buf が指すアドレスは sockaddr 構造体へのポインタを格納できるような適切にアラインメントされたメモリ領域でなければならない。
結果を格納するために必要なメモリサイズが bufsz よりも大きい場合、格納可能な数の名前解決サーバのアドレスが buf に格納される。また、配列の最後の要素には NULL が設定されるので、bufsz は最小でも 1 つのポインタを格納できるサイズが必要である。bufsz が小さくて 1 つのポインタも格納できない場合はエラー EX_INVAL を返す。

```
/* 擬似コード */
union {
        struct sockaddr *top;
        UB c[256];
} buf;
struct sockaddr **res;
int len, i;

len = so_resctl(SO_RES_GET_SERVERS, &buf.top, sizeof(buf));
res = &buf.top;
for( i=0; res[i] != NULL; i++ ) {
        /*
         * res に対する処理
         *   struct sockaddr *addr = res[i];
         */
}
```

SO_RES_FLUSH_SERVERS
bufの型: void*

buf には NULL を指定し、bufsz には 0 を指定する。
名前解決サーバーリストを全て削除する。

SO_RES_ADD_DOMAIN
	bufの型: const char*

	buf には追加する検索ドメインを設定する。
	bufsz には buf のサイズを指定する。
	buf はヌル文字で終端する文字列へのポインタである。検索ドメインに属するホストに対して、
	ドメイン部を省略した短縮名を使用して名前を解決できる。登録可能なドメインは MAXDNSRCH 個まで
	である。複数の探索ドメインが登録されている場合、名前解決の際に一致する名前がみつかるまで
	探索ドメインの各要素を登録順に試す。通常、探索ドメインはローカルドメイン名を最初に設定する。
	登録したドメインのサーバがローカルにない場合、各サーバへ問い合わせるために大量の
	ネットワークトラフィックを生じさせる可能性がある。

	検索ドメインとして 1 つのドメインのみを設定している場合、LOCALDOMAINPARTS レベルの階層まで
	最大 MAXDFLSRCH 個に対して一致する名前検索を行う。例えば、"www.xxx.yyy.zzz" というドメイン
	のみを設定している場合、"www.xxx.yyy.zzz"、"xxx.yyy.zzz"、"yyy.zzz" に対して一致する
	名前検索を行う。

```
	#define MAXDNSRCH		6	/* 検索ドメインに登録可能な最大数 */
	#define MAXDFLSRCH		3	/* ローカルドメインを展開する最大数 */
	#define LOCALDOMAINPARTS	2	/* ローカルドメイン名として扱う最小の階層レベル */
```

SO_RES_DEL_DOMAIN
	bufの型: const char*

	buf には削除する検索ドメインを設定する。
	bufsz には buf のサイズを指定する。

SO_RES_GET_DOMAINS
	bufの型: char**

	buf には検索ドメインリストを格納するためのバッファを設定する。
	bufsz には buf のサイズを指定する。
	buf にはヌル文字で終端する文字列へのポインタの配列形式でドメイン名が格納され、各ドメイン名を格納するために必要な領域も buf から利用される。ポインタの配列の最後の要素には NULL が設定される。
	buf が指すアドレスはヌル文字で終端する文字列へのポインタを格納できるような適切にアラインメントされたメモリ領域でなければならない。
	結果を格納するために必要なメモリサイズが bufsz よりも大きい場合、格納可能な数のドメイン名が buf に格納される。また、配列の最後の要素には NULL が設定されるので、bufsz は最小でも 1 つのポインタを格納できるサイズが必要である。bufsz が小さくて 1 つのポインタも格納できない場合はエラー EX_INVAL を返す。

```
	/* 擬似コード */
	union {
		char *top;
		UB c[256];
	} buf;
	char **res;
	int len, i;

	len = so_resctl(SO_RES_GET_SERVERS, &buf.top, sizeof(buf));
	res = &buf.top;
	for( i=0; res[i] != NULL; i++ ) {
		/*
		 * res に対する処理
		 *   char *domain = res[i];
		 */
	}
```

SO_RES_FLUSH_DOMAINS
	bufの型: void*

	buf には NULL を指定し、bufsz には 0 を指定する。
	検索ドメインリストを全て削除する。

【関連項目】

so_getnameinfo

5.5.22 so_rtlist - ルーティングテーブルのエントリ取得

【C言語インタフェース】

```
#include <t2ex/socket.h>

int len = so_rtlist(int af, int cmd, int flags, void* buf, size_t bufsz);
```

【パラメータ】

int	af	アドレスファミリ
int	cmd	コマンド
int	flags	フラグ
void*	buf	ルーティングテーブルを格納するバッファ
size_t	bufsz	バッファサイズ(バイト長)

【リターンパラメータ】

int	len	ルーティングテーブルを格納するために必要なバッファサイズ（バイト数）、 またはエラーコード
void*	buf	ルーティングテーブル

【エラーコード】

EX_AFNOSUPPORT	指定されたアドレスファミリは実装されていない
EX_INVAL	不正なパラメータ

【解説】

引数で指定された条件を満たすルーティングテーブルの要素を取得する。

af にはアドレスファミリを指定し、この引数で指定されたアドレスファミリに対応する経路を取得する。また、af に AF_UNSPEC を指定すると任意のアドレスファミリに対応する経路を取得する。

cmd には以下のいずれかの値を指定する。

NET_RT_DUMP	指定したアドレスファミリに対応する経路をすべて取得する。
NET_RT_FLAGS	指定したアドレスファミリに対して指定したフラグが立っているすべての経路を取得する。
NET_RT_IFLIST	ネットワークインタフェース毎に指定したアドレスファミリに対応する経路をすべて取得する。

flags には以下のフラグの論理和を指定する。flags に指定したフラグのいずれかのビットが立っている経路を取得する。flags は cmd に NET_RT_FLAGS を設定した場合のみ有効であり、それ以外場合は無効である。

RTF_UP	経路が利用可能である。
RTF_GATEWAY	宛先はゲートウェイである。
RTF_HOST	宛先はホストである。
RTF_REJECT	宛先へ到達不可能である。
RTF_DYNAMIC	ICMPリダイレクトにより経路が生成された。
RTF_MODIFIED	ICMPリダイレクトにより経路が変更された。
RTF_CLONING	新しい経路を複製して生成する。
RTF_LLINFO	有効なデータリンク層情報がある。
RTF_STATIC	経路が手動で追加された。
RTF_BLACKHOLE	宛先へのパケットを破棄する。
RTF_CLONED	複製して生成された経路である。
RTF_PROTO2	プロトコル特有のフラグ
RTF_PROTO1	プロトコル特有のフラグ

buf に経路情報を格納するバッファへのポインタを指定し、bufsz には buf が指すバッファサイズ(バイト長)を指定する。buf に NULL を指定した場合、与えられた引数で取得されるルーティングテーブルを格納するために必要なバッファサイズを返す。

buf にはルーティングメッセージが可能な数だけ格納される。最後のルーティングメッセージのヘッダの rtm_msglen には 0 が設定される。そのため、bufsz は最小でも 1 つの rtm_msglen を格納できるサイズが必要である。bufsz が小さくて 1 つの rtm_msglen も格納できない場合はエラー EX_INVAL を返す。

取得された経路を以下のようなコードで参照することができる。

```
/* 擬似コード */
size_t                  needed;
struct rt_msghdr        *rtm;
char                    *buf;

needed = so_rtlist(AF_UNSPEC, NET_RT_DUMP, 0, NULL, 0);
buf = malloc(needed);
needed = so_rtlist(AF_UNSPEC, NET_RT_DUMP, 0, buf, needed);
for( rtm = (struct rt_msghdr *)buf;
```

```
            rtm->rtm_msglen != 0;
            rtm = (struct rt_msghdr *)((void*)rtm + rtm->rtm_msglen) ) {
               /*
                * rtmへの操作
                *  (rtm->rtm_typeに従い、rtmを適切なヘッダにキャストする。
                *   参照: 5.6.1 ルーティングメッセージ)
                */
        }
        free(buf);
```

5.5.23 so_ifattach - デバイスドライバの接続

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_ifattach(const char* devnm);
```

【パラメータ】

const char*	devnm	デバイス名

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
EX_BUSY	デバイスドライバが使用中
EX_INVAL	不正なパラメータ

【解説】

devnm で示されたT-Engine標準デバイスドライバ仕様準拠のデバイスドライバをネットワーク通信マネージャに接続する。devnm には、T-Engine標準デバイスドライバ仕様で規定されているデバイス名(例、Neta, Netbなど)を指定する。

本APIコールの正常終了後は、devnm に対してアドレス設定、活性化などを行うことができる。

【関連項目】

so_ifdetach, so_ioctl, so_socket

5.5.24 so_ifdetach - デバイスドライバの切り離し

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_ifdetach(const char* devnm);
```

【パラメータ】

const char*	devnm	デバイス名

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
EX_NOENT	デバイスドライバが未接続
EX_BUSY	デバイスドライバが使用中
EX_INVAL	不正なパラメータ

【解説】

devnm で示されたT-Engine標準デバイスドライバ仕様準拠のデバイスドライバをネットワーク通信マネージャから切り離す。devnm には、T-Engine標準デバイスドライバ仕様で規定されているデバイス名(例、Neta, Netbなど)を指定する。

【関連項目】

so_ifattach

5.5.25 so_getifaddrs - インタフェースのアドレス情報の取得

【C言語インタフェース】

```
#include <t2ex/socket.h>

int len = so_getifaddrs(struct ifaddrs** ifap, void* buf, size_t bufsz);
```

【パラメータ】

struct ifaddrs**	ifap	インタフェースのアドレス情報のリンクリストの先頭へのポインタ
void*	buf	インタフェースのアドレス情報を格納するバッファ
size_t	bufsz	インタフェースのアドレス情報格納用バッファのサイズ(バイト数)

【リターンパラメータ】

int	len	インタフェースのアドレス情報を格納するために必要なバッファサイズ（バイト数）、 またはエラーコード
struct ifaddrs**	ifap	インタフェースのアドレス情報のリンクリストの先頭へのポインタ
void*	buf	インタフェースのアドレス情報のリンクリスト

【エラーコード】

EX_INVAL	不正なパラメータ

【解説】

登録されているネットワークインタフェースのアドレス情報をリンクリスト形式で取得する。

ifap は ifaddrs 構造体へのポインタの参照である。

```
    struct ifaddrs   *ifa_next;         /* Pointer to next struct */
    char             *ifa_name;         /* Interface name */
    unsigned int      ifa_flags;        /* Interface flags */
    struct sockaddr  *ifa_addr;         /* Interface address */
    struct sockaddr  *ifa_netmask;      /* Interface netmask */
    struct sockaddr  *ifa_broadaddr;    /* Interface broadcast address */
    struct sockaddr  *ifa_dstaddr;      /* P2P interface destination */
    void             *ifa_data;         /* Address specific data */
```

ifa_next フィールドはリンクリスト上の次の要素へのポインタである。リンクリストの最後の要素の場合、ifa_next フィールドは NULL である。

ifa_name フィールドにはインタフェースの名前が格納される。

ifa_flags フィールドには以下のフラグの論理和が格納される。

下記の説明中の R-、RW は、それぞれ参照のみできることと参照・変更の両方ができることを意味する。

IFF_UP	RW	インタフェースが動作中である。
IFF_BROADCAST	R-	ブロードキャストアドレスが有効である。
IFF_DEBUG	RW	デバッグモードである。
IFF_LOOPBACK	RW	ループバックである。
IFF_POINTOPOINT	R-	point-to-point リンクである。
IFF_NOTRAILERS	RW	トレーラを使用しない。
IFF_RUNNING	R-	資源が割り当て済である。
IFF_NOARP	RW	ネットワークのアドレス解決が無効である。
IFF_PROMISC	R-	すべてのパケットを受信する。
IFF_ALLMULTI	R-	すべてのマルチキャストパケットを受信する。
IFF_OACTIVE	R-	送信中である。
IFF_SIMPLEX	R-	単方向通信である。
IFF_LINK0	RW	リンクレイヤ用の制御フラグ

IFF_LINK1	RW	リンクレイヤ用の制御フラグ
IFF_LINK2	RW	リンクレイヤ用の制御フラグ
IFF_MULTICAST	R-	マルチキャストをサポートしている。

ifa_addr フィールドにはインタフェースのアドレスまたはデータリンク層レベルのアドレスが格納される。いずれのアドレスも存在しない場合、NULL が設定される。

ifa_network フィールドには ifa_addr に関連付けられているネットマスクが格納される。ネットマスクが存在しない場合、NULL が設定される。

P2P インタフェースではない場合、ifa_broadaddr フィールドには ifa_addr に関連付けられているブロードキャストアドレスが格納される。そうなければ、NULL が設定される。

P2P インタフェースの場合、ifa_dstaddr フィールドには宛先アドレスが格納される。さもなければ、 NULL が設定される。

ifa_data フィールドにはアドレスファミリ特有の情報が格納される。AF_LINK の場合、インタフェース情報や統計情報をもつ if_data 構造体のデータが格納される。それ以外のアドレスファミリの場合、NULL が設定される。

buf はネットワークインタフェースのアドレス情報を格納するバッファ領域へのポインタであり、 bufsz にそのサイズをバイト数で指定する。正常終了時の戻値はアドレス情報を格納するために必要なバッファサイズを表す。

アドレス情報を格納するために必要なメモリサイズが bufsz よりも大きい場合、格納可能な数のエントリが buf に格納される。buf に 1 つのエントリも格納できない場合は ifap に NULL が設定される。

【関連項目】

so_ifattach, so_ifdetach, so_ioctl

5.5.26 so_ifindextoname - インタフェースインデックスをインタフェース名に変換

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_ifindextoname(unsigned int ifindex, char* ifname);
```

【パラメータ】

unsigned int	ifindex	インタフェースインデックス
char*	ifname	インタフェース名を格納するバッファ

【リターンパラメータ】

ER	ercd	エラーコード
char*	ifname	インタフェース名

【エラーコード】

E_OK	正常終了
EX_NXIO	インタフェースが存在しない

【解説】

インタフェースインデックスをインタフェース名に変換する。

ifindex にはインタフェースインデックスを指定する。このインタフェースインデックスは、so_getifaddrs() やルーティングメッセージなどで取得される整数値である。

ifname には少なくとも IF_NAMESIZE バイトのバッファを指定する。このAPIコールが正常終了すると、ifname にヌル文字で終端するインタフェース名が格納される。

```
#define IF_NAMESIZE     16      /* インタフェース名の最大サイズ(バイト長、ヌル文字を含む) */
```

【関連項目】

so_ifnametoindex

5.5.27 so_ifnametoindex - インタフェース名をインタフェースインデックスに変換

【C言語インタフェース】

```
#include <t2ex/socket.h>

int ifindex = so_ifnametoindex(const char* ifname);
```

【パラメータ】

const char*	ifname	インタフェース名

【リターンパラメータ】

int	ifindex	インタフェースインデックス、 またはエラーコード

【エラーコード】

EX_NXIO	インタフェースが存在しない

【解説】

インタフェース名をインタフェースインデックスに変換する。

ifname にはヌル文字で終端するインタフェース名を指定する。

【関連項目】

so_ifindextoname

5.5.28 so_ioctl - デバイスの制御

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_ioctl(int sd, int request, ... /* arg */);
```

【パラメータ】

int	sd	ソケットディスクリプタ
int	request	要求

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
EX_BADF	sd が不正
EX_INVAL	要求または要求への引数が不正
EX_FAULT	要求への引数が有効なアドレス空間にない

【解説】

デバイスに関する種々の制御機能を実行する。

sd にデバイスを参照するオープンされたソケットディスクリプタを指定する。

request に実行する制御機能を指定する。request の値は対象となるデバイスに依存する。

arg に対象となるデバイスにおいて、request の実行に必要な付加的情報を指定する。arg の型は、request の種類に依存するが、通常、int 型、またはデバイス固有のデータ構造へのポインタである。

request に指定可能な値とその arg に関する説明は以下の通りである。

FIONBIO	const int* ディスクリプタに対するI/O動作のブロッキング/ノンブロッキングモードを、引数の int* が指す値で設定する。 *arg == 0 の場合は、ブロッキングモードとなる(O_NONBLOCK 状態フラグを解除する)。

*arg != 0 以外の場合は、ノンブロッキングモードとなる(O_NONBLOCK 状態フラグを設定する)。

FIONREAD int*
SIOCINQ int*
直ちに読み込める準備ができているバイト数を、引数の int* が指す領域に返す。

FIONWRITE int*
SIOCOUTQ int*
ディスクリプタの送信キューに入っているバイト数を、引数の int* が指す領域に返す。
それらのバイトは、ディスクリプタに書き込まれたデータで、処理をまっている状態にある。
それらがどのように処理されるかは、デバイスに依存する。

FIONSPACE int*
ディスクリプタの送信キューの空き容量を、引数の int* が指す領域に返す。
この値は、送信キューのサイズからキューに保持されているデータのサイズを引いたものである。

SIOCATMARK int*
帯域外データを受信しているかどうか調べて、引数の int* が指す領域に結果を返す。
この値が 1 の場合、ソケットに帯域外データマークが付けられている。この場合、
帯域外データは so_recv() に MSG_OOB フラグを指定して読み込むことができる。
この値が 0 の場合、ソケットに帯域外データマークが付けられていない。

SIOCSIFADDR const struct ifreq*
指定されたアドレスをネットワークインタフェースに設定する。

SIOCAIFADDR const struct ifaliasreq*
指定されたアドレスをネットワークインタフェースに設定または追加する。
あるいは、既に設定されているアドレスを指定した場合は、そのアドレスに付随する情報を
更新する。このコマンドは一度にホストアドレス、宛先アドレス、ブロードキャストアドレス、
ネットマスクを設定することができる。

SIOCDIFADDR const struct ifaliasreq*
指定されたアドレスをネットワークインタフェースから削除する。

SIOCGIFADDR struct ifreq*
指定されたネットワークインタフェースのアドレスを取得する。

SIOCSIFNETMASK const struct ifreq*
指定されたネットワークインタフェースにネットマスクを設定する。

SIOCGIFNETMASK struct ifreq*
指定されたネットワークインタフェースのネットマスクを取得する。

SIOCSIFFLAGS const struct ifreq*
指定されたネットワークインタフェースにフラグを設定する。

SIOCGIFFLAGS struct ifreq*
指定されたネットワークインタフェースのフラグを取得する。

SIOCSIFBRDADDR const struct ifreq*
指定されたネットワークインタフェースにブロードキャストアドレスを設定する。
ネットワークインタフェースにIFF_BROADCASTフラグが設定されていない場合、
エラーEX_INVALを返す。

SIOCGIFBRDADDR struct ifreq*
指定されたネットワークインタフェースのブロードキャストアドレスを取得する。
ネットワークインタフェースにIFF_BROADCASTフラグが設定されていない場合、
エラーEX_INVALを返す。

SIOCSIFDSTADDR const struct ifreq*
指定されたネットワークインタフェースの宛先アドレスを設定する。
ネットワークインタフェースにIFF_POINTOPOINTフラグが設定されていない場合、
エラーEX_INVALを返す。

SIOCGIFDSTADDR struct ifreq*
指定されたネットワークインタフェースの宛先アドレスを取得する。
ネットワークインタフェースにIFF_POINTOPOINTフラグが設定されていない場合、
エラーEX_INVALを返す。

【関連項目】

so_read, so_recv, so_sockatmark, so_write

5.5.29 so_fcntl - ソケットディスクリプタの操作

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_fcntl(int sd, int cmd, ... /* arg */);
```

【パラメータ】

int	sd	ソケットディスクリプタ
int	cmd	操作コマンド
	arg	コマンドに応じて必要な引数(可変個)

【リターンパラメータ】

ER	ercd	コマンドに応じた0以上の結果 またはエラーコード

E_OK	正常終了
EX_BADF	sd が不正
EX_INVAL	cmd の値が不正

【解説】

ソケットディスクリプタ sd に対して、cmd に応じた以下の操作を行う。

F_GETFL
　　ソケットディスクリプタ sd の状態フラグを取得する。戻値として取得された状態フラグを返す。

F_SETFL
　　ソケットディスクリプタ sd の状態フラグを、第3引数 arg の対応するビットで設定する。
　　第3引数 arg は int 型とする。

F_GETFL, F_SETFL コマンドで利用可能な状態フラグは O_NONBLOCK である。

O_NONBLOCK
　　ノンブロッキングモードを示すフラグである。このフラグが設定されている場合、
　　そのソケットディスクリプタはノンブロッキングモードである。

【関連項目】

so_accept, so_close, so_connect

5.5.30 so_sockatmark - 帯域外データマークが付けられているか調べる

【C言語インタフェース】

```
#include <t2ex/socket.h>

int mark = so_sockatmark(int sd);
```

【パラメータ】

int	sd	ソケットディスクリプタ

【リターンパラメータ】

int	mark	帯域外データマークの有無を示す値、 またはエラーコード

【エラーコード】

EX_BADF	sd が不正
EX_INVAL	不正なパラメータ

【解説】

sd で指定したソケットに帯域外データマークが存在するか調べる。

ストリームに帯域外データマークを付けることにより、帯域外データをサポートするプロトコルがある。指定したソケットのプロトコルがこのようなプロトコルのときにマークの前の全データが読み込まれて帯域外データマークが受信キューの先頭に位置する場合、so_sockatmark() は 1 を返す。マークが受信キューの中に存在しない場合、またはマークの前に通常データが存在する場合は 0 を返す。so_sockatmark() はストリームからマークを取り除かない。

【関連項目】

so_ioctl, so_recvmsg

5.5.31 so_shutdown - 全二重接続の一部を閉じる

【C言語インタフェース】

```
#include <t2ex/socket.h>

ER ercd = so_shutdown(int sd, int how);
```

【パラメータ】

int	sd	ソケットディスクリプタ
int	how	シャットダウンの仕方

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
EX_BADF	sd が不正
EX_INVAL	how が不正
EX_NOTCONN	指定されたソケットは接続されていない

【解説】

ソケットディスクリプタが示すソケットに対するすべてまたは一部の全二重接続を閉じる。

sd にソケットディスクリプタを指定する。

how にシャットダウンの仕方を指定する。次の値のいずれかを指定する。

SHUT_RD
: 以降の受信操作を無効にする。

SHUT_WR
: 以降の送信操作を無効にする。

SHUT_RDWR
: 以降の送信操作と受信操作を無効にする。

so_shutdown() は正常終了後、how の値に従いソケットに対する送信操作と受信操作の両方または片方を禁止する。

5.5.32 so_break - ソケット操作の中止

【C言語インタフェース】

```
#include <t2ex/socket.h>

int ntsk = so_break(ID tskid);
```

【パラメータ】

ID	tskid	待ちを解除するタスクID

【リターンパラメータ】

int	ntsk	待ちを解除したタスクの数、 またはエラーコード

【エラーコード】

E_ID	タスクIDが不正(負または TMaxTskId を超えている)
E_NOEXS	タスクIDのタスクが存在しない

【解説】

ネットワーク通信マネージャの呼び出しに伴うタスク tskid で指定したタスクの待ち状態を解除する。

tskid が TSK_ALL(= -1) の場合、すべてのタスクに対して、ネットワーク通信機能呼び出しの待ちを解除する。

指定したタスクのソケット操作による待ちを即座に解除し、割り込まれたAPIコールは待ち状態にあった場合 EX_INTR を返す、あるいは、so_break() が発行された時点までの処理結果を返す。

so_read(), so_recv(), so_recvmsg(), so_recvfrom() による待ち状態のタスクに対して待ちを解除した場合、これらのAPIコールは何もデータを読込んでいないときはエラー EX_INTR を返す。1 バイト以上の読み込んだデータがあるときは読み込んだデータ数を返す。

so_write(), so_send(), so_sendmsg(), so_sendto() による待ち状態のタスクに対して待ちを解除した場合、これらのAPIコールは何もデータを書き込んでいないときはエラー EX_INTR を返す。1 バイト以上の書き込んだデータがあるときは書き込んだデータ数を返す。

so_accept(), so_connect(), so_select(), so_select_ms(), so_select_us() による待ち状態のタスクに対して待ちを解除した場合、これらのAPIコールはエラー EX_INTR を返す。so_break() により中止されたこれらのAPIコールの処理は、so_select(), so_select_ms(), so_select_us() を用いることにより、処理の完了を知ることができる。

so_getaddrinfo(), so_getaddrinfo_ms(), so_getaddrinfo_us(), so_getnameinfo(), so_getnameinfo_ms(), so_getnameinfo_us() による待ち状態のタスクに対して待ちを解除した場合、これらのAPIコールはエラー EX_INTR を返す。so_break()により中止された処理は再開することができない。

so_close() による待ち状態のタスクに対して待ちを解除した場合、本APIコールは正常終了する。指定したソケットは遅延して閉じられる。

so_break() は、正常終了した場合、待ちを解除したタスクの数を返す。対象となるタスクがネットワーク通信機能の呼び出しを実行していない場合、本APIコールは何も行わず即座に終了して戻値 E_OK を返す。

so_break() による待ち状態の解除要求はキューイングされない。

【関連項目】

so_accept, so_close, so_connect, so_getaddrinfo, so_getaddrinfo_ms, so_getaddrinfo_us, so_getnameinfo, so_getnameinfo_ms, so_getnameinfo_us, so_read, so_recv, so_recvfrom, so_recvmsg, so_select, so_send, so_sendmsg, so_sendto, so_write

5.6 ルーティングソケットに対する操作

アプリケーションは、ルーティングソケットに対してルーティングメッセージを用いてルーティングテーブルの操作を行う。また、ルーティングソケットでルーティングメッセージを受信することにより、ネットワーク通信マネージャからのルーティングテーブルに関連するイベント通知を取得することができる。

ルーティングソケットは以下の引数をAPIコール so_socket() に渡して生成する。

```
s = socket( PF_ROUTE, SOCK_RAW, protocol );
```

protocol に AF_UNSPEC を指定すると、すべてのルーティングメッセージを送受信することができる。また、protocol に特定のアドレスファミリを指定すると、指定されたアドレスファミリに関連するルーティングメッセージを送受信することができる。例えば、protocol に AF_INET を指定するとIP関連のルーティングメッセージのみを送受信する。

ルーティングソケットを通してネットワーク通信マネージャに送信されたメッセージは、すべてのルーティングソケットへ送信される。このため、ルーティングソケットを通してメッセージを送信すると、そのソケットの受信キューに送信したメッセージのコピーが挿入され、受信操作により自身が送信したメッセージを読み込むことになる。そこで、so_setsockopt() を用いて SO_USELOOPBACK をオフにすることにより、自身が送信したメッセージが受信キューに挿入されることを回避できる。あるいは、so_shutdown() を用いて以降のルーティングメッセージがソケットの受信キューに挿入されることを回避できる。

5.6.1 ルーティングメッセージ

ルーティングメッセージは、経路の追加、削除、取得などのルーティングテーブル操作と、経路の追加通知、削除通知、経路探索失敗などのネットワーク通信マネージャからのルーティングテーブルに関連するイベント通知に用いられる。

ルーティングメッセージは、ヘッダとそれに続くいくつかの sockaddr 構造体のアドレスから構成される。ルーティングメッセージには以下のメッセージタイプがある。

RTM_ADD	RW	経路の追加
RTM_DELETE	RW	経路の削除
RTM_CHANGE	RW	経路の変更
RTM_GET	RW	経路の取得
RTM_LOSING	R-	宛先へ到達失敗
RTM_REDIRECT	R-	ICMPリダイレクトの通知
RTM_MISS	R-	経路の探索失敗
RTM_LOCK	RW	指定されたルーティングメトリックのロック
RTM_NEWADDR	R-	インタフェースにアドレスの追加
RTM_DELADDR	R-	インタフェースからアドレスの削除
RTM_IFINFO	R-	インタフェース/リンクの状態変化
RTM_IFANNOUNCE	R-	インタフェースの着脱通知

RW: 設定/参照可能
R-: 参照のみ可能

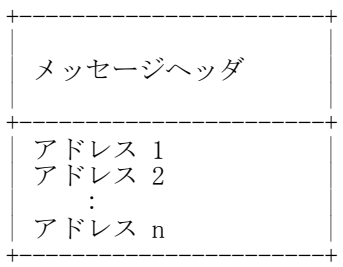


図:ルーティングメッセージの構成

5.6.1.1 メッセージヘッダ部

ルーティングメッセージのヘッダの構造はメッセージタイプに依存する。

RTM_ADD	rt_msghdr構造体
RTM_DELETE	rt_msghdr構造体
RTM_CHANGE	rt_msghdr構造体
RTM_GET	rt_msghdr構造体
RTM_LOSING	rt_msghdr構造体
RTM_REDIRECT	rt_msghdr構造体
RTM_MISS	rt_msghdr構造体
RTM_LOCK	rt_msghdr構造体
RTM_NEWADDR	ifa_msghdr構造体
RTM_DELADDR	ifa_msgdhr構造体
RTM_IFINFO	if_msghdr構造体
RTM_IFANNOUNCE	if_announcemsghdr構造体

5.6.1.2 アドレス部

ルーティングメッセージのヘッダ部の直後には、ヘッダで指定されたアドレスが格納される。ヘッダ毎にメッセージに含まれるアドレスを示す構造体のメンバ名は異なり、それぞれrtm_addrs, ifm_addrs, ifam_addrsである。これらは以下のフラグの論理和である。if_announcemsghdr構造体にこのメンバは存在しない。

RTA_DST	宛先アドレス
RTA_GATEWAY	ゲートウェイアドレス
RTA_NETMASK	ネットマスク
RTA_GENMASK	複製するときのマスク
RTA_IFP	ネットワークインタフェースのデータリンク層アドレス
RTA_IFA	ネットワークインタフェースのアドレス
RTA_AUTHOR	ICMPリダイレクトを送信したノードのアドレス
RTA_BRD	ブロードキャストアドレス

アドレス部には、上記フラグの値の小さい順に格納される。例えば、RTA_DST,RTA_GATEWAY, RTA_NETMASK の論理和の場合、ヘッダの直後に宛先アドレス、ゲートウェイアドレス、ネットマスクの順で格納される。

また、RTA_IFP 以外に対応するアドレスは、アドレスはアドレスファミリに従った sockaddr 構造体の形式でアドレスが格納される。RTA_IFP に対応するアドレスは AF_LINK 用の sockaddr_dl 構造体である。各アドレスのサイ

ズは sockaddr 構造体の sa_len で示される。

5.6.2 ルーティングメッセージの詳細

rt_msghdr構造体, ifa_msgdhr構造体, if_msghdr構造体, if_announcemsghdr構造体の各メンバは以下のような意味をもつ。

rtm_msglen, ifm_msglen, ifam_msglen, ifan_msglen は、メッセージヘッダとそれに続くアドレスを合わせたメッセージ全体のサイズを示す。

rtm_version, ifm_version, ifam_version, ifan_version は、バイナリ互換性の確認に使用される。これらの値は RTM_VERSION である。

```
#define RTM_VERSION     3       /* バージョン */
```

rtm_type, ifm_type, ifam_type, ifan_type は、メッセージタイプを示す。rtm_index, ifm_index, ifam_index, ifan_index は、インタフェース構造体のインデックスを示す。

rtm_flags, ifm_flags, ifam_flags は、ルーティングメッセージのフラグを示す。これらの値はフラグの論理和である。フラグの設定は RTM_ADD メッセージでのみ可能である。

RTF_UP	R-	経路が利用可能である。
RTF_GATEWAY	RW	宛先はゲートウェイである。
RTF_HOST	RW	宛先はホストである。
RTF_REJECT	RW	宛先へ到達不可能である。
RTF_DYNAMIC	R-	ICMPリダイレクトにより経路が生成された。
RTF_MODIFIED	R-	ICMPリダイレクトにより経路が変更された。
RTF_DONE	R-	ネットワーク通信マネージャからの完了通知
RTF_CLONING	RW	新しい経路を複製して生成する。
RTF_LLINFO	R-	有効なデータリンク層情報がある。
RTF_STATIC	RW	経路が手動で追加された。
RTF_BLACKHOLE	RW	宛先へのパケットを破棄する。
RTF_CLONED	RW	複製して生成された経路である。
RTF_PROTO2	RW	プロトコル特有のフラグ
RTF_PROTO1	RW	プロトコル特有のフラグ

RW: 設定/参照可能
R-: 参照のみ可能

RTF_UP
RTF_UP は経路が利用可能であることを示す。これはネットワーク通信マネージャが設定するフラグである。
書込み時にこのフラグを設定しても無視される。

RTF_GATEWAY
RTF_GATEWAY は、宛先がゲートウェイであることを示す。RTF_GATEWAY が設定されていない場合、宛先に直接到達する。アプリケーションから設定可能なフラグである。

RTF_HOST
RTF_HOST は、宛先がホストのアドレスであることを示す。RTF_HOST が設定されていない場合、宛先はネットワークのアドレスである。アプリケーションから設定可能なフラグである。

RTF_REJECT
RTF_REJECT は、経路で示される宛先へ到達不可能であることを示す。
アプリケーションから設定可能なフラグであるが、主にARPが使用することが想定されている。
宛先がARPリクエストに応答しない場合、宛先へのパケットを破棄するようにこのフラグが設定される。
送信する宛先がRTF_REJECT を設定した経路にマッチした場合、so_send() などの送信APIコールは EX_HOSTUNREACH を返す。

RTF_DYNAMIC
RTF_DYNAMIC は、ICMPリダイレクトにより経路が追加されたことを示す。
これはネットワーク通信マネージャが設定するフラグである。書込み時にこのフラグを設定しても無視される。

RTF_MODIFIED
RTF_MODIFIED は、ICMPリダイレクトにより既存の経路のゲートウェイが変更されたことを示す。
これはネットワーク通信マネージャが設定するフラグである。書込み時にこのフラグを設定しても無視される。

RTF_DONE
RTF_DONE は、アプリケーションからのルーティングメッセージの処理が完了したことを示す。
これはネットワーク通信マネージャが設定するフラグである。書込み時にこのフラグを設定しても無視される。

また、このフラグはルーティングメッセージ中でのみ使用され、経路情報として保存されることはない。

RTF_CLONING
RTF_CLONING は、宛先への経路を新しく複製して生成することを示す。
例えば、ネットワークのアドレスを宛先にもつ経路に RTF_CLONING を設定した場合、
この経路にマッチしたホストアドレスへの経路を新たに複製してルーティングテーブルへ追加する。
生成された経路には RTF_CLONED が設定される。RTF_CLONING が設定されている経路を削除する場合、
その経路から複製して生成された RTF_CLONED が設定されている経路も一緒に削除される。

RTF_LLINFO
RTF_LLINFO は、経路に有効なデータリンク層レベルの情報があることを示す。
これはネットワーク通信マネージャが設定するフラグである。具体的にはネットワーク通信マネージャ内の ARP が設定する。

RTF_STATIC
RTF_STATIC は、経路が手動で追加されたことを示す。アプリケーションから設定可能なフラグであり、
アプリケーションが経路を追加する場合にこのフラグを設定する。

RTF_BLACKHOLE
RTF_BLACKHOLE は、宛先へのパケットを破棄する。RTF_REJECT とは異なりパケットを破棄するのみで
エラーを返さない。アプリケーションから設定可能なフラグである。

RTF_CLONED
RTF_CLONED は、経路が RTF_CLONING により複製して生成されたことを示す。
アプリケーションから設定可能なフラグである。RTF_CLONED の設定を解除すると、RTF_CLONING が
設定されている元の経路を削除するときにまとめて削除されることを防ぐことができる。

RTF_PROTO2, RTF_PROTO1
RTF_PROTO1, RTF_PROTO2 は、プロトコルが独自に使用方法を規定して利用するフラグである。

rtm_addrs, ifm_addrs, ifam_addrs は、ルーティングメッセージに含まれているアドレスを示す。これらの値は RTA_ から始まるフラグの論理和である。

rtm_tid, rtm_seq は、メッセージを識別することに用い、ルーティングメッセージを送信するタスクが適切に値を設定する。

rtm_errno は、ルーティングメッセージ処理中に発生したエラーを示す。

EEXIST	既に存在している経路を追加しようとした場合
ESRCH	存在しない経路を削除しようとした場合
ENOBUFS	新しい経路を追加するために十分な資源がない場合

rt_metrics構造体で表されるように複数のルーティングメトリックが存在する。rtm_inits は初期化するルーティングメトリックを指定し、これの値は RTV_から始まるフラグの論理和である。経路情報のルーティングメトリックの初期値には rtm_inits のビットに対応する rtm_rmx の値が用いられる。

rtm_rmx は、経路情報のルーティングメトリックを示す。rt_metric 構造体の rmx_locks には、ネットワーク通信マネージャによる変更を許可しないルーティングメトリックを指定し、以下のフラグの論理和を設定する。

RTV_MTU	rmx_mtu に対応するフラグ
RTV_HOPCOUNT	rmx_hopcount に対応するフラグ
RTV_EXPIRE	rmx_expire に対応するフラグ
RTV_RPIPE	rmx_recvpipe に対応するフラグ
RTV_SPIPE	rmx_sendpipe に対応するフラグ
RTV_SSTHRESH	rmx_ssthresh に対応するフラグ
RTV_RTT	rmx_rtt に対応するフラグ
RTV_RTTVAR	rmx_rttvar に対応するフラグ

ifan_name は、デバイス名を示し、Neta などのデバイス名が格納される。

ifan_what は、インタフェースに関する通知の種類を示す。 これはIFAN_ARRIVAL または IFAN_DEPARTURE の値をとる。

IFAN_ARRIVAL	インタフェースの登録
IFAN_DEPARTURE	インタフェースの切り離し

5.6.2.1 RTM_ADD - 経路の追加

RTM_ADD メッセージは経路を追加する。このメッセージはアプリケーションからネットワーク通信マネージャへ送信される。

rtm_msglen, rtm_version, rtm_type には、それぞれルーティングメッセージ全体のサイズ、RTM_VERSION、RTM_ADD を指定する。

rtm_addrs には、少なくとも RTA_DST と RTA_GATEWAY を設定して、追加する経路を指定する。宛先としてネットワークのアドレスを指定する場合はRTA_NETMASK も設定する。また、RTA_IFP あるいは RTA_IFA を設定して、ネットワークインタフェースを明示的に指定することができる。ネットワークインタフェースを明示的に指定しなかった場合、システムが適切なネットワークインタフェースを選択する。

rtm_flags には、少なくとも RTF_STATIC を指定する。

rtm_tid, rtm_seq に、それぞれタスクIDとシーケンス番号を適切に設定する。

rtm_inits に初期化するルーティングメトリックに対応するフラグを指定して、ルーティングメトリックの値を rtm_rmx に設定する。
rtm_inits で指定されないルーティングメトリックは 0 に初期化される。

上記以外の rt_msghdr 構造体のメンバにはゼロを設定して送信する。

5.6.2.2 RTM_DELETE - 経路の削除

RTM_DELETE メッセージは経路を削除する。このとき、指定された経路から複製して作成した経路(フラグに RTF_CLONED が設定されている経路)もまとめて削除する。このメッセージはアプリケーションからネットワーク通信マネージャへ送信される。

rtm_msglen, rtm_version, rtm_type には、それぞれルーティングメッセージ全体のサイズ、RTM_VERSION、RTM_DELETE を指定する。

rtm_addrs には、RTA_DST を設定して、削除する経路を指定する。宛先としてネットワークのアドレスを指定する場合は RTA_NETMASK も設定する。

rtm_tid, rtm_seq に、それぞれタスクIDとシーケンス番号を適切に設定する。

上記以外の rt_msghdr 構造体のメンバにはゼロに設定して送信する。

5.6.2.3 RTM_CHANGE - 経路の変更

RTM_CHANGE メッセージは指定した経路のゲートウェイ、ネットワークインタフェースまたはフラグを変更する。このメッセージはアプリケーションからネットワーク通信マネージャへ送信される。

rtm_msglen, rtm_version, rtm_type には、それぞれルーティングメッセージ全体のサイズ、RTM_VERSION、RTM_CHANGE を指定する。

rtm_addrs には、少なくとも RTA_DST を設定して、変更する経路を指定する。宛先としてネットワークのアドレスを指定する場合は RTA_NETMASK も設定する。必要に応じてゲートウェイ、ネットワークインタフェースを設定する。

rtm_inits に変更するルーティングメトリックに対応するフラグを指定して、ルーティングメトリックの値を rtm_rmx に設定する。

rtm_tid, rtm_seq に、それぞれタスクIDとシーケンス番号を適切に設定する。

上記以外の rt_msghdr 構造体のメンバにはゼロを設定して送信する。

5.6.2.4 RTM_GET - 経路の取得

RTM_GET メッセージは経路情報を取得する。このメッセージはアプリケーションからネットワーク通信マネージャへ送信されて、ネットワーク通信マネージャは指定された経路の情報をメッセージに格納してアプリケーションへ返す。

rtm_msglen, rtm_version, rtm_type には、それぞれルーティングメッセージ全体のサイズ、RTM_VERSION、RTM_GET を指定する。

rtm_addrs には、少なくとも RTA_DST を設定して、取得する経路を指定する。宛先としてネットワークのアドレスを指定する場合は RTA_NETMASK も設定する。また、データリンク層のアドレス情報を取得する場合は、RTA_IFP を設定してメッセージヘッダ後には sockaddr_dl 構造体のアドレスを追加する。この sockaddr_dl 構造体の sdl_len, sdl_family にそれぞれ sockaddr_dl 構造体のサイズ、AF_LINK を設定し、他のメンバはゼロに設定する。

rtm_tid, rtm_seq に、それぞれタスクIDとシーケンス番号を適切に設定する。

上記以外の rt_msghdr 構造体のメンバにはゼロに設定して送信する。

5.6.2.5 RTM_LOSING - 宛先への到達失敗

RTM_LOSING メッセージは宛先へ到達失敗したことを通知する。具体的には、TCPセグメントを一定回数以上再送し

ても宛先に到達できなかったときに生成されるメッセージである。この場合、宛先への経路として誤ったゲートウェイを選択している疑いがある。このメッセージはネットワーク通信マネージャからアプリケーションへ送信される。

RTM_LOSING メッセージには到達失敗した経路の情報が格納される。

5.6.2.6 RTM_REDIRECT - ICMPリダイレクトの通知

RTM_REDIRECT メッセージはICMPリダイレクトメッセージにより経路が生成または変更されたことを通知する。このメッセージはネットワーク通信マネージャからアプリケーションへ送信される。

RTM_REDIRECT メッセージにはICMPリダイレクトメッセージにより追加された経路、あるいは変更後の経路が格納される。また、RTA_AUTHOR に ICMP リダイレクトメッセージを送信したノードのアドレスが設定される。

5.6.2.7 RTM_MISS - 経路の探索失敗

RTM_MISS メッセージは宛先への経路の探索失敗を通知する。宛先に一致するエントリがルーティングテーブルに存在しないときに探索失敗が発生する。このメッセージはネットワーク通信マネージャからアプリケーションへ送信される。

RTM_MISS メッセージには、経路探索に失敗した宛先のアドレスが格納される。

5.6.2.8 RTM_LOCK - 指定されたルーティングメトリックのロック

RTM_LOCK メッセージは指定した経路に対してネットワーク通信マネージャによる変更を許可・禁止するルーティングメトリックを設定する。他の項目は変更しない。このメッセージはアプリケーションからネットワーク通信マネージャへ送信される。

rtm_msglen, rtm_version, rtm_type には、それぞれルーティングメッセージ全体のサイズ、RTM_VERSION、RTM_LOCK を指定する。

rtm_rmx の rmx_locks に変更する値を設定する。

rtm_tid, rtm_seq に、それぞれタスクIDとシーケンス番号を適切に設定する。

上記以外の rt_msghdr 構造体のメンバにはゼロに設定して送信する。

5.6.2.9 RTM_NEWADDR - インタフェースにアドレスの追加

RTM_NEWADDR メッセージはインタフェースにアドレスが追加されたことを通知する。このメッセージはネットワーク通信マネージャからアプリケーションへ送信される。

追加された経路が ifa_msghdr 構造体をヘッダにもつメッセージに格納される。

5.6.2.10 RTM_DELADDR - インタフェースからアドレスの削除

RTM_DELETE メッセージはインタフェースからアドレスが削除されたことを通知する。このメッセージはネットワーク通信マネージャからアプリケーションへ送信される。

削除された経路が ifa_msghdr 構造体をヘッダにもつメッセージに格納される。

5.6.2.11 RTM_IFINFO - インタフェース/リンクの状態変化

RTM_IFINFO メッセージはインタフェースまたはリンクの状態が変化したことを通知する。このメッセージはネットワーク通信マネージャからアプリケーションへ送信される。

インタフェースとリンクの状態は if_msghdr 構造体をヘッダにもつメッセージに格納される。

5.6.2.12 RTM_IFANNOUNCE - インタフェースの着脱通知

RTM_IFANNOUNCE メッセージはインタフェースが登録または切り離しされたことを通知する。このメッセージはネットワーク通信マネージャからアプリケーションへ送信される。

インタフェースの着脱通知は if_announcemsghdr 構造体をヘッダにもつメッセージに格納される。

第6章　カレンダ機能

6.1 概要

カレンダ機能は、T2EX において、システム時刻 (SYSTIM および SYSTIM_U) および time_t 型の数値表現の時刻と、文字表現や要素別に分解した表現との間で、相互変換するための機能である。
API 名称のプレフィクスとして`dt_’ (date/time) を持つ。

T2EX におけるシステム時刻(SYSTIM および SYSTIM_U)は、協定世界時(UTC) で 1985年1月1日0時からの通算ミリ秒数(SYSTIM)および通算マイクロ秒数(SYSTIM_U)である。

6.2 定義

time_t

time_t 型は、POSIX において使われている時間を秒数単位であらわす整数データ型でカレンダ時刻と呼ぶ。
time_t 型で時刻を表現する際は、UTC で 1970年1月1日0時からの通算秒数とする。
この通算秒数の開始時刻(UTC で1970年1月1日0時)をエポックと呼ぶ。

カレンダ時刻およびシステム時刻と文字列表現および要素別時刻との相互変換には、time_t, SYSTIM, SYSTIM_U 型の
全ての型に対応したAPIコールを提供する。

struct tm

カレンダ時刻を要素別に表す構造体 struct tm を、以下のように定義する。
T2EX 固有の構造体メンバとして tm_usecを加え、ミリ秒・マイクロ秒レベルの時刻も情報として含めることができる。

```
#include <t2ex/datetime.h>

struct tm {
        int     tm_usec;                /* マイクロ秒 [0,999999] */
        int     tm_sec;                 /* 秒 [0,60] */
        int     tm_min;                 /* 分 [0,59] */
        int     tm_hour;                /* 時間 [0,23]*/
        int     tm_mday;                /* 月内の通算日 [1,31] */
        int     tm_mon;                 /* 月 [0,11] */
        int     tm_year;                /* 1900年からの年数 */
        int     tm_wday;                /* 曜日 [0,6] (0が日曜) */
        int     tm_yday;                /* 年内通算日 [0,365] */
        int     tm_isdst;               /* 夏時間 (正: 夏時間、0: 夏時間ではない、負: 不明) */
};
```

タイムゾーン

タイムゾーンとは、協定世界時(UTC)からのある一定の時差を、その地域の標準時として使用している地域全体のことを
指し、時間帯とも言う。
構造体 struct tzinfo は、そのタイムゾーンで使われている時間(ローカル時刻)をUTCからの差分で表す構造体である。

```
#define TZNAME_MAX 8                            /* タイムゾーン名の最大文字数 */

struct tzinfo {
        char                    tzname[2][TZNAME_MAX+1];        /* タイムゾーンの名前 */
        long                    offset;                         /* UTC からのオフセット秒数(西向きを正とする) */
        int                     daylight;                       /* サマータイム */

        /* 以下はサマータイム固有情報(daylight > 0 の場合のみ有効) */
        long                    dst_offset;                     /* サマータイム期間中のUTCからのオフセット秒数 */
        union dsttimespec       dst_start;                      /* サマータイム開始日時 */
        union dsttimespec       dst_end;                        /* サマータイム終了日時 */
};
```

daylight　　正の場合サマータイムがある
　　0 の場合サマータイムがない

dst_start, dst_end

サマータイム実施期間を示す。 dsttimespec 共用体は以下の通り定義される。

```
union dsttimespec {
        uint32_t                v;
        struct julian {
                unsigned int    type: 2,                        /* ローカル時刻の1月1日0:00:00からの通
算秒数による指定 (値は 0 または 1) */
                int             offset: 30,                     /* 通算秒数 [0,31622400] */
        } j;
        struct monthweekday {
                unsigned int    type: 2,                        /* 月・週数・日数を用いた指定(値は2)
*/
                unsigned int    m: 4,                           /* 月 [1,12] */
                unsigned int    n: 4,                           /* 週数 [1,5] */
                unsigned int    d: 4,                           /* 日数 [0,6] */
                int             offset: 18,                     /* 秒 [0,86400] */
        } m;
};
```

ローカル時間の1月1日0:00:00からの通算秒数による指定では、type = 0 の場合は通常の通算秒数とし、type = 1 の場合にはうるう日を数えない通算秒数を示す。後者では2月29日を明示的に参照することはできない。

月・週数・日数を用いた指定(type = 2)では、ローカル時間における、年内の m 月の第 n 週の第 d 番目の日の 0:00:00 から offset 秒経過した日時が指定される。

システムタイムゾーン

システムで唯一つ持っている現在のタイムゾーンをシステムタイムゾーンと呼ぶ。
これは、協定世界時(UTC)に対する時差を規定しているため、現在実行中のシステムのローカルな時刻を表現できる。
通常、システム起動時に dt_setsystz により、現在地域のタイムゾーンをシステムタイムゾーンとして設定する。
設定前の、システムタイムゾーンの初期値は実装依存とするが、T2EXリファレンス実装では、UTCとなっている。

タイムゾーン(tzinfo)のメモリーアドレスを指定する API コールにおいて、struct tzinfo * として NULL が指定された
場合、システムタイムゾーンが使われる。

ローカル時刻

タイムゾーンで規定されるその地域の時刻を、協定世界時(UTC)に対してそのタイムゾーンのローカル時刻と呼ぶ。
単にローカル時刻と言う場合、システムタイムゾーンのローカルな時刻を意味する。

タイムゾーン文字列

タイムゾーン文字列は、タイムゾーン情報を文字列で表現したもので以下のどちらかの形式を持つ:

1. :characters
   ':' で始まる文字列の場合 characters は実装依存
   T2EXリファレンス実装では、この形式のタイムゾーン文字列はサポートしない。
   dt_tzset() API コールで指定した場合は、エラーとなる。

2. std offset dst offset, rule

   std　　標準の名前
   dst　　代替の名前

   offset　協定世界時(UTC)を得るためにローカル時刻に加えられる値
   　　hh[:mm[:ss]] の形式を持つ

   　　hh　　時間 [0,24]
   　　mm　　分 [0,59]
   　　ss　　秒 [0,59]

   　　std には offset が必要である。
   　　dst の後の offset は省略することが出来る。dst 後の offset が省略された場合は、
   　　std の offset の一時間前とみなす。

   タイムゾーン文字列が '-' で始まる場合、グリニッジ子午線の東側であることを意味する。

西側に対しては、オプションで'+'を付けても良い。

rule　代替時間と変更する時間または戻る時間を示す。以下の形式を持つ:
date[/time],date[/time]

最初の date は、標準から代替時間にいつ変更するかを示す。
次の date は代替時間から標準時間にいつ戻るかを示す。
各 time フィールドは、別の時間への変更が行われる時をローカル時刻で示す。

date は以下の形式を持つ:
Jn
ユリウス日 n (1<=n<=365)。うるう日は数えてはならない。
うるう年を含む全ての年で、2月28日は、n=59 で、3月1日は n=60 となる。
2月29日を明示的に参照することはできない。
n
0ベースのユリウス日(0<=n<=365)。うるう日は数えられ、
2月29日を参照できる。
Mm.n.d
年の m 月の n 週の d 番目の日(1<=m<=12, 1<=n<=5, 0<=d<=6)。
n = 1 は、d番目の日が起きる最初の週である。d = 0 が日曜日を表す。

time は、符号('+' または '-')が付かない点を除き offset と同じ形式である。
time が省略された場合は 02:00:00 とみなす。

タイムゾーン文字列の例:
- UTC から時差があり、夏時間も採用しているニュージーランドの場合:
"NZST-12:00:00NZDT-13:00:00,M10.1.0,M3.3.0"

標準の名前	NZST(ニュージーランド標準時)
時差	-12時間 (UTCより12時間進んでいる)
代替の名前	NZDT(ニュージーランド夏時間)
時差	-13時間 (UTCより13時間進んでいる)
代替に切り替わる日付	10月第1日曜日
代替から戻る日付	3月の第3日曜日

- 日本の場合:
"JST-9"

名前は JST のみで代替の名前は無い。
UTC より9時間進んでいるだけで、夏時間は無い。

システムロケール

ロケールは、日付や時刻の表現形式、文字照合順序、数値の表現形式、通貨記号等の各カテゴリ毎に国や地域に応じた
振る舞いを規定するものである。
T2EX では、ロケールを動的に自由に設定する機能は持たず、固定したデフォルトのロケールを使用する。
これをシステムロケールと呼ぶ。
システムロケールは通常、T2EX を搭載した機器を使用する国や地域、あるいは利用目的に応じて、
カテゴリ毎にシステム構成時にカスタマイズした固定値を用いるものとする。
T2EXリファレンス実装では、システムロケールは ISO C ロケール ("C") となっている。
したがって、エラーメッセージの取得APIコールでは、英語のASCII文字列が戻され、また時刻を文字列表現に変換した
場合も、月や曜日は英語の文字列が戻される。

6.3 API

dt_setsystz, dt_getsystz の 2 関数のみシステムコールとして実装され、残りの関数はライブラリ関数として実装される。

6.3.1 dt_main - カレンダ機能の初期化・終了

【C言語インタフェース】

```
#include <t2ex/datetime.h>

ER      er = dt_main(INT ac, UB* arg[]);
```

【パラメータ】

INT	ac	arg[] の要素数または負の値
UB*	arg[]	引数文字列の配列

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了

【解説】

カレンダ機能を初期化(ac >= 0)または終了 (ac < 0) する。
初期化時は、arg[] に ac 個の文字列を引数として渡すことができる。
arg の内容は実装依存である。T2EXリファレンス実装では、引数文字列は使用していない。

6.3.2 dt_setsystz - システムタイムゾーンの設定

【C言語インタフェース】

```
#include <t2ex/datetime.h>

ER er = dt_setsystz(const struct tzinfo* tz);
```

【パラメータ】

const struct tzinfo*	tz	設定するタイムゾーンへのポインタ

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了
EX_FAULT	tz のアドレスが不正
EX_INVAL	tz の内容が不正

【解説】

システムのタイムゾーンを tz で与えられるタイムゾーンに変更する。

【関連項目】

dt_getsystz,dt_tzset

6.3.3 dt_getsystz - システムタイムゾーンの取得

【C言語インタフェース】

```
#include <t2ex/datetime.h>

ER er = dt_getsystz(struct tzinfo* tz);
```

【パラメータ】

struct tzinfo*	tz	取得するタイムゾーンへのポインタ

【リターンパラメータ】

ER	er	エラーコード
struct tzinfo*	tz	システムタイムゾーン

【エラーコード】

E_OK	正常終了
EX_FAULT	tz のアドレスが不正

【解説】

システムタイムゾーンの値を tz が示す領域に返す。

【関連項目】

dt_setsystz,dt_tzset

6.3.4 dt_tzset - タイムゾーン構造体の初期化

【C言語インタフェース】

```
#include <t2ex/datetime.h>

ER er = dt_tzset(struct tzinfo* tz, const char* spec);
```

【パラメータ】

struct tzinfo*	tz	タイムゾーンへのポインタ
const char*	spec	タイムゾーン文字列

【リターンパラメータ】

ER	er	エラーコード
struct tzinfo*	tz	初期化されたタイムゾーン

【エラーコード】

E_OK	正常終了
EX_INVAL	spec の文字列がタイムゾーン文字列として不正

【解説】

spec で与えられるタイムゾーン文字列から、タイムゾーン構造体 tz を初期化する。

【関連項目】

dt_getsystz, dt_setsystz

6.3.5 dt_localtime, dt_localtime_ms, dt_localtime_us - ローカル時刻への変換

【C言語インタフェース】

```
#include <t2ex/datetime.h>

ER er = dt_localtime(time_t tims, const struct tzinfo* tz, struct tm* result);

ER er = dt_localtime_ms(const SYSTIM* tim, const struct tzinfo* tz, struct tm* result);

ER er = dt_localtime_us(SYSTIM_U tim_u, const struct tzinfo* tz, struct tm* result);
```

【パラメータ】

time_t	tims	秒単位のシステム時刻
const SYSTIM*	tim	ミリ秒単位のシステム時刻へのポインタ
SYSTIM_U	tim_u	マイクロ秒単位のシステム時刻
struct tzinfo*	tz	タイムゾーンへのポインタ
struct tm	*result	ローカル時刻の格納領域へのポインタ

【リターンパラメータ】

ER	er	エラーコード
struct tm*	result	変換されたローカル時刻

【エラーコード】

E_OK	正常終了
EX_FAULT	パラメータのアドレスが不正
EX_OVERFLOW	結果が tm で表現できない

【解説】

tims, tim または tim_u で示されるシステム時刻を、tz で示されるタイムゾーンのローカル時刻に変換し、結果を result が指す tm 構造体に格納する。

tz に NULL を指定した場合、システムタイムゾーンが使われる。

【関連項目】

dt_setsystz, dt_getsystz, dt_tzset, dt_strftime, dt_strptime, dt_mktime, dt_gmtime

6.3.6 dt_strftime - 日付と時刻の文字列への変換

【C言語インタフェース】

```
#include <t2ex/datetime.h>

int nb = dt_strftime(char *s, size_t max, const char* format, const struct tm* tm, const struct
tzinfo* tz);
```

【パラメータ】

char*	s	結果の文字列を格納する領域へのポインタ
size_t	max	s が指す領域のサイズ(バイト数)
const char*	format	変換書式指定文字列
const struct tm*	tm	文字列に変換する時刻
const struct tzinfo	tz	タイムゾーン

【リターンパラメータ】

int	nb	ヌル文字を含まない結果のバイト数 またはエラーコード
char*	s	変換結果の文字列

【エラーコード】

EX_INVAL	不正なパラメータ

【解説】

dt_strftime() は、struct tm で表された時刻を format で指定した書式と tz で指定されたタイムゾーンにしたがって文字列に変換し、ヌル文字も含めて最大 max 文字まで s で指定された領域に書き込む。
時刻を文字列表現に変換するには、一般にロケール情報により異なる変換結果となる。
dt_strftime() においては、ロケール情報はシステムのデフォルトのロケール(システムロケール)が使われる。

format は変換書式を表す文字列で、0 個以上の変換を指定する文字列(変換指示子)と通常の文字からなる。
各変換指示子は、'%' 文字で始まり続いて以下の文字の並びがその順で現れる。

1. E または O 修飾子(省略可能)。詳細は後述。

2. 変換タイプを指定する変換指定文字

最後のヌル文字を含む通常の文字は結果にコピーされる。オーバーラップしているオブジェクト間でコピーが行われた場合の結果は不定である。(例えば s と format が同じ値であった場合)

文字列 s には max バイト以上書き込まれない。各変換指示子は、以下のリストにある適切な文字列に置換される。
その適切な文字列は、システムロケールおよび tm が指す要素別時刻の 0 個以上のメンバにより決定される。

指定された値が正常な範囲外の場合、格納される文字列は不定とする。
tz として NULL を与えた場合、`%Z' および `%z' 指定子の解釈においてシステムタイムゾーンが用いられる。

以下の変換指定文字をサポートする:
'[' ']'で囲んだものは、結果が数字列のときの数字の範囲である。
'{' '}'で囲んだものは、参照する tm 構造体のメンバである。

a
曜日の省略名 {tm_wday}
A
曜日の完全な名前 {tm_wday}
b
月の省略名 {tm_mon}
B
月の完全な名前 {tm_mon}
c
システムロケールにおける一般的な日付・時刻
C
年を100で割って10進整数に切り捨てたもの。[00,99] {tm_year}
d
月内通算日 [01,31] {tm_mday}
D
%m/%d/%y と等価 {tm_mon, tm_mday, tm_year}
e
月内通算日。一桁の場合、先頭に空白文字を置く [1,31] {tm_mday}
F
%Y-%m-%d に等しい。{tm_mon, tm_mday, tm_year}
g
週単位で表記した年(下記参照)の下2桁 [00,99] {tm_year, tm_wday, tm_yday}
G
週単位で表記した年 {tm_year, tm_wday, tm_yday}
h
%b に等しい {tm_mon}
H
時間(24時間制) [00,23] {tm_hour}
I
時間(12時間制) [01,12] {tm_hour}
j
年内通算日 [001,366] {tm_yday}
m
月 [01,12] {tm_mon}
M
分 [00,59] {tm_min}
n
改行
p
システムロケールにおける午前、午後に相当する文字列
r
午前、午後表記の時刻 {tm_hour, tm_min, tm_sec}
R
24時間表記の時刻 (%H:%M) {tm_hour, tm_min}
S
秒 [00,60] {tm_sec}
t
タブ
T
時刻 (%H:%M:%S) {tm_hour, tm_min, tm_sec}
u
1を月曜とした曜日 [1,7] {tm_wday}
U
年内の週番号 [00,53]
1月の最初の日曜日が、週番号1の最初の日。
新年のそれ以前の日は、週番号0 {tm_year, tm_wday, tm_yday}
V
月曜を週内の最初の日とした年内の週番号 [01,53]
1月1日を含んだ週が4日以上ある場合、週番号1とみなす。
<<<<<<< .mine
1月1日を含んだ週が3日未満の場合は前年の最終週となり、その翌週が1となる。
=======
そうでない場合は、その週は前年の最終週とみなされ、その翌週が1となる。
>>>>>>> .r1920
1月4日と1月の最初の木曜日は常に1となる。{tm_year, tm_wday, tm_yday}
w

週内の日番号 [0,6]。0 は日曜日とする。{tm_wday}

W
年内の週番号 [00,53]。1月の最初の月曜を週番号1とする。それ以前の日は0とする。
{tm_year, tm_wday, tm_yday}

x
システムロケールにおける日付表現

X
システムロケールにおける時刻表現

y
年の下2桁 [00,99] {tm_year}

Y
年 {tm_year}

z
UTC からのオフセットの ISO 8601:2000 標準形式

Z
システムタイムゾーン名

%
'%'文字

E または O 修飾子

いくつかの変換指示子は E または O 修飾子を付けることができる。
この修飾子を付けた場合、代替の書式があればその代替の書式で出力される。
システムロケールに対して代替の書式が無い場合は、修飾子無しの場合と同じ動作となる。

%Ec
ロケールの代替日付および時刻

%EC
ロケールの代替表現での基準年(期間)の名前

$Ex
ロケールの代替日付

%Ey
ロケールの代替時刻

%EY
完全な代替表現での年

%Od
ロケールの代替数値シンボルを用いた月内通算日。
ゼロの代替シンボルがある場合は先頭は必要ならゼロで埋められ、無い場合は空白で埋められる。

%Oe
ロケールの代替数値シンボルを用いた月内通算日。
必要なら、先頭は空白で埋められる。

%OH
ロケールの代替数値シンボルを用いた時間(24時制)

%OI
ロケールの代替数値シンボルを用いた時間(12時制)

%Om
ロケールの代替数値シンボルを用いた月

%OM
ロケールの代替数値シンボルを用いた分

%OS
ロケールの代替数値シンボルを用いた秒

%Ou
ロケールの代替数値シンボルを用いた曜日番号(月曜=1)

%OU
ロケールの代替数値シンボルを用いた年内週番号
(%U 同様に日曜を週の初日とする。)

%OVV
ロケールの代替数値シンボルを用いた年内週番号
(%V 同様に月曜を週の初日とする。)

%Ow
ロケールの代替数値シンボルを用いた曜日番号(日曜=0)

%OW
ロケールの代替数値シンボルを用いた年内週番号(月曜を週の初日とする)

%Oy
ロケールの代替数値シンボルを用いた年(%C からのオフセット)

%g, %G および %V は、ISO 8601:2000 標準の年の初めからの週番号を用いる。
この仕組みでは、週は月曜から始まり、年の第一週は、1月4日を含む週であり、年の最初の木曜も含む。
また、年の最初の週は少なくとも4日を含む。

1月の第一月曜が、2, 3, または 4日目の場合、最初の1～3日は前年の最後の週の一部となる。
したがって、1999年1月2日の土曜に対して、%G は 1998 に置き換えられ、%V は、53 に置き換えられる。

12月29日、30日、および31日が月曜の場合、それらの日は翌年の第一週の一部となる。
したがって、1997年12月30日の火曜日は、%Gでは 1998 に置き換えられ、%V では 01 に置き換えられる。

変換指示子が上記の一つ以外の場合、動作は未定義である。

【関連項目】

dt_strptime, dt_mktime

6.3.7 dt_strptime - 文字列の日付および時刻への変換

```
#include <t2ex/datetime.h>

int index = dt_strptime(const char *str, const char *format, struct tm *tm);
```

【パラメータ】

const char*	str	変換する文字列
const char*	format	変換書式文字列
struct tm*	tm	結果の日付時刻を格納するtm構造体

【リターンパラメータ】

int	index	str中の処理されなかった最初の文字へのインデックス またはエラーコード
struct tm*	tm	文字列から変換された時刻

【エラーコード】

EX_INVAL	不正なパラメータ

【解説】

dt_strptime() は、str で示す文字列を format で指定した書式にしたがってtm構造体の時刻に変換し結果を tm に格納する。

format は、0 個以上の指令から成り、各指令は以下のいずれかである:
・1個以上の isspace() が真となる空白文字
・'%' および上記空白文字を除く通常の文字
・変換指示子

各変換指示子は、'%' で始まり、以下のものがその順に現れる:
1. E または O 修飾子(省略可能)
2. 変換のタイプを指定する変換指示子

変換に必要なロケール情報には、システムロケールが使用される。

すべての隣接する変換指示子が、変換指示子によって決まっている数の文字を変換するのでないならば、
アプリケーションは、任意の二つの変換指示子の間に、空白または英数字以外の文字があることを保証しなければならない。

以下の変換指示子がサポートされている。

a
　　ロケールの曜日名を用いた曜日。省略形か完全な名前が指定できる。
A
　　%a に等しい。
b
　　ロケールの月名を用いた月。省略形か完全な名前が指定できる。
B
　　%b に等しい。
c
　　ロケールの標準の日付と時刻。
C
　　年を100で割って切り捨てた整数値[00,99]。
　　先行のゼロは許されているが必須ではない。
d
　　月内の日にち[01,31]。先頭のゼロは必須ではない。
D
　　%m/%d/%y と同じ。

e
　　%d に等しい。
h
　　%b に等しい。
H
　　時間[00,23]。先頭のゼロは必須ではない。
I
　　時間[01,12]。先頭のゼロは必須ではない。
j
　　年内の通算日[001,366]。先頭のゼロは必須ではない。
m
　　月[01,12]。先頭のゼロは必須ではない。
M
　　分[00,59]。先頭のゼロは必須ではない。
n
　　空白文字
p
　　a.m. または p.m. に相当するロケールでの表現。
r
　　AM/PM表記での12時間制の時刻。
R
　　%H:%M と同じ。
S
　　秒[00,60]。先頭のゼロは必須ではない。
t
　　空白文字。
T
　　%H:%M:%S と同じ。
U
　　(日曜を週内の最初の日とした)年内の週番号[00,53]。先頭のゼロは必須ではない。
　　リファレンス実装では値は無視され、 tm には格納されない。
w
　　曜日の数字[0,6]。0 が日曜。
　　リファレンス実装では値は無視され、 tm には格納されない。
W
　　(月曜を週内の最初の日とした)年内の週番号[00,53]。先頭のゼロは必須ではない。
x
　　ロケールの日付形式の日付。
X
　　ロケールの時刻形式の時刻。
y
　　年の下2桁。format が C も Y も含まない場合、[69,99] の値は、1969から1999 を意味し、
　　[00,68] の値は、2000から2068を意味する。先頭のゼロは必須ではない。
　　先頭の'+'または'-'文字は許されているが、必須ではない。
Y
　　4桁の年。先頭のゼロは必須ではない。
　　先頭の'+'または'-'文字は許されているが必須ではない。
%
　　'%' 文字。

E または 0 修飾子

幾つかの変換指示子は、代替書式が使われるよう E および 0 修飾子によって修飾される。
代替書式がシステムロケールで存在しない場合は、修飾子無しの場合と同じ振る舞いとなる。

%Ec
　　ロケールの代替日付および時刻
%EC
　　ロケールの代替表現での基準年(期間)の名前
%Ex
　　ロケールの代替日付
%EX
　　ロケールの代替時刻
%Ey
　　ロケールの代替表現での %EC (年のみ)からのオフセット
%EY
　　完全な代替表現での年
%Od
　　ロケールの代替数値シンボルでの月内の通算日。先頭のゼロは必須ではない。
%Oe
　　%Od に等しい
%OH
　　ロケールの代替数値シンボルでの(24時間制の)時間

%OI
　　ロケールの代替数値シンボルでの(12時間制の)時間
%Om
　　ロケールの代替数値シンボルでの月
%OM
　　ロケールの代替数値シンボルでの分
%OS
　　ロケールの代替数値シンボルでの秒
%OU
　　ロケールの代替数値シンボルでの(日曜を週内の初日とした)年内の週番号
%Ow
　　ロケールの代替数値シンボルでの(日曜を0とした)曜日番号
%OW
　　ロケールの代替数値シンボルでの(月曜を週内の初日とした)年内の週番号
%Oy
　　ロケールの代替数値シンボルでの(%C からのオフセットの)年

空白文字からなる変換指示子の場合、空白でない最初の文字までスキャンされるか、文字がなくなるまでスキャンされる。

通常の文字の変換指示子の場合、バッファから次の文字をスキャンする。スキャンされた文字が指令のものと異なる場合は、指令は失敗し異なる文字と残りはスキャンされないまま残る。

%n, %t, 空白文字またはその組み合わせで構成される変換指定の場合、空白(スキャンされずに残っている)でない最初の文字まで、または文字がなくなるまでスキャンされる。

それ以外の変換指示子の場合、次の指令が一致する文字まで、または文字がなくなるまでスキャンされる。
これらの文字は、次の指令に一致するものを除き、変換指示子と結び付けられたロケールの値と比較される。
一致した場合、適切な tm 構造体のメンバの値は、ロケールの情報に対応した値に設定される。
月や曜日名等の str 内の項目を比較するときは、大文字小文字の違いは無視される。
一致が見つからない場合、dt_strptime() は失敗し、文字はそれ以上スキャンされない。

【関連項目】

dt_strftime

6.3.8 dt_mktime, dt_mktime_ms, dt_mktime_us - システム時刻への変換

【C言語インタフェース】

```
#include <t2ex/datetime.h>

ER er = dt_mktime(struct tm* tm, const struct tzinfo* tz, time_t* result);
ER er = dt_mktime_ms(struct tm* tm, const struct tzinfo* tz, SYSTIM* result);
ER er = dt_mktime_us(struct tm* tm, const struct tzinfo* tz, SYSTIM_U* result);
```

【パラメータ】

struct tm*	tm	変換する時刻データ
const struct tzinfo*	tz	タイムゾーン
time_t*	result	変換結果のシステム時刻(秒単位)
SYSTIM*	result	変換結果のシステム時刻(ミリ秒単位)
SYSTIM_U*	result	変換結果のシステム時刻(μ秒単位)

【リターンパラメータ】

ER	er	エラーコード
struct tm*	tm	tm_wday, tm_yday が適切に設定される
time_t*	result	変換結果のシステム時刻(秒単位)
SYSTIM*	result	変換結果のシステム時刻(ミリ秒単位)
SYSTIM_U*	result	変換結果のシステム時刻(μ秒単位)

【エラーコード】

E_OK	正常終了
EX_OVERFLOW	変換結果を指定された形式で表現できない

【解説】

tm で指定されたローカル時刻の要素別のデータを tz で指定されたタイムゾーンの元でAPIコールで指定された形式のシステム時刻に変換する。
dt_mktime は秒単位の time_t 形式に変換する。
dt_mktime_ms はミリ秒単位の SYSTIM 形式に変換する。
dt_mktime_us はμ秒単位の SYSTIM_U 形式に変換する。
tz が NULL の場合は、システムタイムゾーンが用いられる。
tm の元の値のうち、tm_wday および tm_yday は無視される。

成功した場合には、tm_wday と tm_yday 要素は適切に設定され、他の要素はシステム時刻の開始時刻(1985年1月1日0時(GMT))からの値に設定される。
result には、NULL を指定することも可能である。この場合でも tm の要素は適切に設定される。

結果が result の型で表現できない場合は、tm の内容は変化せず EX_OVERFLOW のエラーを返す。
【関連項目】

dt_gmtime

6.3.9 dt_gmtime, dt_gmtime_ms, dt_gmtime_us - UTC 時刻への変換

【C言語インタフェース】

```
#include <t2ex/datetime.h>

ER er = dt_gmtime(time_t tim, struct tm* result);
ER er = dt_gmtime_ms(const SYSTIM* tim_m, struct tm* result);
ER er = dt_gmtime_us(SYSTIM_U tim_u, struct tm* result);
```

【パラメータ】

time_t	tim	システム時刻(秒単位)
const SYSTIM*	tim_m	システム時刻(ミリ秒単位)
SYSTIM_U	tim_u	システム時刻(μ秒単位)
struct tm*	result	UTC 時刻への変換結果

【リターンパラメータ】

ER	er	エラーコード
struct tm*	result	UTC 時刻のtm構造体

【エラーコード】

E_OK	正常終了
EX_OVERFLOW	変換結果が表現できない

【解説】

システム時刻を協定世界時(UTC)で表現された要素別の時刻に変換し result に格納する。
dt_gmtime は秒単位のシステム時刻を与える。
dt_gmtime_ms はミリ秒単位のシステム時刻を与える。
dt_gmtime_us はμ秒単位のシステム時刻を与える。
結果の result の解像度は、tim_u, tim_m, tim で与えたものとなる。

【関連項目】

dt_mktime

第7章　プログラムロード機能

7.1 概要

プログラムロード機能は、T2EX におけるプログラムモジュールをロードし、実行するための機能である。
API 名称のプレフィクスとして`pm_’(program module) を持つ。

本機能が対象とするプログラムモジュールは、以下の2種類に分類される。

一般プログラムモジュール
　　アプリケーションプログラムやシステムレベルのコンポーネントから利用する際に、
　　利用する側と同一の保護レベルで動作するプログラムモジュールである。ロード後の
　　モジュールの利用はSVC割り込みを伴わず関数呼び出しとして実行される。

システムプログラムモジュール
　　システムメモリ空間に配置される特権的なプログラムモジュールのことであり、
　　T-Kernel 2.0 におけるサブシステムおよびデバイスドライバがこれにあたる。
　　ユーザレベルのプログラムからシステムに関する特権レベルの操作を安全かつ整理された形で
　　アプリケーション側に提供する想定で利用される。システムプログラムの機能の利用は、
　　デバイスドライバインタフェースやサブシステムのエントリポイントなどを用いて、
　　SVC割り込み経由で行われる。

一般プログラムモジュールとシステムプログラムモジュールの本質的な違いは、プログラムモジュールが実行される保護レベルの違いにある。前者では呼び出し元の保護レベルのままでモジュール機能が実行されるのに対し、後者では呼び出し元の実行時保護レベルに関わらず常にシステムレベルでモジュールが実行される。
一般的には、アプリケーションのプログラムをモジュール単位に分割するような用途には、実行効率や安全性の面から一般プログラムモジュールを用いることが望ましい。システムプログラムモジュールはデバイスドライバやサブシステムを含む、システムレベルのインタフェースを提供する目的に限って用いられるべきである。

7.2 一般プログラムモジュール

7.2.1 一般プログラムモジュールのインタフェース

一般プログラムモジュールのエントリポイントの名称は module_main であり、以下の形式を持つ。

```
int module_main(BOOL startup, void* arg)
{
        if ( startup ) {
                /* 一般プログラム起動処理 */
        }
        else {
                /* 一般プログラム終了処理 */
        }

        return ercd;
}
```

startup が TRUE の場合は起動処理の実行を意味し、モジュールが利用可能な状態となるように初期化を行わなければならない。エラー(負の値)を戻したときは起動に失敗したものとみなされるが、この場合は起動処理によって確保された資源は解放しなければならない。引数 arg は起動処理のパラメータとして用いることができる。

一方、startup が FALSE の場合は終了処理が実行される。終了処理はモジュールが割り当てた資源を解放しなければならない。終了処理中にエラーが発生した場合にも終了処理を中止してはならず、可能な限り資源の解放を行うものとする。一部の終了処理が正常に実行できなかった場合は、戻値としてエラーを返す。
引数 arg は終了処理のパラメータとして用いることができる。

エントリポイント自体は起動処理または終了処理を行うだけで、モジュールが提供するサービスへのインタフェースは別途提供されるものとする。本仕様ではこのための方法を特に限定しないが、一般的には起動時の引数 arg が指す領域に、モジュールのインタフェース関数群のポインタを設定する方法が想定される。
具体的なモジュールの定義例については、付録 A.3.1 節を参照すること。

7.2.2 一般プログラムモジュールの利用

一般プログラムモジュールの利用開始は、以下の手順で行われる。

1. 一般プログラムモジュールのロード
　　対象となるプログラムモジュールをメモリ上にロードする。これは
　　pm_load() を用いて行うことができる。

2. 起動処理の呼び出し
　　pm_load() によって得られたエントリポイントを startup = TRUE として呼び出す。
　　これによって、モジュールの起動処理が実行される。

一方、一般プログラムモジュールの終了は以下の手順によって行われる。

1. 終了処理の呼び出し
   pm_load() によって得られたエントリポイントを startup = FALSE として呼び出す。
   これによって、モジュールの終了処理が実行され、モジュールが割り当てた資源が解放される。
2. 一般プログラムモジュールのアンロード
   対象となるプログラムモジュールをメモリ上からアンロードする。これは
   pm_unload() を用いて行うことができる。

具体的なモジュールの利用方法の例については、付録 A.3.2 節を参照すること。

7.3 システムプログラムモジュール

システムプログラムのインタフェース仕様は、T-Kernel 2.0 のサブシステム・デバイスドライバのインタフェースに準ずる。エントリポイントの形式も以下のとおりであり、それに準じたものである。
仕様の詳細については、T-Kernel 2.0 仕様書の 5.11節「サブシステムおよびデバイスドライバの起動」を参照すること。

```
ER main(INT ac, UB* av[])
{
        if ( ac >= 0 ) {
                /* システムプログラム起動処理 */
        }
        else {
                /* システムプログラム終了処理 */
        }

        return ercd;
}
```

システムプログラムモジュールのロードと起動ならびに終了とアンロードはそれぞれ不可分な処理であり、これらはそれぞれ pm_loadspg(), pm_unload() でまとめて実行される。これらについての詳細は、7.5節を参照すること。

7.4 データ型の定義

・pm_entry_t
   一般プログラムモジュールのエントリポイント(module_main)の型

```
typedef int pm_entry_t(BOOL startup, void* arg);
```

・struct pm
   ロード対象のプログラムモジュールの指定に用いる型

```
struct pm {
        ATR     pmtype;  /* プログラムの種別 (PM_FILE, PM_PTR) */
        void*   pmhdr;   /* ロードするプログラム */

        /* その他の実装依存情報 */
};
```

pmtype は以下の通り指定される。

```
pmtype := (PM_FILE || PM_PTR)
```

pmtype が PM_FILE であるときロード対象のプログラムモジュールはファイルであり、pmhdr にはファイルパス名を示す文字列(char*型)へのポインタを指定する。pmtype が PM_PTR であるときは対象のプログラムモジュールはメモリ上に配置されているものとして扱われ、モジュールが配置されているメモリ領域の先頭アドレスを pmhdr に指定する。

ロードされるプログラムのモジュールの形式については本仕様では規定せず、実装依存とする。

7.5 API

本機能における全てのAPIコールはシステムコールとして実装される。本機能によって提供されるAPIコールにおいて戻値が負の場合、戻値は ER 型の拡張エラーコードとして解釈されるものとする。

7.5.1 pm_main - プログラムロード機能の初期化・終了

【C言語インタフェース】

```
#include <t2ex/load.h>

ER      er = pm_main(INT ac, UB* arg[]);
```

【パラメータ】

INT	ac	arg[] の要素数または負の値
UB*	arg[]	引数文字列の配列

【リターンパラメータ】

ER	er	エラーコード

【エラーコード】

E_OK	正常終了

【解説】

プログラムロード機能を初期化(ac >= 0)または終了 (ac < 0) する。
初期化時は、arg[] に ac 個の文字列を引数として渡すことができる。
arg の内容は実装依存である。T2EXリファレンス実装では、引数文字列は使用していない。

7.5.2 pm_load - 一般プログラムモジュールのロード

【C言語インタフェース】

```
#include <t2ex/load.h>

ID progid = pm_load(const struct pm* prog, UINT attr, pm_entry_t** entry);
```

【パラメータ】

const struct pm*	prog	ロード対象のプログラム
UINT	attr	ロード先のメモリ空間の属性
pm_entry_t**	entry	プログラムのエントリポイントを返す領域へのポインタ

【リターンパラメータ】

ID	progid	正常終了のときプログラムID (> 0)、またはエラーコード (< 0)
pm_entry_t*	entry	プログラムのエントリポイント

【エラーコード】

E_LIMIT	ロードできるプログラム数が上限に達した
E_MACV	引数として与えられたメモリアドレスへのアクセスが不可 (prog, entry)
EX_ACCES	対象のプログラムをロードする許可がない
EX_FBIG	対象となるプログラムのファイルが大きすぎる
EX_INTR	fs_break() による中止
EX_INVAL	不正なパラメータ - ac < 0 - prog, attr が無効
EX_IO	I/Oエラー
EX_ISDIR	prog がディレクトリである
EX_NFILE	システムでオープン可能なファイル数を超えた
EX_NODEV	ロード対象のファイルの接続ポイントが存在しない
EX_NOENT	ロード対象のファイルが存在しない
EX_NOEXEC	ロード対象のプログラム形式が不正である
EX_PERM	対象のプログラムをロードする許可がない

【解説】

一般プログラムモジュール prog をメモリ上にロードする。モジュールの起動処理(エントリポイント)の呼び出しは行われない。

ロード先のメモリ空間の属性は、attr を用いて以下の通り指定する:
attr := (TA_RNG0 || TA_RNG1 || TA_RNG2 || TA_RNG3)

正常にロードが行われた場合、戻値としてプログラムIDが返される。このとき、*entry にはロードされた一般プログラムモジュールのエントリポイントのポインタが返される。得られた関数ポインタを TRUE, FALSE を第1引数に指定して呼び出すことで、モジュールに対する初期化・終了処理をそれぞれ実行できる。

ロードしたプログラムモジュールはアンロードされるまでの間、得られたエントリポイントの関数ポインタを通して、単純な関数呼び出しにより繰り返し実行することが可能である。

【関連項目】

pm_loadspg, pm_unload

7.5.3 pm_loadspg - システムプログラムモジュールのロード

【C言語インタフェース】

```
#include <t2ex/load.h>

ID progid = pm_loadspg(const struct pm* prog, INT ac, UB* av[]);
```

【パラメータ】

const struct pm*	prog	ロード対象のプログラム
INT	ac	起動処理時にエントリポイントに渡す引数の数 (≧ 0)
UB*	av[]	起動処理時にエントリポイントに渡す文字列引数リスト

【リターンパラメータ】

ID	progid	正常終了のときプログラムID (> 0)、またはエラーコード (< 0)

【エラーコード】

E_LIMIT	ロードできるプログラム数が上限に達した
E_MACV	引数として与えられたメモリアドレスへのアクセスが不可 (prog, av)
EX_ACCES	対象のプログラムをロードする許可がない
EX_FBIG	対象となるプログラムのファイルが大きすぎる
EX_INTR	fs_break() による中止
EX_INVAL	不正なパラメータ - ac < 0 - prog, av が無効
EX_IO	I/Oエラー
EX_ISDIR	prog がディレクトリである
EX_NFILE	システムでオープン可能なファイル数を超えた
EX_NODEV	ロード対象のファイルの接続ポイントが存在しない
EX_NOENT	ロード対象のファイルが存在しない
EX_NOEXEC	ロード対象のプログラム形式が不正である
EX_PERM	対象のプログラムをロードする許可がない

【解説】

システムプログラムモジュール prog を特権レベルのメモリ空間にロードし、起動処理を本APIの呼び出し元タスクの準タスク部として実行する。

起動処理においてエントリポイントが0または正の値を返したとき、正常に起動処理が実行されたものとし、戻値としてプログラムIDが返される。
一方、エントリポイントが負の値を返したとき、システムプログラムのロードは失敗したものとして処理され、エントリポイントの戻値がそのまま返される。
後者のケースでは終了処理が自動的に呼び出されることはないので、システムプログラムモジュール側では起動処理からエラーを返す前に
獲得したリソース等の解放を行っておく必要がある。

起動処理の終了(エントリポイントからのリターン)とともに本APIコール自体の実行は完了するが、プログラムはアンロードされるまでの間、メモリにマップされた状態となる。

【関連項目】

pm_load, pm_unload

7.5.4 pm_status - プログラムモジュールの情報参照

【C言語インタフェース】

```
#include <t2ex/load.h>

ER ercd = pm_status(ID progid, struct pm_stat* status);
```

【パラメータ】

ID	progid	対象のプログラムID
struct pm_stat*	status	プログラムの状態を返す領域へのポインタ

【リターンパラメータ】

ER	ercd	エラーコード
status の内容		
BOOL	sysprg	プログラム属性(TRUE:システムプログラム, FALSE:一般プログラム)
UINT	attr	ロード先のメモリの属性
void*	entry	プログラムのエントリポイントのアドレス
void*	start	プログラムの先頭アドレス
void*	end	プログラムの終端アドレス

【エラーコード】

E_OK	正常終了
E_ID	プログラムIDが不正
E_MACV	引数として与えられたメモリアドレスへのアクセスが不可 (status)

【解説】

プログラムモジュールID progid で示されるプログラムの情報を参照し、*status に情報を格納する。

【関連項目】

pm_loadspg, pm_load, pm_unload

7.5.5 pm_unload - プログラムモジュールのアンロード

【C言語インタフェース】

```
#include <t2ex/load.h>

ER ercd = pm_unload(ID progid);
```

【パラメータ】

ID	progid	対象のプログラムID

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	プログラムIDが不正

【解説】

プログラムモジュールID progid で示されるプログラムモジュールをアンロードする。

アンロードされたプログラムモジュールを実行したり、同プログラムの持つメモリ領域を参照した場合の動作は保証されない。

pm_loadspg() によってロードされたシステムプログラムモジュールに対して本APIコールが実行された場合は、以下の処理が実行される。

- 終了処理として、対象となるシステムプログラムのエントリポイント(main 関数)を呼び出す。
  この時、第1引数 ac は (-1) となっており、ロード時の初期化処理と区別することが可能である。
  呼び出された main 関数は、pm_unload() の発行タスクの準タスク部として実行される。

- main 関数が0または正の値を返した場合は終了処理が成功したものと見なし、プログラムのアンロードを実行する。一方、main 関数が負の値を返した場合は終了処理が成功しなかったものと見なし、アンロードは実行されず main 関数の負の戻値がそのまま pm_unload() の戻値となる。

pm_load() によってロードされた一般プログラムモジュールに対して本APIコールが実行された場合は、終了処理の
呼び出しは行わずにプログラムのアンロードのみを実行する。したがって、本APIコールを用いてアンロードを実行する前に必ず終了処理(エントリポイント)を呼び出さなければならない。

【関連項目】

pm_loadspg, pm_load

第8章　標準C互換ライブラリ

8.1 概要

標準C互換ライブラリは、標準Cライブラリ(JIS X3010:2003)と互換性の高いライブラリ群である。
T2EX は、API がスレッドセーフである事を重視している。そのため、標準Cライブラリの中で、スレッドセーフではない関数は、排除したり、同等機能の別関数を用意したり、引数を追加したりすることでスレッドセーフなものに変更している。
また、ロケール等T2EXが提供しない機能に関連した関数は削除している。
したがって、標準Cライブラリと完全な互換性は持っていない。

T2EX では、以下のヘッダファイルとそこで定義される関数を提供する。

arpa/inet.h	BSD ソケット　※1
assert.h	診断機能
complex.h	複素数計算
ctype.h	文字種分類
dirent.h	ディレクトリの読出し　※2
errno.h	エラー番号定義
float.h	浮動小数点型の特性
inttypes.h	整数型の書式の変換
iso646.h	代替つづり(Alternate spellings)
limits.h	各種制限値
math.h	数値演算
netinet/in.h	BSD ソケット　※1
search.h	検索　※2
stdarg.h	可変個の実引数
stdbool.h	論理型および論理値
stddef.h	共通定義
stdint.h	整数型
stdio.h	標準入出力
stdlib.h	一般ユーティリティ
string.h	文字列操作
time.h	日付及び時間
wchar.h	多バイトおよびワイド文字拡張

※1 TCP/IP を実現するソケットを扱うために、BSD ソケットで定義されているヘッダを追加している。

※2 汎用性が高いものとして、POSIX (IEEE Std 1003.1-2008) に基づいて追加している。

以下のヘッダは、標準Cライブラリ(JIS X3010:2003)では定義されているが、標準C互換ライブラリからは削除している。

fenv.h	浮動小数点環境 ※ T2EX では、浮動小数点例外を定義しないため。
locale.h	ロケール操作 ※ T2EX では、ロケールを操作する関数はサポートしておらず、システムロケール一つに固定されているため。
setjmp.h	非局所分岐 ※ 一般にマルチタスク下で使用するには制約があるため。
signal.h	シグナル操作 ※ T2EX では、シグナルは存在しないため。
tgmath.h	型総称数学関数 ※ T2EX では、その必要性を認められないため。
wctype.h	ワイド文字種分類およびワイド文字大文字小文字変換 ※ T2EX ではワイド文字/マルチバイト文字ライブラリをサポートしないため。

8.2 互換性

T2EXでは、標準Cライブラリの各ヘッダで定義されている関数について一部をサポートしていない。
また、関数名にポストフィックスをつけて引数を追加している場合もある。
したがって、T2EX 標準C互換ライブラリは、標準Cライブラリとの完全な互換性は持っていない。
標準Cライブラリと異なる点は、以下の通りである。

ファイルとソケット

標準Cライブラリでは、ファイルとソケットはストリームとして同様に扱うことができる。
T2EX 標準C互換ライブラリでは、ファイルとソケットは同一の関数で扱うことはできない。
ファイルを扱う関数とソケットを扱う関数は別々に存在する。
特に、stdio.h で規定される C 言語標準入出力ライブラリでは、関数はファイルに対してのみ使用できる。
ソケットに対して、標準入出力関数を用いることはできない。

エラーコードとエラー番号

T2EX システムコール群では、全てのAPIコールは ER 型の戻値を持ち、その値からエラーの詳細を知ることができる。
標準Cライブラリの関数では、戻値に -1 や NULL を用いてエラーの発生は通知するが詳細なエラー情報は戻値からは得られない。
通常 POSIX 仕様に従ったオペーレーティングシステムではシンボル errno の内容を読み出すことによりエラー情報を取得する。
errno の値は正の整数値を持つ詳細なエラー情報であり、この正の整数値をエラー番号と呼び、T2EX の ER 型の負のエラーコードと区別する。

補足: なお、T2EXでは、POSIX仕様のerrnoに相当する静的変数は導入していない。
マクロと関数を用いてerrnoを定義し、タスク毎のエラー情報をerrnoに持たせるような仕様や実装も考えられるが、仕様の複雑化や実行時のオーバーヘッドから、この仕様は採用していない。

本章で述べる標準C互換ライブラリの関数でも、関数の戻値でエラーの発生は通知するがエラーの詳細情報は得られない。
また、システムコール群のAPIコールのように ER 型を用いたエラーコードを返す関数は存在しない。

標準C互換ライブラリの関数では、errno に相当するエラー番号は以下の方法で取得する:
1. 発生したエラー番号を返す関数を呼び出す。ferror() 関数。
2. 関数自体にエラー番号を返す領域を指定する引数を追加し、引数で指定した領域にエラー番号を戻す。
   fopen_eno() 等。

エラー番号は以下の型で表現する。

```
typedef int             errno_t;
```

エラー番号を T2EX エラーコード体系にマップし、ER 型として統一して扱うための手段を提供している。
エラー番号に対応して、以下のようにしてエラーコードを派生させる。
・メインエラーコードとして EC_ERRNO を持つ。
・サブエラーコードとしてエラー番号を持つ。
・エラー番号に対応したエラーコードのシンボルとして EX_.... を定義する。

EC_ERRNO をメインエラーコードに持つエラーコードを拡張エラーコードと呼ぶ。
エラー番号値(eno)を、T2EX のER型の拡張エラーコードに変換するには、以下のマクロを用いる。

```
#define ERRNOtoER(eno)  (ERCD(EC_ERRNO, (eno))
```

以下のマクロにより、ER型の拡張エラーコード(er)から errno_t 型のエラー番号の値を得ることができる。

```
#define ERRNO(er)               (MERCD(er) == EC_ERRNO ? SERCD(er) : 0)
```

・T2EX では、エラー番号の取得方法が、標準Cライブラリで規定されている errno ではないため、標準Cライブラリ関数にエラー番号を返す引数を追加して、関数名に _eno のポストフィックスを追加している。
以下の関数がそれに相当する。機能的には、元の名前の関数と等価である。

```
fopen_eno
fdopen_eno
freopen_eno
fclose_eno
opendir_eno
closedir_eno
gmtime_r_eno
localtime_r_eno
mktime_eno
realpath_eno
realpath2_eno
```

エラー番号の詳細については、errno.h の節を参照すること。

スレッドセーフ

T2EX では、以下のスレッドセーフではない関数はサポートしない。

```
asctime
ctime
gets
gmtime
```

```
localtime
rand
readdir
srand
strerror
strtok
```

また、以下のスレッドセーフではない関数については、引数を追加し、関数名に _r を
ポストフィックスとして付け加えスレッドセーフな関数に変更している。

```
lgamma_r
lgammaf_r
lgammal_r

drand48_r
lrand48_r
mrand48_r
srand48_r
seed48_r
lcong48_r
```

大容量ファイル

T2EX では、64bit サイズの大容量ファイルをサポートする。64ビットのファイルオフセットを扱う
ための関数は、従来の関数と並存して利用できるように別な名前で追加している。

```
fgetpos64
fsetpos64
fseek64
ftell64
```

ファイルロック

T2EX では、ファイルロック関連機能はサポートしない。

```
flockfile
ftrylockfile
funlockfile
_unlocked
```

パイプ

T2EX では、プロセスが存在しないため、パイプを扱う関数はサポートしない。

```
popen
pclose
```

ロケール

T2EX では、locale.h は提供しない。したがって、ロケールを任意に設定する機能や lconv 構造体で
ロケールの情報を得る機能も提供しない。
ただし、文字照合等ロケールを参照する必要がある場合に、システムが持つデフォルトの固定した
ロケールを参照する。このデフォルトのロケールをシステムロケールと呼ぶ。
システムロケールの値は実装定義である。
T2EXリファレンス実装では、システムロケールは USA (en_US) である。

その他

errno を暗黙に必要とする関数はサポートしない。

```
perror
```

関数記述形式

本章では、以前の章と異なり、各関数の説明は以下の形式で記述している。

【名称】
【書式】
【解説】
【戻値】
【エラー】
【関連項目】
【補足事項】

【エラー】の項に記述されたエラー番号は、各関数が -1 や NULL 等、関数ごとに決まった値を

返すことでエラー発生を通知した場合に、以下で得られるエラー番号である。
・ferror() 関数の戻値
・関数が errno_t* 型の引数を持つ場合に、その引数に格納されるエラー番号

APIコール

標準C互換ライブラリが提供する関数やマクロは、T2EX API の一部であり、その一つ一つは
T2EX API コールに相当する。
ただし、本章では互換性という観点からAPIコールという表記は用いず、標準Cライブラリの一般的な
記述にしたがって「関数」および「マクロ」という表記を用いる。

文字・文字列

T2EX では、文字コードとして UTF-8 を使用することを前提としている。
UTF-8 は、1 文字は1バイトとは限らず、複数バイトで表されるマルチバイト文字を用いている。
したがって全ての文字に対し 1 文字を表現するためにワイド文字 wchar_t 型が存在している。
しかし T2EX 標準C互換ライブラリでは、マルチバイト文字やワイド文字に関連したライブラリ関数は
提供しない。
したがって、本章の記述では、「文字」と表記した場合、1バイトを意味する。
「文字列」と表記した場合、NULL で終了するバイト列を意味する。

浮動小数点

浮動小数点の表現形式および演算は、原則として IEC 60559 (IEEE 754) に従うものとする。
アーキテクチャの都合でそれに従うことが出来ない実装も許容する。
ただし、この規格で規定される浮動小数点例外(INVALID, DENORMAL, ZERIDIVIDE, OVERFLOW,
UNDERFLOW, INEXACT)は、T2EX では発生しない。
そのような例外の発生する演算については、NAN, INFINITY, HUGE_VAL 等の何らかの値を適切に返す。

その他

数値の区間を表すのに [0,1] という表記を用いる。これは 0 以上、1 以下の区間である。
[0.0, 1.0) であれば、0.0 以上、1.0 未満の範囲を意味する。
(0.0, 1.0] であれば、0.0 より大きく、1.0 以下の範囲を意味する。
また、±0 という表記は、+0 または -0 を意味する。

8.3 arpa/inet.h

ヘッダ arpa/inet.h は、以下のマクロや構造体、および関数プロトタイプ宣言を定義する。

型

in_port_t
　　netinet/in.h で定義される型

in_addr_t
　　netinet/in.h で定義される型

in_addr
　　netinet/in.h で定義される構造体

マクロ

INET_ADDRSTRLEN
　　netinet/in.h で定義されるマクロ

関数またはマクロ

8.3.1 htonl, htons, ntohl, ntohs - ホストとネットワークバイトオーダーの変換

【書式】

```
#include <arpa/inet.h>

uint32_t        htonl(uint32_t hostlong);
uint16_t        htons(uint16_t hostshort);
uint32_t        ntohl(uint32_t netlong);
uint16_t        ntohs(uint16_t netshort);
```

【解説】

htonl(), htons(), ntohl(), ntohs() は、16ビットおよび32ビットのデータに対する、ホストのバイトオーダーとネットワークのバイトオーダー間の変換を行う。
htonl(), htons(), ntohl(), ntohs() は、T2EXリファレンス実装では関数として実装しているが、マクロとして実装してもよい。

【戻値】

htonl(), htons() は、hostlong, hostshort 値をホストのバイトオーダーからネットワークバイトオーダーに変換して返す。
ntohl(), ntohs() は、netlong, netshort 値をネットワークバイトオーダーからホストバイトオーダーに変換して返す。

【エラー】

なし

8.3.2 inet_addr - 文字列のIPv4アドレスへの変換

【書式】

```
#include <arpa/inet.h>

in_addr_t       inet_addr(const char *cp);
```

【解説】

inet_addr() は、cp が示す標準のドット付き10進表記された文字列を、インターネットアドレスの整数値に変換する。
インターネットアドレスは、ネットワークオーダーで返す。(左から右のバイト順)

ドット付き10進表記による文字列は、以下の形式の一つをとる:

a.b.c.d
　四つの部分を指定した場合、各部は1バイトのデータと解釈し、左から右に、4バイトのインターネットアドレスとする。

a.b.c
　三つの部分を指定した場合、最後の部分は16ビットデータと解釈し、ネットワークアドレスの右側2バイトとする。
　これは、"128.net.host" のような Class B ネットワークアドレスを指定するのを容易にする。

a.b
　二つの部分を指定した場合、最後の部分は24ビットデータと解釈し、ネットワークアドレスの右側3バイトとする。
　これは、"net.host" のような Class A ネットワークアドレスを指定するのを容易にする。

a
　一部のみを指定した場合、値は、そのままネットワークアドレスとする。

IPv4のドット付き10進表記の各数字部分は、ISO C 標準の10進、8進、16進数のいずれでもよい。
(すなわち、0x または 0X で始まると16進数、 '0' で始まると8進数、その他は10進数と解釈される。)

【戻値】

inet_addr() は、成功した場合、インターネットアドレスを返す。それ以外の場合 (in_addr_t)(-1)を返す。

【エラー】

なし

【補足事項】

標準Cライブラリには inet_ntoa() という関数が存在するが、スレッドセーフではないため T2EX では提供しない。
inet_ntop() 関数で代替する。

8.3.3 inet_ntop, inet_pton - IPv4 アドレスのバイナリとテキストとの変換

【書式】

```
#include <arpa/inet.h>

const char      *inet_ntop(int af, const void *src, char *dst, socklen_t size);
int             inet_pton(int af, const char *src, void *dst);
```

【解説】

inet_ntop() は、数値のPアドレスを文字列に変換する。
af は、アドレスファミリーを指定する。これは AF_INET である。
src は、af が AF_INET の場合、IP アドレスを保持しているバッファを指す。
IPアドレスは、ネットワークバイトオーダーでなければならない。
dst は、結果の文字列を格納するバッファを指す。
size は、dst のバッファのサイズを指定するもので文字列を格納するのに十分な大きさが必要である。
(IPv4 に対しては、INET_ADDRSTRLEN 以上)

inet_pton() は、標準の文字列表現のIPアドレスを数値IPアドレスに変換する。
af は、アドレスファミリーを指定する。AF_INET のアドレスファミリーをサポートする必要がある。
src は、変換する文字列を指す。
dst は、数値IPアドレスを格納するバッファを指す。これは数値アドレスを格納するのに十分な大きさが必要である。
(AF_INET に対しては、32ビット)

inet_pton() の af が AF_INET の場合、src 文字列は、下記に示す標準のIPv4ドット付き10進形式でなければならない:

　　ddd.ddd.ddd.ddd

ここで、"ddd" は、0 から 255 の間の10進数字である。inet_pton() は、他の形式を受け付けない

(8進数、16進数、4部分以外の形式等 inet_addr() では受け付ける形式)。

【戻値】

inet_ntop() は、変換に成功した場合は、テキスト文字列を格納したバッファのポインタを返し、それ以外の場合はNULLを返す。

inet_pton() は、変換に成功した場合は、dst が指す領域にネットワークバイトオーダーのアドレスを格納し 1 を返す。
入力文字列が正しい IPv4ドット付き10進文字列ではない場合、0 を返す。
af が未知の場合 -1 を返す。

【エラー】

なし

8.4 assert.h

ヘッダ assert.h は、以下の診断用マクロを定義する。

マクロ

assert()
assert() マクロは、assert.h ヘッダ内では定義されないマクロ NDEBUG を参照している。
assert.h をインクルードする前に NDEBUG が定義されていると、
assert() マクロは以下のように定義される。

```
#define assert(ignore) ((void) 0)
```

NDEBUG が定義されていない場合、assert() マクロは下記の assert() の項で説明されている void 型の式として定義される。

assert.h がインクルードされるたびに assert() マクロは、NDEBUG の現在の状態に応じて再定義される。

assert() は、マクロとして実装する必要がある。

8.4.1 assert - 診断の挿入

【書式】

```
#include <assert.h>

void    assert(scalar expression);
```

【解説】

assert() マクロは、プログラムに診断を挿入する。このマクロは void 型の式に展開される。

スカラ型の式が偽の場合、assert() は、標準エラー出力に情報を出力し abort() を呼び出して異常終了を発生させる。
出力される情報には、引数の文字列表現、ソースファイル名、ソースの行番号、および assert() を呼び出した関数名を含む。
最後の三つは、マクロ __FILE__, __LINE, および識別子 __func__ の値である。

コンパイラのコマンドラインか、プリプロセッサ制御文で #include <assert.h> の前に NDEBUG を定義すると、診断はコンパイルされたコードに含まれない。

【戻値】

なし

8.5 complex.h

ヘッダ complex.h は、以下の複素数に関するマクロや関数プロトタイプ宣言を定義する。

マクロ

complex
: _Complex に展開される

_Complex_I
: 虚数部の値を持つ const float _Complex 型の定数式に展開される

imaginary
: _Imaginary に展開される

_Imaginary_I
: 虚数部の値を持つ const float _Imaginary 型の定数式に展開される

I
: _Imaginary_I が未定義の場合 _Complex_I に展開され、定義されている場合 _Imaginary_I に展開される。

imaginary と _Imaginary_I は、実装が虚数型をサポートしているときのみ定義される。
T2EXリファレンス実装では、虚数型はサポートせず、これらは定義されていない。

関数

8.5.1 cabs, cabsf, cabsl - 複素数の絶対値

【書式】

```
#include <complex.h>

double          cabs(double complex z);
float           cabsf(float complex z);
long double     cabsl(long double complex z);
```

【解説】

これらの関数は、複素数 z の絶対値を計算する。

【戻値】

これらの関数は、複素数の絶対値(ノルム(norm), モジュラス(modulus)、または大きさ(magnitude)ともよばれる。)を返す。

【エラー】

なし

8.5.2 cacos, cacosf, cacosl - 複素数逆余弦

【書式】

```
#include <complex.h>

double complex          cacos(double complex z);
float complex           cacosf(float complex z);
long double complex     cacosl(long double complex z);
```

【解説】

これらの関数は、複素数 z の逆余弦を計算する。

【戻値】

これらの関数は、虚軸方向には数学的に無限の区間，実軸方向には区間[0,+π]を値域とする複素数逆余弦値を返す。

【エラー】

なし

【関連項目】

ccos

8.5.3 cacosh, cacoshf, cacoshl - 複素数の逆双曲線余弦

【書式】

```
#include <complex.h>

double complex          cacosh(double complex z);
float complex           cacoshf(float complex z);
long double complex     cacoshl(long double complex z);
```

【解説】

これらの関数は、複素数 z の逆双曲線余弦を計算する。

【戻値】

これらの関数は、実数軸方向には0以上である半区間，虚数軸方向には区間[−iπ,+iπ]を値域とする複素数逆双曲線余弦値を返す。

【エラー】

なし

【関連項目】

ccosh

8.5.4 carg, cargf, cargl - 複素数の位相角

【書式】

```
#include <complex.h>

double          carg(double complex z);
float           cargf(float complex z);
long double     cargl(long double complex z);
```

【解説】

これらの関数は、複素数 z の位相角を計算する。

【戻値】

これらの関数は、区間[−π,+π]を値域とする位相角の値を返す。

【エラー】

なし

【関連項目】

cimag, conj, cproj

8.5.5 casin, casinf, casinl - 複素数の逆正弦

【書式】

```
#include <complex.h>

double complex          casin(double complex z);
float complex           casinf(float complex z);
long double complex     casinl(long double complex z);
```

【解説】

これらの関数は、複素数 z の逆正弦を計算する。

【戻値】

これらの関数は、虚数軸方向には数学的に無限の区間, 実数軸方向には区間[−π/2,+π/2]を値域とする複素数逆正弦値を返す。

【エラー】

なし

【関連項目】

csin

8.5.6 casinh, casinhf, casinhl - 複素数の逆双曲線余弦

【書式】

```
#include <complex.h>

double complex          casinh(double complex z);
float complex           casinhf(float complex z);
long double complex     casinhl(long double complex z);
```

【解説】

これらの関数は、複素数 z の逆双曲線余弦を計算する。

【戻値】

これらの関数は、実数軸方向には数学的に無限の区間, 虚数軸方向には区間[−iπ/2,+iπ/2]を値域とする複素数逆双曲線正弦値を返す。

【エラー】

なし

【関連項目】

csinh

8.5.7 catan, catanf, catanl - 複素数の逆正接

【書式】

```
#include <complex.h>

double complex          catan(double complex z);
float complex           catanf(float complex z);
long double complex     catanl(long double complex z);
```

【解説】

これらの関数は、複素数 z を逆正接を計算する。

【戻値】

これらの関数は、虚数軸方向には数学的に無限の区間, 実数軸方向には区間[−π/2,+π/2]を値域とする複素数逆正接値を返す。

【エラー】

なし

【関連項目】

ctan

8.5.8 catanh, catanhf, catanhl - 複素数の逆双曲線正接

【書式】

```
#include <complex.h>

double complex          catanh(double complex z);
float complex           catanhf(float complex z);
long double complex     catanhl(long double complex z);
```

【解説】

これらの関数は、複素数 z の逆双曲線正接を計算する。

【戻値】

これらの関数は、実数軸方向には数学的に無限の区間, 虚数軸方向には区間[−iπ/2,+iπ/2]を値域とする複素数逆双曲線正接値を返す。

【エラー】

なし

【関連項目】

ctanh

8.5.9 ccos, ccosf, ccosl - 複素数の余弦

【書式】

```
#include <complex.h>

double complex          ccos(double complex z);
float complex           ccosf(float complex z);
long double complex     ccosl(long double complex z);
```

【解説】

これらの関数は、複素数 z の余弦を計算する。

【戻値】

これらの関数は、複素数の余弦を返す。

【エラー】

なし

【関連項目】

cacos

8.5.10 ccosh, cconshf, cconshl - 複素数の逆双曲線余弦

【書式】

```
#include <complex.h>

double complex          ccosh(double complex z);
float complex           ccoshf(float complex z);
long double complex     ccoshl(long double complex z);
```

【解説】

これらの関数は、複素数 z の逆双曲線余弦を計算する。

【戻値】

これらの関数は、複素数の逆双曲線余弦を返す。

【エラー】

なし

【関連項目】

cacosh

8.5.11 cexp, cexpf, cexpl - 複素数の指数関数値

【書式】

```
#include <complex.h>

double complex          cexp(double complex z);
```

```
float complex           cexpf(float complex z);
long double complex     cexpl(long double complex z);
```

【解説】

これらの関数は、自然対数の底 e の複素数 z 乗を計算する。

【戻値】

これらの関数は、複素数の指数関数値を返す。

【エラー】

なし

【関連項目】

clog

8.5.12 cimag, cimagf, cimagl - 複素数の虚数部

【書式】

```
#include <complex.h>

double          cimag(double complex z);
float           cimagf(float complex z);
long double     cimagl(long double complex z);
```

【解説】

これらの関数は、複素数 z の虚数部を計算する。

【戻値】

これらの関数は、複素数の虚数部を実数値として返す。

【エラー】

なし

【関連項目】

carg, conj, cproj, creal

8.5.13 clog, clogf, clogl - 複素数の自然対数

【書式】

```
#include <complex.h>

double complex          clog(double complex z);
float complex           clogf(float complex z);
long double complex     clogl(long double complex z);
```

【解説】

これらの関数は、複素数 z の(e を底とする)自然対数を計算する。

【戻値】

これらの関数は、実数軸方向には数学的に無限の区間，虚数軸方向には区間[−i π,+i π]を値域とする複素数の自然対数値を返す。

【エラー】

なし

【関連項目】

cexp

8.5.14 conj, conjf, conjl - 複素共役

【書式】

```
#include <complex.h>

double complex          conj(double complex z);
float complex           conjf(float complex z);
long double complex     conjl(long double complex z);
```

【解説】

これらの関数は、複素数 z の虚数部の符号を反転し複素共役を計算する。

【戻値】

これらの関数は、複素共役の値を返す。

【エラー】

なし

【関連項目】

carg, cimag, cproj, creal

8.5.15 cpow, cpowf, cpowl - 複素数のべき乗

【書式】

```
#include <complex.h>

double complex          cpow(double complex x, double complex y);
float complex           cpowf(float complex x, float complex y);
long double complex     cpowl(long double complex x, long double complex y);
```

【解説】

これらの関数は、複素数 x の複素数 y 乗を計算する。

【戻値】

これらの関数は、複素数のべき乗関数値を返す。

【エラー】

なし

【関連項目】

cabs, csqrt

8.5.16 cproj, cprojf, cprojl - リーマン球面への射影

【書式】

```
#include <complex.h>

double complex          cproj(double complex z);
float complex           cprojf(float complex z);
long double complex     cprojl(long double complex z);
```

【解説】

これらの関数は、複素数 z のリーマン球面への射影を計算する。
無限の値を持つ複素数(たとえ, 実数部又は虚数部の一方が無限大であり, 他方が NaN であっても)は、
実軸上の正の無限大に射影することを除き、複素数 z は、z 自身に射影する。
z の実数部又は虚数部が無限大である場合, cproj(z)は,

```
        INFINITY + I * copysign(0.0, cimag(z))
```

と等価である。

【戻値】

これらの関数は、リーマン球面への射影の値を返す。

【エラー】

なし

【関連項目】

carg, cimag, conj, creal

8.5.17 creal, crealf, creall - 複素数の実数部

【書式】

```
#include <complex.h>

double          creal(double complex z);
float           crealf(float complex z);
long double     creall(long double complex z);
```

【解説】

これらの関数は、複素数 z の実数部を求める。

【戻値】

これらの関数は、実数部を返す。

【エラー】

なし

【関連項目】

carg, cimag, conj, cproj

8.5.18 csin, csinf, csinl - 複素数の正弦

【書式】

```
#include <complex.h>

double complex          csin(double complex z);
float complex           csinf(float complex z);
long double complex     csinl(long double complex z);
```

【解説】

これらの関数は、複素数 z を正弦を計算する。

【戻値】

これらの関数は、複素数の正弦の値を返す。

【エラー】

なし

【関連項目】

casin

8.5.19 csinh, csinhf, csinhl - 複素数の双曲線正弦

【書式】

```
#include <complex.h>

double complex          csinh(double complex z);
float complex           csinhf(float complex z);
long double complex     csinhl(long double complex z);
```

【解説】

これらの関数は、複素数 z の双曲線正弦を計算する。

【戻値】

これらの関数は、複素数の双曲線正弦値を返す。

【エラー】

なし

【関連項目】

casinh

8.5.20 csqrt, csqrtf, csqrtl - 複素数の平方根

【書式】

```
#include <complex.h>

double complex          csqrt(double complex z);
float complex           csqrtf(float complex z);
long double complex     csqrtl(long double complex z);
```

【解説】

これらの関数は、複素数 z の平方根を計算する。

【戻値】

これらの関数は、(虚数軸を含む)右半面内の複素数平方根の値を返す。

【エラー】

なし

【関連項目】

cabs, cpow

8.5.21 ctan, ctanf, ctanl - 複素数の正接

【書式】

```
#include <complex.h>

double complex          ctan(double complex z);
float complex           ctanf(float complex z);
long double complex     ctanl(long double complex z);
```

【解説】

これらの関数は、複素数 z の正接を計算する。

【戻値】

これらの関数は、複素数の正接の値を返す。

【エラー】

なし

【関連項目】

catan

8.5.22 ctanh, ctanhf, ctanhl - 複素数の双曲線正接

【書式】

```
#include <complex.h>

complex                 ctanh(double complex z);
float complex           ctanhf(float complex z);
long double complex     ctanhl(long double complex z);
```

【解説】

これらの関数は、複素数 z の双曲線正接を計算する。

【戻値】

これらの関数は、複素数の双曲線正接の値を返す。

【エラー】

なし

【関連項目】

catanh

8.6 ctype.h

ヘッダ ctype.h は、以下の文字種判定の関数およびマクロを定義する。

関数またはマクロ

```
int   isalnum(int c);
              c が英字または数字の場合 0 以外の値を返す。

int   isalpha(int c);
              c が英字の場合 0 以外の値を返す。

int   isascii(int c);
              c が 7ビット US-ASCII 文字コードの場合 0 以外の値を返す。

int   isblank(int c);
              c がブランク文字(スペースかタブ)の場合 0 以外の値を返す。

int   iscntrl(int c);
              c が制御文字の場合 0 以外の値を返す。

int   isdigit(int c);
              c が数字の場合 0 以外の値を返す。

int   isgraph(int c);
              c が表示可能な文字の場合 0 以外の値を返す。

int   islower(int c);
              c が小文字の場合 0 以外の値を返す。

int   isprint(int c);
              c が印刷可能な文字の場合 0 以外の値を返す。

int   ispunct(int c);
              c が句読点文字の場合 0 以外の値を返す。

int   isspace(int c);
              c が空白の場合 0 以外の値を返す。

int   isupper(int c);
              c が大文字の場合 0 以外の値を返す。

int   isxdigit(int c);
              c が16進数字の場合 0 以外の値を返す。
```

上記、文字種判定関数またはマクロにおいて、c は、unsigned char で表現できる値か、
EOF でなければならない。それ以外の場合動作は未定義とする。

```
int   toascii(int c);
              c を 7bit US-ASCII に変換して返す。(上位ビットをクリアする)

int   tolower(int c);
              c が小文字に出来る文字の場合、小文字に変換して返す。それ以外の場合 c を返す。

int   toupper(int c);
              c が大文字に出来る文字の場合、大文字に変換して返す。それ以外の場合 c を返す。
```

以下はマクロによる定義に限定する。

```
int   _toupper(int c);
              小文字 c を大文字に変換して返す。

int   _tolower(int c);
              大文字 c を小文字に変換して返す。
```

文字種の判定は、システムロケールに基づいて行う。

8.7 dirent.h

ヘッダ dirent.h は、ディレクトリエントリをストリームのように読み込むための以下の構造体および関数プロトタイプ宣言を定義する。

型

DIR 型
　　ディレクトリストリームを表現する型
　　T2EXリファレンス実装では、DIR型はファイルディスクリプタを含んだ構造体となっている。

struct dirent 構造体
　　ディレクトリエントリを表す構造体で、以下の要素を含む。

```
ino_t   d_ino                           ファイル通し番号
char    d_name[NAME_MAX+1]              エントリの名前
```

ino_t 型
　　ファイルシステム内のファイル通し番号またはファイルを識別する番号。

関数

標準Cライブラリにある readdir() は、スレッドセーフではないので T2EX では提供しない。
代わりに readdir_r() を用いる。

8.7.1 closedir_eno, closedir - ディレクトリストリームのクローズ

【書式】

```
#include <dirent.h>

int     closedir_eno(DIR *dirp, errno_t* enop);
int     closedir(DIR *dirp);
```

【解説】

closedir_eno() は、dirp が参照するディレクトリストリームをクローズする。
エラーが発生した場合、そのエラー番号を enop が指す領域に格納する。
enop を NULL とした場合には、エラー番号は格納されない。
完了時、dirp の値は、もはや DIR型のアクセス可能なオブジェクトをポイントしていない。
DIR型の実装で、ファイルディスクリプタを使用している場合、そのファイルディスクリプタはクローズされる。

closedir() は、closedir_eno(dirp, NULL) と等価である。

【戻値】

成功した場合、これらの関数は 0 を返す。エラー発生時、これらの関数は -1 を返す。

【エラー】

エラー発生時、enop が指す領域に以下が格納される。

```
EBADF           dirp がオープンされたディレクトリストリームを参照していない
EINTR           fs_break() による中止
```

【関連項目】

opendir, opendir_eno, readdir_r

8.7.2 opendir_eno, opendir - ディレクトリストリームのオープン

【書式】

```
#include <dirent.h>

DIR     *opendir_eno(const char *path, errno_t* enop);
DIR     *opendir(const char *path);
```

【解説】

opendir_eno() は、path で指定したディレクトリ名のディレクトリをオープンし、DIR 型の
ディレクトリストリームを返す。
オープンしたディレクトリストリームは、最初のエントリを指している。
DIR型が、ファイルディスクリプタを用いて実装されている場合、システムがオープンできるファイルディスクリプタ数を超えてオープンすることは出来ない。
DIR型で使用するファイルディスクリプタは、fs_open() のフラグとして O_DIRECTORY を指定して得られるものでなければならない。
エラーが発生した場合、そのエラー番号を enop が指す領域に格納する。
enop == NULL とした場合には、エラー番号は格納されない。

opendir(path) は、opendir_eno(path, NULL) と等価である。

【戻値】

opendir(), opendir_eno() は、成功した場合 DIR型へのポインタを返す。エラー発生時、これらの関数は NULL を返す。

【エラー】

エラー発生時、enop が指す領域に以下が格納される。

EACCES	path 中のディレクトリの読込み許可属性がない
ENAMETOOLONG	ファイル名が長すぎる - path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX) - path 全体の長さが長すぎる(最大PATH_MAX)
ENOENT	path が存在しない
ENOTDIR	path がディレクトリではない
ENFILE	システム内でオープン中のファイルディスクリプタ数の制限を超える

【関連項目】

closedir, closedir_eno, readdir_r

8.7.3 readdir_r - ディレクトリエントリの読込み

【書式】

```
#include <dirent.h>

errno_t readdir_r(DIR *dirp, struct dirent *entry, struct dirent **result);
```

【解説】

readdir_r() は、dirp で指定された、ディレクトリストリームの現在位置の
ディレクトリエントリを読み込む。そして entry が指す struct dirent 型のデータを読み込んだ内容で初期化し、
このデータへのポインタを result が指す場所に格納し、ディレクトリストリームを次の位置に移動する。

entry が指す領域は、要素 char d_name[] のサイズが NAME_MAX+1 文字以上の dirent 型を格納できる大きさでなければならない。

成功した場合、*result は entry と同じ値を持つ。ディレクトリストリームの終端に達した場合は *result は NULL となる。

readdir_r() は、一回の実際の読込み操作に対していくつかのディレクトリエントリをバッファリングしてよい。

【戻値】

readdir_r() は、成功した場合 0 を返す。それ以外の場合エラー番号を返す。

【エラー】

EOVERFLOW	値を設定する構造体内のいずれかで、表現できない値が発生した
EBADF	dirp が、オープンされたディレクトリストリームを参照していない
ENOENT	ディレクトリストリームの現在位置が不正

【関連項目】

opendir_eno, closedir_eno

8.7.4 rewinddir - ディレクトリストリームの位置を先頭にリセット

【書式】

```
#include <dirent.h>

void    rewinddir(DIR *dirp);
```

【解説】

rewindir() は、dirp が参照しているディレクトリストリームの位置をディレクトリの先頭にリセットする。
dirp がディレクトリストリームを参照していなかった場合その作用は未定義とする。

【戻値】

rewindir() は、値を返さない。

【エラー】

なし

【関連項目】

seekdir, telldir

8.7.5 seekdir - ディレクトリストリームの位置を移動

【書式】

```
#include <dirent.h>

void    seekdir(DIR *dirp, long loc);
```

【解説】

seekdir() は、dirp で指定したディレクトリストリームに関して、次の readdir_r() の位置を
loc で指定した位置に設定する。loc の値は、以前に telldir() を呼び出して返った値でなければならない。
新しい位置は、以前に telldir() を実行したときの位置となる。

【戻値】

seekdir() は、値を返さない。

【エラー】

なし

【関連項目】

readdir, rewinddir, telldir

8.7.6 telldir - ディレクトリストリームの現在位置

【書式】

```
#include <dirent.h>

long    telldir(DIR *dirp);
```

【解説】

telldir() は、dirp で指定したディレクトリストリームの現在位置を返す。
dirp が不正の場合の動作は未定義である。

【戻値】

telldir() は、指定したディレクトリストリームの現在位置を返す。

【エラー】

なし

【関連項目】

seekdir, readdir, rewinddir

8.8 errno.h

ヘッダ errno.h は、以下のエラー処理のための型やエラー番号に関するマクロを定義する。
エラー番号については、8.2 節を参照すること。

型

```
typedef int errno_t;
```
エラー番号の型

マクロ定義

```
#define EC_ERRNO          (-73)
```
エラー番号に対応する T2EX メインエラーコード

```
#define ERRNOtoER(eno)    (ERCD(EC_ERRNO, (eno)))
```
エラー番号を、T2EX 拡張エラーコードに変換するマクロ

```
#define ERRNO(er)         (MERCD(er) == EC_ERRNO ? SERCD(er) : 0)
```
T2EX エラーコードから、エラー番号を取り出すマクロ

エラー番号

エラー番号とその意味の一覧を示す。エラーの意味に関する詳細は、そのエラー番号を返す各APIコールの説明を参照すること。

EPERM	操作は許されていない。
ENOENT	そのようなファイルやディレクトリは存在しない。
ESRCH	そのようなプロセスは存在しない。※1
EINTR	関数が割り込まれた。
EIO	I/Oエラー。
ENXIO	デバイスが準備されていない。
E2BIG	引数リストが長すぎる。
ENOEXEC	実行ファイルの形式エラー。
EBADF	不正なファイルディスクリプタ。
ECHILD	子プロセスが存在しない。※1
EAGAIN	リソースが利用できないので、再試行せよ。(EWOULDBLOCK と同じ値でよい)
EDEADLK	リソースデッドロックに陥る可能性がある。
ENOMEM	メモリが不足した。
EACCES	許可要求が拒否された。
EFAULT	不正なアドレス。
EBUSY	デバイスまたはリソースがビジー状態。
EEXIST	ファイルがすでに存在する。
EXDEV	デバイスをまたぐリンクである。
ENODEV	そのようなデバイスは無い。
ENOTDIR	ディレクトリではない。
EISDIR	ディレクトリである。
EINVAL	不正な引数。
ENFILE	オープンしているファイルが多すぎる。
EMFILE	プロセス内でオープンしているファイルが多すぎる。※1
ENOTTY	不適切な ioctl 操作である。
EFBIG	ファイルが大きすぎる。
ENOSPC	デバイスの空き容量不足。
ESPIPE	不正なシークである。
EROFS	読込み専用ファイルシステムである。
EMLINK	ハードリンクが多すぎる。※1
EPIPE	パイプが破壊されている。※1
EDOM	数値引数が、定義域の範囲外にある。
ERANGE	結果が大きすぎる。
EWOULDBLOCK	操作は待ちに入る。(EAGAIN と同じ値でよい)
EINPROGRESS	操作は処理中である。
EALREADY	操作はすでに進行中である。
ENOTSOCK	ソケットではない。
EDESTADDRREQ	宛先アドレスが要求されている。
EMSGSIZE	メッセージが長すぎる。
EPROTOTYPE	ソケットに対しプロトコルのタイプが間違っている。
ENOPROTOOPT	プロトコルが利用できない。
EPROTONOSUPPORT	プロトコルがサポートされていない。
ESOCKTNOSUPPORT	ソケットタイプがサポートされていない。
EOPNOTSUPP	操作がサポートされていない。
EPFNOSUPPORT	プロトコルファミリはサポートされていない。
EAFNOSUPPORT	アドレスファミリはサポートされていない。

EADDRINUSE	アドレスは使用中である。
EADDRNOTAVAIL	アドレスは利用できない。
ENETDOWN	ネットワークがダウンしている。
ENETUNREACH	ネットワークに到達できない。
ENETRESET	ネットワークにより、接続が切断された。
ECONNABORTED	接続が中断された。(自ホスト側の問題)
ECONNRESET	接続がリセットされた。(接続相手によるリセット)
ENOBUFS	利用可能なバッファが無い。
EISCONN	ソケットはすでに接続されている。
ENOTCONN	ソケットは接続されていない。
ESHUTDOWN	ソケットはシャットダウンされている。
ETIMEDOUT	操作がタイムアウトした。
ECONNREFUSED	接続が拒絶された。(接続相手が拒否した)
ELOOP	シンボリックリンクのレベルが多すぎる。※これは発生しない
ENAMETOOLONG	ファイル名が長すぎる。
EHOSTDOWN	宛先ホストがダウンしている。
EHOSTUNREACH	ホストに到達できない。
ENOTEMPTY	ディレクトリが空ではない。
EDQUOT	ディスククォータを超過した。※1
ENOLCK	ロックが利用できない。※1
ENOSYS	関数が実装されていない。
EOVERFLOW	指定された型で表現できない値が発生した
EFTYPE	ファイルの型または形式が不適切である。
EILSEQ	不正なバイトシーケンス。
ENOTSUP	サポートされていない。

※1 POSIX との互換性のために、T2EXリファレンス実装では実際には発生しないエラーに関しても、エラー番号のマクロ名を定義する。
(ファイルシステム実装部を独自に実装した場合、それらの中から適切なものを使用することができる。)

以下のエラー番号は、ネットワーク通信機能のみで使用される。

EAI_AGAIN	名前解決が一時的に失敗した。
EAI_BADFLAGS	ai_flags の値が不正。
EAI_FAIL	名前解決時に回復不可能なエラーが発生した。
EAI_FAMILY	不正なアドレスファミリ。
EAI_MEMORY	メモリ割当に失敗した。
EAI_NODATA	ホスト名に関連付けられたアドレスはない。
EAI_NONAME	ホスト名とサービス名の両方が与えられない。または名前が見つからない。
EAI_SERVICE	サービスは指定されたソケットタイプでサポートされていない。
EAI_SOCKTYPE	ソケットタイプはサポートされていない
EAI_SYSTEM	内部エラーが発生した。
EAI_BADHINTS	hints の値が不正である。
EAI_OVERFLOW	引数のバッファがオーバーフローした。

エラー番号に相当する拡張エラーコード(EX_...)を以下のように定義する。

```
#define EX_PERM                 ERCD(EC_ERRNO, EPERM)
#define EX_NOENT                ERCD(EC_ERRNO, ENOENT)
#define EX_SRCH                 ERCD(EC_ERRNO, ESRCH)
#define EX_INTR                 ERCD(EC_ERRNO, EINTR)
#define EX_IO                   ERCD(EC_ERRNO, EIO)
#define EX_NXIO                 ERCD(EC_ERRNO, ENXIO)
#define EX_2BIG                 ERCD(EC_ERRNO, E2BIG)
#define EX_NOEXEC               ERCD(EC_ERRNO, ENOEXEC)
#define EX_BADF                 ERCD(EC_ERRNO, EBADF)
#define EX_CHILD                ERCD(EC_ERRNO, ECHILD)
#define EX_AGAIN                ERCD(EC_ERRNO, EAGAIN)
#define EX_DEADLK               ERCD(EC_ERRNO, EDEADLK)
#define EX_NOMEM                ERCD(EC_ERRNO, ENOMEM)
#define EX_ACCES                ERCD(EC_ERRNO, EACCES)
#define EX_FAULT                ERCD(EC_ERRNO, EFAULT)
#define EX_BUSY                 ERCD(EC_ERRNO, EBUSY)
#define EX_EXIST                ERCD(EC_ERRNO, EEXIST)
#define EX_XDEV                 ERCD(EC_ERRNO, EXDEV)
#define EX_NODEV                ERCD(EC_ERRNO, ENODEV)
#define EX_NOTDIR               ERCD(EC_ERRNO, ENOTDIR)
#define EX_ISDIR                ERCD(EC_ERRNO, EISDIR)
#define EX_INVAL                ERCD(EC_ERRNO, EINVAL)
#define EX_NFILE                ERCD(EC_ERRNO, ENFILE)
#define EX_MFILE                ERCD(EC_ERRNO, EMFILE)
#define EX_NOTTY                ERCD(EC_ERRNO, ENOTTY)
#define EX_FBIG                 ERCD(EC_ERRNO, EFBIG)
```

```
#define EX_NOSPC                ERCD(EC_ERRNO, ENOSPC)
#define EX_SPIPE                ERCD(EC_ERRNO, ESPIPE)
#define EX_ROFS                 ERCD(EC_ERRNO, EROFS)
#define EX_MLINK                ERCD(EC_ERRNO, EMLINK)
#define EX_PIPE                 ERCD(EC_ERRNO, EPIPE)
#define EX_DOM                  ERCD(EC_ERRNO, EDOM)
#define EX_RANGE                ERCD(EC_ERRNO, ERANGE)
#define EX_WOULDBLOCK           ERCD(EC_ERRNO, EWOULDBLOCK)
#define EX_INPROGRESS           ERCD(EC_ERRNO, EINPROGRESS)
#define EX_ALREADY              ERCD(EC_ERRNO, EALREADY)
#define EX_NOTSOCK              ERCD(EC_ERRNO, ENOTSOCK)
#define EX_DESTADDRREQ          ERCD(EC_ERRNO, EDESTADDRREQ)
#define EX_MSGSIZE              ERCD(EC_ERRNO, EMSGSIZE)
#define EX_PROTOTYPE            ERCD(EC_ERRNO, EPROTOTYPE)
#define EX_NOPROTOOPT           ERCD(EC_ERRNO, ENOPROTOOPT)
#define EX_PROTONOSUPPORT       ERCD(EC_ERRNO, EPROTONOSUPPORT)
#define EX_SOCKTNOSUPPORT       ERCD(EC_ERRNO, ESOCKTNOSUPPORT)
#define EX_OPNOTSUPP            ERCD(EC_ERRNO, EOPNOTSUPP)
#define EX_PFNOSUPPORT          ERCD(EC_ERRNO, EPFNOSUPPORT)
#define EX_AFNOSUPPORT          ERCD(EC_ERRNO, EAFNOSUPPORT)
#define EX_ADDRINUSE            ERCD(EC_ERRNO, EADDRINUSE)
#define EX_ADDRNOTAVAIL         ERCD(EC_ERRNO, EADDRNOTAVAIL)
#define EX_NETDOWN              ERCD(EC_ERRNO, ENETDOWN)
#define EX_NETUNREACH           ERCD(EC_ERRNO, ENETUNREACH)
#define EX_NETRESET             ERCD(EC_ERRNO, ENETRESET)
#define EX_CONNABORTED          ERCD(EC_ERRNO, ECONNABORTED)
#define EX_CONNRESET            ERCD(EC_ERRNO, ECONNRESET)
#define EX_NOBUFS               ERCD(EC_ERRNO, ENOBUFS)
#define EX_ISCONN               ERCD(EC_ERRNO, EISCONN)
#define EX_NOTCONN              ERCD(EC_ERRNO, ENOTCONN)
#define EX_SHUTDOWN             ERCD(EC_ERRNO, ESHUTDOWN)
#define EX_TIMEDOUT             ERCD(EC_ERRNO, ETIMEDOUT)
#define EX_CONNREFUSED          ERCD(EC_ERRNO, ECONNREFUSED)
#define EX_LOOP                 ERCD(EC_ERRNO, ELOOP)
#define EX_NAMETOOLONG          ERCD(EC_ERRNO, ENAMETOOLONG)
#define EX_HOSTDOWN             ERCD(EC_ERRNO, EHOSTDOWN)
#define EX_HOSTUNREACH          ERCD(EC_ERRNO, EHOSTUNREACH)
#define EX_NOTEMPTY             ERCD(EC_ERRNO, ENOTEMPTY)
#define EX_DQUOT                ERCD(EC_ERRNO, EDQUOT)
#define EX_NOLCK                ERCD(EC_ERRNO, ENOLCK)
#define EX_OVERFLOW             ERCD(EC_ERRNO, EOVERFLOW)
#define EX_NOSYS                ERCD(EC_ERRNO, ENOSYS)
#define EX_FTYPE                ERCD(EC_ERRNO, EFTYPE)
#define EX_ILSEQ                ERCD(EC_ERRNO, EILSEQ)
#define EX_NOTSUP               ERCD(EC_ERRNO, ENOTSUP)
#define EX_AI_ADDRFAMILY        ERCD(EC_ERRNO, EAI_ADDRFAMILY)
#define EX_AI_AGAIN             ERCD(EC_ERRNO, EAI_AGAIN)
#define EX_AI_BADFLAGS          ERCD(EC_ERRNO, EAI_BADFLAGS)
#define EX_AI_FAIL              ERCD(EC_ERRNO, EAI_FAIL)
#define EX_AI_FAMILY            ERCD(EC_ERRNO, EAI_FAMILY)
#define EX_AI_MEMORY            ERCD(EC_ERRNO, EAI_MEMORY)
#define EX_AI_NODATA            ERCD(EC_ERRNO, EAI_NODATA)
#define EX_AI_NONAME            ERCD(EC_ERRNO, EAI_NONAME)
#define EX_AI_SERVICE           ERCD(EC_ERRNO, EAI_SERVICE)
#define EX_AI_SOCKTYPE          ERCD(EC_ERRNO, EAI_SOCKTYPE)
#define EX_AI_SYSTEM            ERCD(EC_ERRNO, EAI_SYSTEM)
#define EX_AI_BADHINTS          ERCD(EC_ERRNO, EAI_BADHINTS)
#define EX_AI_PROTOCOL          ERCD(EC_ERRNO, EAI_PROTOCOL)
#define EX_AI_OVERFLOW          ERCD(EC_ERRNO, EAI_OVERFLOW)
```

8.9 float.h

ヘッダ float.h は、以下の浮動小数点に関する属性や制限値などのマクロを定義する。

マクロ

FLT_ROUNDS
浮動小数点加算の丸めモードを示す。これは非定数でもよい。

-1	不確定
0	ゼロ方向へ
1	最も近い値へ
2	正の無限大方向へ
3	負の無限大方向へ

そのほかの値は、非標準の実装定義の丸め方とする。
T2EXリファレンス実装では、FLT_ROUNDS は 1 である。

FLT_EVAL_METHOD
浮動小数点オペランドの演算結果の値、通常の算術型変換の結果の値、および浮動小数点定数の値を、その型に要求されるものよりも大きな範囲と精度を持つ形式で評価してもよいかどうかを表す。
FLT_EVAL_METHOD は、実装が定義する評価方法を与える。

-1	不確定
0	全ての演算と定数を、その型の範囲と精度で評価する。
1	float および double型の演算と定数を、double型の範囲と精度で評価する。 long double 型の演算と定数を、long double 型の範囲と精度で評価する。
2	全ての演算と定数を、long double 型の範囲と精度で評価する。

その他の値は、実装定義の動作とする。
T2EXリファレンス実装では、FLT_EVAL_METHOD は 0 である。

以下に列挙する値は、右側に示された値以上の絶対値と、同じ符号を持つ実装定義の定数式でなければならない。

FLT_RADIX 2
指数部を表現する基数

FLT_MANT_DIG
DBL_MANT_DIG
LDBL_MANT_DIG
浮動小数点の数字部の(FLT_RADIXを基数とする)有効桁数
T2EXリファレンス実装では、FLT_MANT_DIG は 24、DBL_MANT_DIG と LDBL_MANT_DIG は 53 となっている。

DECIMAL_DIG 10
サポートしている最大幅の浮動小数点型の数値を、一旦 n 桁の10進数に丸め、元に戻したとき変化が発生しない最小の n の値
T2EXリファレンス実装では、17 となっている。

FLT_DIG 6
DBL_DIG 10
LDBL_DIG 10
この桁数の10進数は、それぞれ float, double, long double 型に丸めることができ、元に戻したとき変化が発生しない。
T2EXリファレンス実装では、FLT_DIG は 6、DBL_DIG と LDBL_DIG は 15 となっている。

FLT_MIN_EXP
DBL_MIN_EXP
LDBL_MIN_EXP
FLT_RADIX をその値から 1 減算した値でべき乗したとき，正規化された浮動小数点数となる最小の負の整数
T2EXリファレンス実装では、FLT_MIN_EXP は -125、DBL_MIN_EXP と LDBL_MIN_EXP は -1021 となっている。

FLT_MIN_10_EXP -37
DBL_MIN_10_EXP -37
LDBL_MIN_10_EXP -37
10をその値でべき乗したとき，正規化された浮動小数点数の範囲内の値となる最小の負の整数
T2EXリファレンス実装では、FLT_MIN_10_EXP は -37、DBL_MIN_10_EXP と LDBL_MIN_10_EXP は -307 となっている。

FLT_MAX_EXP
DBL_MAX_EXP

LDBL_MAX_EXP
FLT_RADIX をその値から 1 減算した値でべき乗したとき，
表現可能な有限の浮動小数点数となる最大の整数
T2EXリファレンス実装では、FLT_MAX_EXP は 128、DBL_MAX_EXP と LDBL_MAX_EXP は 1024 となっている。

FLT_MAX_10_EXP +37
DBL_MAX_10_EXP +37
LDBL_MAX_10_EXP +37
10をその値でべき乗したとき，表現可能な有限の浮動小数点数の範囲内の値となる最大の整数
T2EXリファレンス実装では、FLT_MAX_10_EXP は 38、DBL_MAX_10_EXP と LDBL_MAX_10_EXP は 308 となっている。

以下に列挙する値は，右側に示す値以上の実装定義の値をもつ定数式でなければならない。

FLT_MAX 1E+37
DBL_MAX 1E+37
LDBL_MAX 1E+37
表現可能な最大の有限浮動小数点数
T2EXリファレンス実装では、以下の値となっている。
FLT_MAX 3.40282347e+38F
DBL_MAX 1.7976931348623157e+308
LDB_MAX 1.7976931348623157e+308L

以下に列挙する値は，右側に示すある値以下の実装定義の(正の)値をもつ定数式でなければならない。

FLT_MIN 1E-37
DBL_MIN 1E-37
LDBL_MIN 1E-37
最小の正規化された正の浮動小数点数
T2EXリファレンス実装では、以下の値となっている。
FLT_MIN 1.17549435e-38F
DBL_MIN 2.2250738585072014e-308
LDBL_MIN 2.2250738585072014e-308L

FLT_EPSILON 1E-5
DBL_EPSILON 1E-9
LDBL_EPSILON 1E-9
与えられた浮動小数点型で表現可能な 1 より大きい最小の値と 1 との差
T2EXリファレンス実装では、以下の値となっている。
FLT_EPSILON 1.19209290e-7F
DBL_EPSILON 2.2204460492503131e-16
LDBL_EPSILON 2.2204460492503131e-16L

8.10 inttypes.h

ヘッダ inttypes.h は、固定サイズの整数の扱いに関連した型や関数およびマクロを定義する。
ヘッダ inttypes.h は、その内部で〈stdint.h〉をインクルードしている。

型

imaxdiv_t
　　imaxdiv() が返す値を格納する構造体型

マクロ

以下のマクロは、対応する整数型を変換する時、書式付入出力関数の format で使うためのものである。
これらのマクロは、それぞれが変換指示子を含み、場合によっては長さ修飾子で修飾される文字列リテラルに展開される。

これらのマクロの一般的な形式は、以下のいずれかで始まる:
・PRI(fprintf() 系列の関数に対する文字列リテラル)
・SCN(fscanf() 系列の関数に対する文字列リテラル)

次に変換指示子(d,i,o,u,x,X)が続き、さらに stdint.h で規定されるのと同様な型名に対応する名前が続く。

これらの名前において、N は stdint.h で述べる型の幅を表す。
例えば PRIdFAST32 は、int_fast32_t 型の整数値を出力するための書式文字列として使うことが出来る。

符号付き整数に対する fprintf() 用マクロ:

PRIdN	PRIdLEASTN	PRIdFASTN	PRIdMAX	PRIdPTR
PRIiN	PRIiLEASTN	PRIiFASTN	PRIiMAX	PRIiPTR

符号無し整数に対する fprintf() 用マクロ:

PRIoN	PRIoLEASTN	PRIoFASTN	PRIoMAX	PRIoPTR
PRIuN	PRIuLEASTN	PRIuFASTN	PRIuMAX	PRIuPTR
PRIxN	PRIxLEASTN	PRIxFASTN	PRIxMAX	PRIxPTR
PRIXN	PRIXLEASTN	PRIXFASTN	PRIXMAX	PRIXPTR

符号付き整数に対する fscanf() 用マクロ:

SCNdN	SCNdLEASTN	SCNdFASTN	SCNdMAX	SCNdPTR
SCNiN	SCNiLEASTN	SCNiFASTN	SCNiMAX	SCNiPTR

符号無し整数に対する fscanf() 用マクロ:

SCNoN	SCNoLEASTN	SCNoFASTN	SCNoMAX	SCNoPTR
SCNuN	SCNuLEASTN	SCNuFASTN	SCNuMAX	SCNuPTR
SCNxN	SCNxLEASTN	SCNxFASTN	SCNxMAX	SCNxPTR

処理系は、stdint.h が提供する各型に対し, 対応する fprintf 用マクロを定義する必要がある。
また、処理系がその型に対する適切な長さ修飾子を持ってるならば、対応する fscanf 用マクロを定義する必要がある。

関数(マクロとして定義してもよい)

8.10.1 imaxabs - 整数の絶対値

【書式】

```
#include 〈inttypes.h〉

intmax_t        imaxabs(intmax_t j);
```

【解説】

imaxabs() は、整数 j の絶対値を求める。結果が表現できない場合、動作は不定とする。

【戻値】

imaxabs() は、絶対値を返す。

【エラー】

なし

【関連項目】

imaxdiv

8.10.2 imaxdiv - 整数の商と余り

【書式】

```
#include <inttypes.h>

imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);
```

【解説】

imaxdiv() は、numer / denom と number % denom を単一の操作で求める。

【戻値】

imbxdiv() は、商と余りを格納した imaxdiv_t 型の構造体を返す。
その構造体は、各々が intmax_t 型の quot(商) および rem(余り) というメンバを(順序は関係なく)含む必要がある。
結果のどちらかが表現できない場合、動作は不定とする。

【エラー】

なし

【関連項目】

imaxabs

8.10.3 strtoimax, strtoumax - 文字列を整数型に変換

【書式】

```
#include <inttypes.h>

intmax_t        strtoimax(const char *nptr, char **endptr, int base);
uintmax_t       strtoumax(const char *nptr, char **endptr, int base);
```

【解説】

これらの関数は、変換結果の型が intmax_t および unitmax_t であることを除き
strtol(), strtoll(), strtoul() および strtoull() と等価である。

【戻値】

これらの関数は、変換が行われれば変換された値を返す。変換が行われなかった場合 0 を返す。
結果が表現できる値の範囲を超える場合、値の符号と型に応じて INTMAX_MAX, INTMAX_MIN, または UINTMAX_MAX を返す。

【エラー】

なし

【関連項目】

strtol, strtoul

8.11 iso646.h

ヘッダ iso646.h は、演算子に関する代替の記述を提供する以下のマクロを定義する。

マクロ

各マクロは、右側に示す内容に展開される。

and	&&
and_eq	&=
bitand	&
bitor	|
compl	~
not	!
not_eq	!=
or	||
or_eq	|=
xor	^
xor_eq	^=

8.12 limits.h

ヘッダ limits.h は、各種の制限に関するマクロを定義する。

定数

数値の制限値:
右側に示す値が正の場合はその値以下、負の場合はその値以上でなければならない。
char 型の値が符号付き整数として扱われる場合、CHAR_MIN の値は SCHAR_MIN と、CHAR_MAX の値は、SCHAR_MAX と同じになる。そうでない場合は CHAR_MIN の値は 0 で、CHAR_MAX の値は、UCHAR_MAX と同じになる。

CHAR_BIT　　8
char 型のビット数

CHAR_MAX　　UCHAR_MAX または SCHAR_MAX
char 型オブジェクトの最大値

CHAR_MIN　　0 または SCHAR_MIN
char 型オブジェクトの最小値

INT_MAX　　+2147483647
int 型オブジェクトの最大値

INT_MIN　　-2147483647
int 型オブジェクトの最小値

LLONG_MAX　　+9223372036854775807
long long 型オブジェクトの最大値

LLONG_MIN　　-9223372036854775807
long long 型オブジェクトの最小値

LONG_BIT　　32
long 型オブジェクトのビット数

LONG_MAX　　+2147483647
long 型オブジェクトの最大値

LONG_MIN　　-2147483647
long 型オブジェクトの最小値

MB_LEN_MAX　　1
システムロケールにおける一文字の最大バイト数

SCHAR_MAX　　+127
signed char 型オブジェクトの最大値

SCHAR_MIN　　-128
signed char 型オブジェクトの最小値

SHRT_MAX　　+32767
short 型オブジェクトの最大値

SHRT_MIN　　-32767
short 型オブジェクトの最小値

SSIZE_MAX　　32767
ssize_t 型オブジェクトの最大値

UCHAR_MAX　　255
unsigned char 型オブジェクトの最大値

UINT_MAX　　4294967295
unsigned int 型オブジェクトの最大値

ULLONG_MAX　　18446744073709551615
unsigned long long 型オブジェクトの最大値

ULONG_MAX　　4294967295
unsigned long 型オブジェクトの最大値

USHRT_MAX　　65535

unsigned short 型オブジェクトの最大値

WORD_BIT　　32
int 型オブジェクトのビット数

パス名に関する制限値
右側に示す値以上でなければならない。

NAME_MAX　　255
(最後のヌルを含まない)ファイル名の最大バイト数

PATH_MAX　　1024
ユーザが提供するバッファにパス名を格納するのに必要な最後のヌル文字まで含めたバイト数

8.13 math.h

ヘッダ math.h は、数学的な事項に関して以下のマクロや関数プロトタイプ宣言を定義する。

マクロ

```
int fpclassify(浮動小数点型 x);
```

fpclassify マクロは、x の値を NaN、無限大、正規化数、非正規化数、0、または別の実装定義のカテゴリに分類する。
fpclassify は、下で述べる数値分類マクロ値を返す。
T2EX リファレンス実装では、それ以外のカテゴリは定義しない。

```
int isfinite(浮動小数点型 x);
```

isfinite マクロは、x が有限の値(0, 非正規化数, 又は正規化数であって, 無限大や NaNではない)かどうかを判定する。
isfinite マクロは、x が有限の値を持つ場合は 0 以外の値を返す。

```
int isinf(浮動小数点型 x);
```

isinf マクロは、x が無限大(正または負)かどうか判定する。
isinf マクロは、x が無限大の場合は 0 以外の値を返す。

```
int isnan(浮動小数点型 x);
```

isnan マクロは、x の値が NaN かどうか判定する。
isnan マクロは、x が NaN の値を持つ場合は 0 以外の値を返す。

```
int isnormal(浮動小数点型 x);
```

isnormal マクロは、x の値が正規化されている(ゼロ、非正規化、無限大、NaN のいずれでもない)か判定する。
isnormal マクロは、x が正規化された値を持つ場合は 0 以外の値を返す。

```
int signbit(浮動小数点型 x);
```

signbit マクロは、x の符号が負かどうか判定する。
NaN, ゼロ、および無限大であっても、符号ビットは持っている。
signbit マクロは、x の符号が負の場合 0 以外の値を返す。

```
int isgreater(浮動小数点型 x, 浮動小数点型 y);
```

isgreater マクロは、x が y より大きいかどうか判定する。
isgreater(x, y) の値は、(x) > (y) と等しくなければならないが、(x) > (y) と異なり x, y が順序付けできない場合でも浮動小数点例外を発生しない。
isgreater() マクロは、(x) > (y) の値を返す。x または y が NaN の場合 0 を返す。

```
int isgreaterequal(浮動小数点型 x, 浮動小数点型 y);
```

isgreaterequal マクロは、x が y と等しいか、より大きいかどうか判定する。
isgreaterequal(x, y) の値は、(x) >= (y) と等しくなければならないが、(x) >= (y) と異なりx, y が順序付けできない場合でも浮動小数点例外を発生しない。
isgreaterqual マクロは、(x) >= (y) の値を返す。x または y が NaN の場合 0 を返す。

```
int isless(浮動小数点型 x, 浮動小数点型 y);
```

isless マクロは、x が y より小さいかどうか判定する。
isless(x, y) の値は、(x) < (y) と等しくなければならないが、(x) < (y) と異なり x, y が順序付けできない場合でも浮動小数点例外を発生しない。
isless() マクロは (x) < (y) の値を返す。x または y が NaN の場合 0 を返す。

```
int islessequal(浮動小数点型 x, 浮動小数点型 y);
```

islessequal マクロは、x が y と等しいか、より小さいかどうか判定する。
islessequal(x, y) の値は、(x) <= (y) と等しくなければならないが、(x) <= (y) と異なり x, y が順序付けできない場合でも浮動小数点例外を発生しない。
islessequal() マクロは (x) <= (y) の値を返す。x または y が NaN の場合 0 を返す。

```
int islessgreater(浮動小数点型 x, 浮動小数点型 y);
```

islessgreater マクロは、x が y より小さいか、より大きいかどうか判定する。

islessgreater(x, y) の値は、(x) < (y) || (x) > (y) と等しくなければならないが、x, y が順序付けできない場合でも浮動小数点例外を発生しない。
islessgreater() マクロは (x) < (y) || (x) > (y) の値を返す。
x または y が NaN の場合 0 を返す。

int isunordered(浮動小数点型 x, 浮動小数点型 y);

isunordered マクロは、x と y が順序付けられないかどうか判定する。
isunordered マクロは、x と y が順序付けられない場合は 1 を返し、
それ以外の場合は 0 を返す。
x または y が NaN の場合は 1 が返る。

数値分類マクロ値

数値分類マクロ値は、浮動小数点数値の種類を表す。
これらは、それぞれ異なる値の定数値に展開される。
T2EXリファレンス実装では以下を定義している。
実装独自に他の浮動小数点の分類定義を追加しても良い。
ただし、マクロ名は FP_ で始まり、大文字からなる文字列とすること。

FP_INFINITE	無限大(正または負) (+Inf, -Inf または±Inf と記述する)
FP_NAN	数ではない (NaN と記述する)
FP_NORMAL	正規化されている
FP_SUBNORMAL	正規化されていない
FP_ZERO	0 (+0 または -0)

以下のマクロは、ilogb(x) の戻値を表す整数定数に展開される。

FP_ILOGB0	ilogb(x) で、x が 0 の場合返す値。 INT_MIN か -INT_MAX のどちらかでなければならない。
FP_ILOGBNAN	ilogb(x) で、x が NaN の場合返す値。 INT_MAX か INT_MIN のどちらかでなければならない。

数学定数マクロ値

以下の数値定数を定義する。
値は、double 型で、double 型の精度を持つ。

M_E	自然対数の底 e の値
M_LOG2E	log2(e) の値
M_LOG10E	log10(e) の値
M_LN2	loge(2) の値
M_LN10	loge(10) の値
M_PI	円周率 π の値
M_PI_2	π/2 の値
M_PI_4	π/4 の値
M_1_PI	1/π の値
M_2_PI	2/π の値
M_2_SQRTPI	2/sqrt(π) の値
M_SQRT2	sqrt(2) の値
M_SQRT1_2	1/sqrt(2) の値
MAXFLOAT	float.h で定義される FLT_MAX と同じ

特殊数値定数マクロ値

以下のマクロを定義する。

HUGE_VAL	正の double 型定数で、float 型として表現できない値。 数学ライブラリが返すエラー値として用いる。
HUGE_VALF	正の float 型定数。 数学ライブラリが返すエラー値として用いる。
HUGE_VALL	正の long double 型定数。 数学ライブラリが返すエラー値として用いる。
INFINITY	正または符号なしの無限大を表す float 型の定数式。 それが表現できないシステムでは、変換時にオーバーフローを起こす float 型の正の定数。
NAN	

quiet NaN を表す float 型の定数式。このマクロは、実装が
float 型の quiet NaN をサポートしている場合のみ定義される。
quiet NaN とは、それが演算の対象となったとき、演算時に
いかなる例外も発生せずに演算結果として出力されるものである。
T2EX では、原則として NaN をサポートし、NaN はすべて
quiet NaN とする。
ただし、NaN をサポートしない実装も許容する。

関数

8.13.1 acos, acosf, acosl - 逆余弦

【書式】

```
#include <math.h>

double          acos(double x);
float           acosf(float x);
long double     acosl(long double x);
```

【解説】

これらの関数は、 x の逆余弦の主値を計算する。x は [-1,1] の範囲でなければならない。

【戻値】

成功した場合、これらの関数は、x の逆余弦を [0,π] の範囲で返す。
x が [-1,1] の範囲外の場合、NaN を返す。
x が NaN の場合、NaN を返す。
x が +1 の場合、+0 を返す。
x が ±Inf の場合、NaN を返す。

【エラー】

なし

【関連項目】

cos

8.13.2 acosh, acoshf, acoshl - 双曲線逆余弦

【書式】

```
#include <math.h>

double          acosh(double x);
float           acoshf(float x);
long double     acoshl(long double x);
```

【解説】

これらの関数は、 x の双曲線逆余弦を計算する。

【戻値】

成功した場合、これらの関数は、x の双曲線逆余弦を返す。
x < 1 の場合、NaN を返す。
x が NaN の場合、NaN を返す。
x が +1 の場合、+0 を返す。

x が +Inf の場合、+Inf を返す。
x が -Inf の場合、NaN を返す。

【エラー】

なし

【関連項目】

cosh

8.13.3 asin, asinf, asinl - 逆正弦

【書式】

```
#include <math.h>

double          asin(double x);
float           asinf(float x);
long double     asinl(long double x);
```

【解説】

これらの関数は、 x の逆正弦の主値を計算する。x は [-1,1] の範囲でなければならない。

【戻値】

成功した場合、これらの関数は x の逆正弦を [-π/2,π/2] の範囲で返す。
x が [-1,1] の範囲外の場合、NaN を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、x を返す。
x が ±Inf の場合、NaN を返す。
x が非正規化数の場合、x を返す。

【エラー】

なし

【関連項目】

sin

8.13.4 asinh, asinhf, asinhl - 双曲線逆正弦

【書式】

```
#include <math.h>

double          asinh(double x);
float           asinhf(float x);
long double     asinhl(long double x);
```

【解説】

これらの関数は、 x の双曲線逆正弦を計算する。

【戻値】

成功した場合、これらの関数は、 x の双曲線逆正弦を返す。
x が NaN の場合、NaN を返す。
それ以外の場合は、x を返す。

【エラー】

なし

【関連項目】

sinh

8.13.5 atan, atanf, atanl - 逆正接

【書式】

```
#include <math.h>

double          atan(double x);
float           atanf(float x);
long double     atanl(long double x);
```

【解説】

これらの関数は、 x の逆正接の主値を計算する。

【戻値】

成功した場合、これらの関数は、x の逆正接を [-π/2,π/2] の範囲で返す。
x が NaN の場合、NaN を返す。
x が ±Inf の場合、それぞれ ±π/2 を返す。
それ以外の場合は、x を返す。

【エラー】

なし

【関連項目】

tan

8.13.6 atan2, ata2f, atan2l - 逆正接

【書式】

```
#include <math.h>

double          atan2(double y, double x);
float           atan2f(float y, float x);
long double     atan2l(long double y, long double x);
```

【解説】

これらの関数は、y と x の符号を戻値の象限の決定に用いて、y/x の逆正接の主値を計算する。

【戻値】

成功した場合、これらの関数は、y/x の逆正接を [-π,π]の範囲で返す。
y が ±0 で x < 0 の場合、±π を返す。
y が ±0 で x > 0 の場合、±0 を返す。
y < 0 で x が±0 の場合、-π/2 を返す。
y > 0 で x が±0 の場合、π/2 を返す。
x または y が NaN の場合、NaN を返す。
結果がアンダーフローの場合、y/x を返す。
y が ±0 で x が -0 の場合 ±π を返す。
y が ±0 で x is +0 の場合 ±0 返す。
y が有限で ± y > 0 の場合、x が -Inf ならば、±πを返す。

y が有限で ± y > 0 の場合、x が +Inf ならば、±0 を返す。
x が有限で y が ±Inf の場合、±π/2 を返す。
y が ±Inf で x が -Inf の場合、 ±3*π/4 を返す。
y が ±Inf で x が +Inf の場合、±π/4 を返す。

【エラー】

なし

【関連項目】

tan, atan

8.13.7 atanh, atanhf, atanhl - 双曲線逆正接

【書式】

```
#include <math.h>

double          atanh(double x);
float           atanhf(float x);
long double     atanhl(long double x);
```

【解説】

これらの関数は、x の双極性逆正接を計算する。

【戻値】

成功した場合、これらの関数は、x の双極性逆正接を返す。
x が ±1 の場合、これらの関数は、それぞれ HUGE_VAL, HUGE_VALF, HUGE_VALL を数学的に正しい符号で返す。
|x| > 1 の場合、これらの関数は NaN を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、x を返す。
x が ±Inf の場合、NaN を返す。
x が非正規化数の場合、x を返す。

【エラー】

なし

【関連項目】

tanh

8.13.8 cbrt, cbrtf, cbrtl - 立方根

【書式】

```
#include <math.h>

double          cbrt(double x);
float           cbrtf(float x);
long double     cbrtl(long double x);
```

【解説】

これらの関数は、 x の立方根を計算する。

【戻値】

成功した場合、これらの関数は x の立方根を返す。
x が NaN の場合、NaN を返す。

x が ±0 または ±Inf の場合、x を返す。

【エラー】

なし

8.13.9 ceil, ceilf, ceill - x 以上の最小整数値

【書式】

```
#include <math.h>

double          ceil(double x);
float           ceilf(float x);
long double     ceill(long double x);
```

【解説】

これらの関数は、x 以上の最小の整数値を計算する。

【戻値】

成功した場合、これらの関数は x 以上の最小の整数値をその関数の戻値型で返す。
x が NaN の場合、NaN を返す。
x が ±0 または ±Inf の場合、x を返す。
オーバーフローを起こす場合、それぞれ HUGE_VAL, HUGE_VALF, HUGE_VALL を返す。

【エラー】

なし

8.13.10 copysign, copysignf, copysignl - 符号のコピー

【書式】

```
#include <math.h>

double          copysign(double x, double y);
float           copysignf(float x, float y);
long double     copysignl(long double x, long double y);
```

【解説】

これらの関数は、x の絶対値で y の符号を持つ値を生成する。

【戻値】

これらの関数は、x の絶対値で y の符号を持つ値を返す。

【エラー】

なし

【関連項目】

floor

8.13.11 cos, cosf, cosl - 余弦

【書式】

```
#include <math.h>

double          cos(double x);
float           cosf(float x);
long double     cosl(long double x);
```

【解説】

これらの関数は、ラジアン単位の x の余弦を計算する。

【戻値】

これらの関数は、 x の余弦を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、1.0 を返す。
x が ±Inf の場合、NaN を返す。

【エラー】

なし

【関連項目】

sin, tan

8.13.12 cosh, coshf, coshl - 双曲線余弦

【書式】

```
#include <math.h>

double          cosh(double x);
float           coshf(float x);
long double     coshl(long double x);
```

【解説】

これらの関数は、x の双曲線余弦を計算する。

【戻値】

これらの関数は、x の双曲線余弦を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、1.0 を返す。
x が ±Inf の場合、+Inf を返す。

【エラー】

なし

【関連項目】

sinh, tanh

8.13.13 erf, erff, erfl - 誤差関数

【書式】

```
#include <math.h>
```

```
double          erf(double x);
float           erff(float x);
long double     erfl(long double x);
```

【解説】

これらの関数は、 x の誤差関数を計算する。
誤差関数は、以下のように定義される。

```
        erf(x) = 2/sqrt(pi) * integral from 0 to x of exp(-t*t) dt
```

【戻値】

これらの関数は、誤差関数の値を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、±0 を返す。
x が ±Inf の場合、±1 を返す。
x が非正規化数の場合、2 * x / sqrt(π) を返す。

【エラー】

なし

【関連項目】

erfc

8.13.14 erfc, erfcf, erfcl - 相補誤差関数

【書式】

```
#include <math.h>

double          erfc(double x);
float           erfcf(float x);
long double     erfcl(long double x);
```

【解説】

これらの関数は、相補誤差関数 1.0 - erf(x) を計算する。

【戻値】

これらの関数は、相補誤差関数の値を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、+1 を返す。
x が -Inf の場合、+2 を返す。
x が +Inf の場合、+0 を返す。

【エラー】

なし

【関連項目】

erf

8.13.15 exp, expf, expl - 底eの指数関数

【書式】

```
#include <math.h>
```

```
double          exp(double x);
float           expf(float x);
long double     expl(long double x);
```

【解説】

これらの関数は、底e の x 乗を計算する。

【戻値】

成功した場合、これらの関数は、底e の x 乗の値を返す。
オーバーフローが発生する場合、これらの関数は、それぞれ HUGE_VAL, HUGE_VALF, HUGE_VALL を返す。
アンダーフローが発生する場合、0.0 を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、+1 を返す。
x が -Inf の場合、+0 を返す。
x が +Inf の場合、x を返す。

【エラー】

なし

【関連項目】

log

8.13.16 exp2, exp2f, exp2l - 2 の x 乗

【書式】

```
#include <math.h>

double          exp2(double x);
float           exp2f(float x);
long double     exp2l(long double x);
```

【解説】

これらの関数は、2 の x 乗を計算する。

【戻値】

成功した場合、これらの関数は、2 の x 乗の値を返す。
オーバーフローが発生する場合、これらの関数は、それぞれ HUGE_VAL, HUGE_VALF, HUGE_VALL を返す。
アンダーフローが発生する場合、0.0 を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、+1 を返す。
x が -Inf の場合、+0 を返す。
x が +Inf の場合、x を返す。

【エラー】

なし

【関連項目】

exp, log

8.13.17 expm1, expm1f, expm1l - 底eの指数関数-1

【書式】

```
#include <math.h>

double          expm1(double x);
float           expm1f(float x);
long double     expm1l(long double x);
```

【解説】

これらの関数は、底e の x 乗から 1 を引いた値を計算する。

【戻値】

成功した場合、これらの関数は、底e の x 乗から 1 を引いた値を返す。
オーバーフローが発生する場合、これらの関数は、それぞれ HUGE_VAL, HUGE_VALF, HUGE_VALL を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、±0 を返す。
x が -Inf の場合、-1 を返す。
x が +Inf の場合、x を返す。
x が非正規化数の場合、x を返す。

【エラー】

なし

【関連項目】

exp, ilogb, log1p

8.13.18 fabs, fabsf, fabsl - 浮動小数点数の絶対値

【書式】

```
#include <math.h>

double          fabs(double x);
float           fabsf(float x);
long double     fabsl(long double x);
```

【解説】

これらの関数は、 浮動小数点数 x の絶対値 |x| を計算する。

【戻値】

これらの関数は、 浮動小数点数 x の絶対値を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、+0 を返す。
x が ±Inf の場合、+Inf を返す。

【エラー】

なし

【関連項目】

isnan

8.13.19 fdim, fdimf, fdiml - 正の差分

【書式】

```
#include <math.h>
```

```
double          fdim(double x, double y);
float           fdimf(float x, float y);
long double     fdiml(long double x, long double y);
```

【解説】

これらの関数は、x と y の間の正の差分を求める。
x > y の場合、x - y を返す。
x <= y の場合、+0 を返す。

【戻値】

成功した場合、これらの関数は、x と y の間の正の差分を返す。
x-y が正でオーバーフローを起こす場合、これらの関数は、それぞれ HUGE_VAL, HUGE_VALF, HUGE_VALL を返す。
x-y が正でアンダーフローを起こす場合、0.0 を返す。
x または y が NaN の場合、NaN を返す。

【エラー】

なし

8.13.20 floor, floorf, floorl - x 以下の最大の整数値

【書式】

```
#include <math.h>

double          floor(double x);
float           floorf(float x);
long double     floorl(long double x);
```

【解説】

これらの関数は、x 以下の最大の整数値を求める。

【戻値】

これらの関数は、x 以下の最大の整数値をその関数の戻値型で返す。
x が NaN の場合、NaN を返す。
x が ±0 または ±Inf の場合、x を返す。
オーバーフローを起こす場合、これらの関数は、それぞれ -HUGE_VAL, -HUGE_VALF, -HUGE_VALL を返す。

【エラー】

なし

【関連項目】

fmax, fmin

8.13.21 fma, fmaf, fmal - 浮動小数点数の積和

【書式】

```
#include <math.h>

double          fma(double x, double y, double z);
float           fmaf(float x, float y, float z);
long double     fmal(long double x, long double y, long double z);
```

【解説】

これらの関数は、一つの三項演算として丸められた (x * y) + z を計算する。
これらの関数は、x, y, z の値が無限の精度を持つかのように演算し FLOT_ROUNDS で指定された丸めモードに従い
結果の形式に一回だけ丸める。

【戻値】

これらの関数は、一つの三項演算として丸められた (x * y) + z の値を返す。
x または y が NaN の場合、NaN を返す。
x * y が無限大で、z も逆の符号の無限大の場合、NaN を返す。
x, y の一方が無限大で、他方が 0、z が NaN でない場合、NaN を返す。
x, y の一方が無限大で、他方が 0、z が NaN の場合、NaN を返す。
x * y が 0*Inf でもInf*0 でもなく、z が NaN の場合、NaN を返す。

【エラー】

なし

8.13.22 fmax, fmaxf, fmaxl - 浮動小数点数の最大値

【書式】

```
#include <math.h>

double          fmax(double x, double y);
float           fmaxf(float x, float y);
long double     fmaxl(long double x, long double y);
```

【解説】

これらの関数は、x と y の大きい方を求める。

【戻値】

これらの関数は、x と y の大きい方を返す。
x, y の一方が NaN の場合、他方の値を返す。
x も y も NaN の場合、NaN を返す。

【エラー】

なし

【関連項目】

fdim, fmin

8.13.23 fmin, fminl, fminf - 浮動小数点数の最小値

【書式】

```
#include <math.h>

double          fmin(double x, double y);
float           fminf(float x, float y);
long double     fminl(long double x, long double y);
```

【解説】

これらの関数は、x と y の小さい方を求める。

【戻値】

これらの関数は、x と y の小さい方を返す。

x, y の一方が NaN の場合、他方の値を返す。
x も y も NaN の場合、NaN を返す。

【エラー】

なし

【関連項目】

fdim, fmax

8.13.24 fmod, fmodf, fmodl - 浮動小数点剰余

【書式】

```
#include <math.h>

double          fmod(double x, double y);
float           fmodf(float x, float y);
long double     fmodl(long double x, long double y);
```

【解説】

これらの関数は、x / y の浮動小数点の剰余を求める。

【戻値】

これらの関数は、整数 i に対し、x - i * y の値を返す。y が 0 でない場合、結果は x と同じ符号を持ち、
その絶対値は y の絶対値より小さい。
アンダーフローが発生する場合、0.0 を返す。
x または y が NaN の場合、NaN を返す。
y が 0 の場合、これらの関数は NaN を返す。
x が無限大の場合、NaN を返す。
x が±0で、y がゼロで無い時、±0を返す。
x が無限大ではなく、y が±Infの場合、x を返す。

【エラー】

なし

【関連項目】

isnan

8.13.25 frexp, frexpf, frexpl - 浮動小数点数の仮数部と指数部の取り出し

【書式】

```
#include <math.h>

double          frexp(double value, int *exp);
float           frexpf(float value, int *exp);
long double     frexpl(long double value, int *exp);
```

【解説】

これらの関数は、浮動小数点数 value を、正規化された少数部と 2 の整数べき乗に分解し、
整数の指数部を、exp が示す領域に格納する。

【戻値】

有限の引数に対し、これらの関数は、以下の二つの条件を満たす値 x を返す。
・x の絶対値が、区間 [1/2, 1) の範囲、または 0

・value は x に 2 の *exp 乗した値をかけたもの

value が 0 の場合、0 を返し、*exp も 0 となる。
value が NaN の場合、NaN を返し、*exp は不定とする。
value が±0の場合、±0 を返し、*exp の値は 0 とする。
value が±Infの場合、valueを返し、*exp の値は不定とする。

【エラー】

なし

【関連項目】

ldexp, modf

8.13.26 hypot, hypotf, hypotl - ユークリッド距離関数

【書式】

```
#include <math.h>

double          hypot(double x, double y);
float           hypotf(float x, float y);
long double     hypotl(long double x, long double y);
```

【解説】

これらの関数は、オーバーフローおよびアンダーフロー無しで sqrt(x*x + y+y) を計算する。

【戻値】

これらの関数は、直角部の二辺の長さが x と y の直角三角形の斜辺の長さを返す。
結果がオーバーフローとなる場合、これらの関数は、それぞれ HUGE_VAL, HUGE_VALF, HUGE_VALL を返す。
x または y が±Infの場合、(x, y の一方がNaNでも)±Inf を返す。
x または y がNaNで、他方が±Infでない場合、NaNを返す。

【エラー】

なし

【関連項目】

atan2, sqrt

8.13.27 ilogb, ilogbf, ilogbl - 指数の抽出

【書式】

```
#include <math.h>

int     ilogb(double x);
int     ilogbf(float x);
int     ilogbl(long double x);
```

【解説】

これらの関数は、 x の指数部を符号付き整数として返す。

【戻値】

成功した場合、これらの関数は、 x の指数部を符号付き整数として返す。
x が 0 の場合、これらの関数は FP_ILOGB0 を返す。
x が ±Inf の場合、これらの関数は INT_MAX を返す。

x が NaN の場合、これらの関数は FP_ILOGBNAN を返す。
結果が INT_MAX より大きい場合、INT_MAX を返す。
結果が INT_MIN より小さい場合、INT_MIN を返す。

【エラー】

なし

【関連項目】

logb, scalbln

8.13.28 j0, j1, jn - 第一種ベッセル関数

【書式】

```
#include <math.h>

double  j0(double x);
double  j1(double x);
double  jn(int n, double x);
```

【解説】

j0(), j1(), jn() は、それぞれ 0, 1, n 次の第一種ベッセル関数を計算する。

【戻値】

これらの関数は、それぞれ 0, 1, n 次の第一種ベッセル関数を返す。
x が大きすぎる場合、またはアンダーフローが発生する場合、0 を返す。
x が NaN の場合、NaN を返す。

【エラー】

なし

【関連項目】

y0

8.13.29 ldexp, ldexpf, ldexpl - 浮動小数点数と2のべき乗との積

【書式】

```
#include <math.h>

double          ldexp(double x, int exp);
float           ldexpf(float x, int exp);
long double     ldexpl(long double x, int exp);
```

【解説】

これらの関数は、x と 2 の exp 乗との積を計算する。

【戻値】

成功した場合、これらの関数は x と 2 の exp 乗との積を計算する。
オーバーフローが発生する場合、これらの関数は、それぞれ ±HUGE_VAL, ±HUGE_VALF, ±HUGE_VALL を返す。
アンダーフローが発生する場合、0.0 を返す。
x が NaN の場合、NaN を返す。
x が ±0 または ±Inf の場合、x を返す。

exp が 0 の場合、x を返す。

【エラー】

なし
frexp(), isnan()

8.13.30 llrint, llrintf, llrintl - 整数に丸める

【書式】

```
#include <math.h>

long long       llrint(double x);
long long       llrintf(float x);
long long       llrintl(long double x);
```

【解説】

これらの関数は、x を現在の丸め方向に応じて最も近い整数に丸める。

【戻値】

成功した場合、これらの関数は丸めた整数値を返す。
結果が正で long long で表現できない場合、戻値は不定とする。
結果が負で long long で表現できない場合、戻値は不定とする。
x がNaN、±Inf の場合、戻値は不定とする。

【エラー】

なし

【関連項目】

lrint

8.13.31 llround, llroundr, llroundl - 最も近い整数値に丸める

【書式】

```
#include <math.h>

long long       llround(double x);
long long       llroundf(float x);
long long       llroundl(long double x);
```

【解説】

これらの関数は、現在の丸め方向とは無関係にちょうど中間の場合はゼロから離れる方向で x を最も近い整数値に丸める。

【戻値】

成功した場合、これらの関数は丸めた整数値を返す。
値が正で long long で表現できない場合、戻値は不定とする。
値が負で long long で表現できない場合、戻値は不定とする。
x が NaN、+Inf または -Inf の場合、戻値は不定とする。

【エラー】

なし

【関連項目】

lround

8.13.32 log, logf, logl - 自然対数

【書式】

```
#include <math.h>

double          log(double x);
float           logf(float x);
long double     logl(long double x);
```

【解説】

これらの関数は、 x の自然対数(e を底とする対数)を計算する。

【戻値】

成功した場合、これらの関数は x の自然対数を返す。
x が ±0 の場合、これらの関数は、それぞれ -HUGE_VAL, -HUGE_VALF, -HUGE_VALL を返す。
x が負または -Inf の場合、NaN を返す。
x が NaN の場合、NaN を返す。
x が 1 の場合、0 を返す。
x が +Inf の場合、x を返す。

【エラー】

なし

【関連項目】

exp, log10, log1p

8.13.33 log10, log10f, log10l - 10を底とする対数

【書式】

```
#include <math.h>

double          log10(double x);
float           log10f(float x);
long double     log10l(long double x);
```

【解説】

これらの関数は、10 を底とする対数を計算する。

【戻値】

成功した場合、これらの関数は、x の 10 を底とする対数を返す。
x が ±0 の場合、これらの関数は、それぞれ -HUGE_VAL, -HUGE_VALF, -HUGE_VALL を返す。
x が負または -Inf の場合、NaN を返す。
x が NaN の場合、NaN を返す。
x が 1 の場合、0 を返す。
x が +Inf の場合、+Inf を返す。

【エラー】

なし

【関連項目】

log, pow

8.13.34 log1p, log1pf, log1pl - 引数に1を加えた値の自然対数

【書式】

```
#include <math.h>

double          log1p(double x);
float           log1pf(float x);
long double     log1pl(long double x);
```

【解説】

これらの関数は、log(1.0 + x) と等価である。

【戻値】

成功した場合、これらの関数は、log(1.0 + x) の値を返す。
x が -1 の場合、これらの関数は、それぞれ -HUGE_VAL, -HUGE_VALF, -HUGE_VALL を返す。
x が -1 以下、または -Inf の場合、NaN を返す。
x が NaN の場合、NaN を返す。
x が ±0 または +Inf の場合、NaN を返す。

【エラー】

なし

【関連項目】

log

8.13.35 log2, log2f, log2l - 2 を底とする対数

【書式】

```
#include <math.h>

double          log2(double x);
float           log2f(float x);
long double     log2l(long double x);
```

【解説】

これらの関数は、 x の 2 を底とする対数を計算する。

【戻値】

成功した場合、これらの関数は、 x の 2 を底とする対数を返す。
x が ±0 の場合、これらの関数は、それぞれ -HUGE_VAL, -HUGE_VALF, -HUGE_VALL を返す。
x が負または -Inf の場合、NaN を返す。
x が NaN の場合、NaN を返す。
x が 1 の場合、0 を返す。
x が +Inf の場合、+Inf を返す。

【エラー】

なし

【関連項目】

log

8.13.36 logb, lobf, lobl - 引数の指数

【書式】

```
#include <math.h>

double          logb(double x);
float           logbf(float x);
long double     logbl(long double x);
```

【解説】

これらの関数は、x の指数を符号付き浮動小数点数として計算する。
CPUの浮動小数点演算の基数(FLT_RADIXの値)を r とすると、
これは、r を底とする x の対数の整数部分を浮動小数点値で表現したものである。
x が非正規化数の場合、正規化されているものとして扱う。

【戻値】

成功した場合、これらの関数は、x の指数を返す。
x が ±0 の場合、これらの関数は、それぞれ -HUGE_VAL, -HUGE_VALF, -HUGE_VALL を返す。
x が NaN の場合、NaN を返す。
x が +Inf の場合、+Inf を返す。

【エラー】

なし

【関連項目】

ilogb, scalbln

8.13.37 lrint, lrintf, lrintl - 整数に丸める

【書式】

```
#include <math.h>

long    lrint(double x);
long    lrintf(float x);
long    lrintl(long double x);
```

【解説】

これらの関数は、x を現在の丸め方向に応じて最も近い整数に丸める。

【戻値】

成功した場合、これらの関数は、丸めた整数値を返す。
x がNaN、+Infまたは-Infの場合、戻値は不定とする。
結果が正で long で表現できない場合、戻値は不定とする。
結果が負で long で表現できない場合、戻値は不定とする。

【エラー】

なし

【関連項目】

llrint

8.13.38 lround, lroundr, lroundl - 最も近い整数値に丸める

【書式】

```
#include <math.h>

long    lround(double x);
long    lroundf(float x);
long    lroundl(long double x);
```

【解説】

これらの関数は、現在の丸め方向とは無関係にちょうど中間の場合はゼロから離れる方向で x を最も近い整数値に丸める。

【戻値】

成功した場合、これらの関数は、丸めた整数値を返す。
x が NaN、+Inf または -Inf の場合、戻値は不定とする。
値が正で long で表現できない場合、戻値は不定とする。
値が負で long で表現できない場合、戻値は不定とする。

【エラー】

なし

8.13.39 modf, modff, modfl - 浮動小数点数の分解

【書式】

```
#include <math.h>

double          modf(double value, double *iptr);
float           modff(float value, float *iptr);
long double     modfl(long double value, long double *iptr);
```

【解説】

これらの関数は、value を、それと同じ符号を持つ、整数部と小数部に分解する。
整数部を iptr が指す領域に、それぞれ dobule, float, long double 値として格納する。

【戻値】

成功した場合、これらの関数は、x の符号付き小数部を返す。
x が NaN の場合、NaN を返し、*iptr も NaN に設定する。
x が ±Inf の場合、±0 を返し、*iptr は、±Inf に設定する。

【エラー】

なし

【関連項目】

llround

8.13.40 nan, nanf, nanl - NaN を返す

【書式】

```
#include <math.h>
```

```
double          nan(const char *tagp);
float           nanf(const char *tagp);
long double     nanl(const char *tagp);
```

【解説】

nan() は、tagp が "<n文字列>" を指している場合、strtod("NAN(<n文字列>)", (char**)NULL); と等価とする。
tagp が空文字列を指している場合、strtod("NAN()", (char**)NULL) と等価とする。
tagp が上記以外を指している場合、strtod("NAN", (char**)NULL); と等価とする。

nanf(), nall() は、それぞれ同様に strtof(), stotold() 呼び出しに対応する。

<n文字列>は、NaN のバイナリ表現内の空いている場所に格納される文字列で、NaN の発生原因などを記載するのに用いる。
T2EXリファレンス実装では、NaN のバイナリ表現において、float 型で 22ビット、double および long double 型で 51ビットの空き領域があり、<n文字列>は、16進数字列として解釈し、整数値に変換してその領域に格納する。

【戻値】

これらの関数は、tagp で示された(利用可能な分の)<n文字列>の内容を持つ quiet NaN を返す。
T2EX では quiet NaN をサポートするが、quiet NaN をサポートしていないシステムでは、これらの関数は 0 を返すこと。

【エラー】

なし

【関連項目】

strtod

8.13.41 nearbyint, nearbyintf, nearbyintl - 浮動小数点丸め関数

【書式】

```
#include <math.h>

double          nearbyint(double x);
float           nearbyintf(float x);
long double     nearbyintl(long double x);
```

【解説】

これらの関数は、現在の丸め方向を用いて x を浮動小数点形式の整数に丸める。

【戻値】

成功した場合、これらの関数は、丸めた整数の値を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、±0 を返す。
x が ±Inf の場合、x を返す。

【エラー】

なし

8.13.42 nextfter, nextafterf, nextafterl nexttoward, nexttowordf, nexttowordl - 次に表現可能な値

【書式】

```
#include <math.h>
```

```
double          nextafter(double x, double y);
float           nextafterf(float x, float y);
long double     nextafterl(long double x, long double y);
double          nexttoward(double x, long double y);
float           nexttowardf(float x, long double y);
long double     nexttowardl(long double x, long double y);
```

【解説】

nextafter(), nextafterf(), および nextafterl() は、x の y 方向で次に表現可能な浮動小数点値を求める。
したがって、y が x より小さい場合、nextafter() は、x より小さい最大の表現可能な浮動小数点数を返す。
x と y が等しい場合、これらの関数は、y を返す。

nexttoward(), nexttowardf(), および nexttowardl() は、y が long double 型で、x と y が等しい場合
y を関数の戻値型に変換することを除き対応する nextafter() 関数と等価である。

【戻値】

成功した場合、これらの関数は、x の y 方向で次に表現可能な浮動小数点値を求める。
x と y が等しい場合、y を返す。
x が有限でオーバーフローが発生する場合、±HUGE_VAL, ±HUGE_VALF, および ±HUGE_VALL を関数の戻値に応じて
(x と同じ符号で)返す。
x または y が NaN の場合、NaN を返す。

【エラー】

なし

8.13.43 pow, powf, powl - 累乗関数

【書式】

```
#include <math.h>

double          pow(double x, double y);
float           powf(float x, float y);
long double     powl(long double x, long double y);
```

【解説】

これらの関数は、x の y 乗の値を計算する。
x が負の場合、アプリケーションは y が整数となるようにする必要がある。

【戻値】

成功した場合、これらの関数は、x の y 乗の値を返す。
x が負の有限な値で、y が非整数の場合 NaN を返す。
オーバーフローが発生する場合、±HUGE_VAL, ±HUGE_VALF, および ±HUGE_VALL を結果の符合に応じて返す。
アンダーフローが発生する場合、0.0 を返す。
(NaNを含む)いかなるyの値に対して、x が +1 の場合、1.0 を返す。
(NaNを含む)いかなるxの値に対して、y が ±0 の場合、1.0 を返す。
奇数の整数 y > 0 に対し、x が±0 の場合、±0 を返す。
奇数整数以外の y > 0 に対し、x が±0 の場合、+0 を返す。
x が -1 で y が ±Inf の場合、1.0 を返す。
|x| < 1 で y が -Inf の場合、+Inf を返す。
|x| > 1 で、y が -Inf の場合、+0 を返す。
|x| < 1 で、y が +Inf の場合、+0 を返す。
|x| > 1 で、y が +Inf の場合、+Inf を返す。
奇数整数の y < 0 で、x が =Inf の場合、-0 を返す。
奇数整数以外の y < 0 で、x が =Inf の場合、+0 を返す。
奇数整数の y > 0 で、x が =Inf の場合、-Inf を返す。
奇数整数以外の y > 0 で、x が =Inf の場合、+Inf を返す。
y < 0 で、x が +Inf の場合、+0 を返す。
y > 0 で、x が +Inf の場合、+Inf を返す。

【エラー】

なし

【関連項目】

exp, isnan

8.13.44 remainder, remainderf, remainderl - 剰余関数

【書式】

```
#include <math.h>

double          remainder(double x, double y);
float           remainderf(float x, float y);
long double     remainderl(long double x, long double y);
```

【解説】

これらの関数は、y が 0 でない時、浮動小数点剰余 r = x - n * y を返す。
n の値は、x / y に最も近い整数値である。
|n - x/y| = 1/2 の場合、n は偶数が選ばれる。
remainder() の動作は、丸めモードに依存しない。

【戻値】

成功した場合、これらの関数は、y が 0 でない場合浮動小数点剰余 r = x - n * y を返す。

IEC 60559 浮動小数点オプションをサポートしていないシステムでは、y がゼロの場合、0 を返す。

【エラー】

なし

【関連項目】

abs, div, ldiv

8.13.45 remquo, remquof, remquol - 商と剰余

【書式】

```
#include <math.h>

double          remquo(double x, double y, int *quo);
float           remquof(float x, float y, int *quo);
long double     remquol(long double x, long double y, int *quo);
```

【解説】

これらの関数は、remainder() 関数群と同じ剰余を計算する。
quo が指す領域には、符号が x/y の符号で、大きさが x/y の整数の商の絶対値と2のn乗を法として合同である絶対値をもつ値を格納する。ここで、n は実装定義の 3 以上の整数である。
T2EXリファレンス実装では n=31 である。

y がゼロの場合、quo が指す領域に格納される値は不定である。

【戻値】

これらの関数は、x REM y を返す。

IEC 60559 浮動小数点オプションをサポートしていないシステムでは、y がゼロの場合 0 を返す。

【エラー】

なし

【関連項目】

remainder

8.13.46 rint, rintf, rintl - 最も近い整数値に丸める

【書式】

```
#include <math.h>

double          rint(double x);
float           rintf(float x);
long double     rintl(long double x);
```

【解説】

これらの関数は、現在の丸めモードの方向で x に最も近い(double で表現された)整数値を返す。

現在の丸めモードが負の無限大方向の場合、rint() は floor と等価である。
現在の丸めモードが正の無限大方向の場合、rint() は ceil と等価である。

【戻値】

成功した場合、これらの関数は、現在の丸めモードの方向で、x に最も近い(double で表現された)整数値を返す。
x が NaN の場合、NaN を返す。
x が ±0 または ±Inf の場合、x を返す。
オーバーフローが発生する場合、これらの関数は、それぞれ ±HUGE_VAL, ±HUGE_VALF, ±HUGE_VALL を(x と同じ符号で)返す。

【エラー】

なし

【関連項目】

abs, ceil, floor, nearbyint

8.13.47 round, roundf, roundl - 最も近い整数値に丸める

【書式】

```
#include <math.h>

double          round(double x);
float           roundf(float x);
long double     roundl(long double x);
```

【解説】

これらの関数は、現在の丸め方向とは無関係に、中間の場合はゼロから離れた方向に丸めることで、
x を最も近い浮動小数点形式の整数値に丸める。

【戻値】

成功した場合、これらの関数は、丸められた整数の値を関数の戻値型で返す。
x が NaN の場合、NaN を返す。
x が ±0 または ±Inf の場合、x を返す。
オーバーフローが発生する場合、これらの関数は、それぞれ ±HUGE_VAL, ±HUGE_VALF, ±HUGE_VALL を(x と同

じ符号で)返す。

【エラー】

なし

8.13.48 scanbln, scanblnf, scanblnl, scanbn, scanbnf, scanbnl - FLT_RADIX を用いた指数の計算

【書式】

```
#include <math.h>

double          scalbln(double x, long n);
float           scalblnf(float x, long n);
long double     scalblnl(long double x, long n);
double          scalbn(double x, int n);
float           scalbnf(float x, int n);
long double     scalbnl(long double x, int n);
```

【解説】

これらの関数は、x と FLT_RADIX の n 乗の積を、通常 FLT_RADIX の n 乗を本当に計算することなく効率的に計算する。

【戻値】

成功した場合、これらの関数は、x と FLT_RADIX の n 乗の積を返す。
オーバフローが発生した場合、それぞれの戻値型に応じて ±HUGE_VAL, ±HUGE_VALF, ±HUGE_VALL を(x の符号に応じて)返す。

アンダーフローが発生した場合、0.0 を返す。
x が NaN の場合、NaN を返す。
x が ±0 または ±Inf の場合、x を返す。
n が 0 の場合、x を返す。

【エラー】

なし

8.13.49 sin, sinf, sinl - 正弦関数

【書式】

```
#include <math.h>

double          sin(double x);
float           sinf(float x);
long double     sinl(long double x);
```

【解説】

これらの関数は、ラジアンで指定された x の正弦を計算する。

【戻値】

成功した場合、これらの関数は、x の正弦を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、x を返す。
x が非正規化数の場合、x を返す。
x が ±Inf の場合、NaN を返す。

【エラー】

なし

【関連項目】

cos, tan, asin

8.13.50 sinh, sinhf, sinhl - 双曲線正弦関数

【書式】

```
#include <math.h>

double          sinh(double x);
float           sinhf(float x);
long double     sinhl(long double x);
```

【解説】

これらの関数は、 x の双曲線正弦を計算する。

【戻値】

成功した場合、これらの関数は、 x の双曲線正弦を返す。
オーバーフローが発生する場合、これらの関数は、それぞれ ±HUGE_VAL, ±HUGE_VALF, ±HUGE_VALL を(x と同じ符号で)返す。
x がNaNの場合、NaNを返す。
x が±0または±Infの場合、x を返す。
x が非正規化数の場合、x を返す。

【エラー】

なし

【関連項目】

asinh, cosh, tanh

8.13.51 sqrt, sqrtf, sqrtl - 平方根関数

【書式】

```
#include <math.h>

double          sqrt(double x);
float           sqrtf(float x);
long double     sqrtl(long double x);
```

【解説】

これらの関数は、 x の平方根を計算する。

【戻値】

成功した場合、これらの関数は、 x の平方根を返す。
有限な x < -0 の場合、NaN を返す。
x が NaN の場合、NaN を返す。
x が ±0 または +Inf の場合、x を返す。
x が -Inf の場合、NaN を返す。

【エラー】

なし

8.13.52 tan, tanf, tanl - 正接関数

【書式】

```
#include <math.h>

double          tan(double x);
float           tanf(float x);
long double     tanl(long double x);
```

【解説】

これらの関数は、ラジアンで指定した x の正接を計算する。

【戻値】

成功した場合、これらの関数は、 x の正接を計算を返す。
アンダーフローが発生する場合、0.0 を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、x を返す。
x が非正規化数の場合、x を返す。
x が ±Inf の場合、NaN を返す。
オーバーフローが発生する場合、これらの関数は、それぞれ ±HUGE_VAL, ±HUGE_VALF, ±HUGE_VALLを(結果と同じ符号で)返す。

【エラー】

なし

【関連項目】

atan, sin, cos

8.13.53 tanh, tanhf, tanhl - 双曲線正接関数

【書式】

```
#include <math.h>

double          tanh(double x);
float           tanhf(float x);
long double     tanhl(long double x);
```

【解説】

これらの関数は、 x の双曲線正接を計算する。

【戻値】

成功した場合、これらの関数は、 x の双曲線正接を返す。
x が NaN の場合、NaN を返す。
x が ±0 の場合、x を返す。
x が ±Inf の場合、±1 を返す。
x が非正規化数の場合、x を返す。

【エラー】

なし

【関連項目】

atanh, tan

8.13.54 tgamma, tgammaf, tgammal - ガンマ関数

【書式】

```
#include <math.h>

double          tgamma(double x);
float           tgammaf(float x);
long double     tgammal(long double x);
```

【解説】

これらの関数は、x のガンマ関数を計算する。

【戻値】

成功した場合、これらの関数は Gamma(x) を返す。
x が負の整数の場合、NaN を返す。
x が負の整数の場合、IEC 60559 浮動小数点オプションをサポートしているシステムでは NaN を返す。

x が ±0 の場合、これらの関数は、それぞれ ±HUGE_VAL, ±HUGE_VALF, ±HUGE_VALL を返す。
オーバーフローを起こす場合、それぞれ ±HUGE_VAL, ±HUGE_VALF, ±HUGE_VALL を結果の符合で返す。
x が NaN の場合、NaN を返す。
x が +Inf の場合、x を返す。
x が -Inf の場合、NaN を返す。

【エラー】

なし

【関連項目】

lgamma_r

8.13.55 trunc, truncf, truncl - 0 に近いほうの整数値に丸める

【書式】

```
#include <math.h>

double          trunc(double x);
float           truncf(float x);
long double     truncl(long double x);
```

【解説】

これらの関数は、x をその絶対値が x の絶対値を超えないもっとも近い整数値に丸め浮動小数点形式にする。

【戻値】

成功した場合、これらの関数は、x をその絶対値が x の絶対値を超えないもっとも近い整数値に丸めた値を返す。
x が NaN の場合、NaN を返す。
x が ±0 または ±Inf の場合、x を返す。

【エラー】

なし

8.13.56 y0, y1, yn - 第二種ベッセル関数

【書式】

```
#include <math.h>

double  y0(double x);
double  y1(double x);
double  yn(int n, double x);
```

【解説】

これらの関数は、それぞれx の 0次、1次、n次の第二種ベッセル関数を計算する。

【戻値】

成功した場合、これらの関数は、それぞれ対応する第二種ベッセル関数値を返す。
x が NaN の場合、NaN を返す。
x が負の場合、-HUGE_VAL または NaN を返す。
x が 0.0 の場合、-HUGE_VAL を返す。
アンダーフローを起こす場合、0.0 を返す。
オーバーフローを起こす場合、-HUGE_VAL または 0.0 を返す。

【エラー】

なし

【関連項目】

j0

8.13.57 lgamma_r, lgammaf_r, lgammal_r - ログガンマ関数

【書式】

```
#include <math.h>

double          lgamma_r(double x, int* signp);
float           lgammaf_r(float x, int* signp);
long double     lgammal_r(long double x, int* signp);
```

【解説】

これらの関数は、x のガンマ関数の絶対値の自然対数を計算する。
x は、正でない整数であってはならない。

【戻値】

成功した場合、これらの関数は、x のガンマ関数の絶対値の自然対数を返す。
x のガンマ関数の符号は、signp が指す領域に格納する。
x が正でない整数の場合、それぞれ +HUGE_VAL, +HUGE_VALF, +HUGE_VALL を返す。
オーバーフローを起こす場合、それぞれ ±HUGE_VAL, ±HUGE_VALF, ±HUGE_VALL を(結果と同じ符合で)返す。
x が NaN の場合、NaN を返す。
x が 1 または 2 の場合、+0 を返す。
x が ±Inf の場合、+Inf を返す。

【エラー】

なし

【関連項目】

exp

【補足事項】

これらの関数は、標準Cライブラリの lgamma, lgammaf, lgammal をスレッドセーフな形式に変更したものである。
T2EX では extern int signum という静的変数は存在しない。
signum で得られる符号は、各関数の signp で指定した領域に格納する。

8.14 netinet/in.h

ヘッダ netinet/in.h は、以下のネットワーク通信に関連したマクロや構造体、および関数プロトタイプ宣言を定義する。

型

in_port_t
: ネットワーク通信のポート番号に用いる型である。
stdint.h で定義されている uint16_t に等しい。

in_addr_t
: ネットワーク通信のIPv4アドレスを表現する型である。
stdint.h で定義されている uint32_t に等しい。

struct in_addr
: ネットワーク通信のアドレスを in_addr_t より抽象的に表現する構造体である。
少なくとも以下のメンバを含む構造体の必要がある。

in_addr_t	s_addr	

struct sockaddr_in
: ネットワーク通信のソケットのアドレスを表現する構造体である。
少なくとも以下のメンバを含む構造体の必要がある。

uint8_t	sin_len	アドレスのサイズ(バイト数)
sa_family_t	sin_family	アドレスファミリー(AF_INET)
in_port_t	sin_port	ポート番号
struct in_addr	sin_addr	IPアドレス

sin_port と sin_addr は、ネットワークバイトオーダーで格納する必要がある。

getsockopt(), setsockopt() 関数の level の値に用いるために、以下のシンボル定数を定義する。

IPPROTO_IP
: インターネットプロトコル

IPPROTO_ICMP
: コントロールメッセージプロトコル

IPPROTO_RAW
: 生のIPパケットプロトコル

IPPROTO_TCP
: TCP プロトコル

IPPROTO_UDP
: UDP プロトコル

so_connect(), so_sendmsg(), sendto() 関数に対する、宛先アドレスとして用いるために、
以下のシンボル定数を定義する。

INADDR_ANY
: IPv4 ローカルホストアドレス

INADDR_BROADCAST
: IPv4 ブロードキャストアドレス

IPv4 アドレスを文字列として格納するためのサイズとして、以下のシンボル定数を定義する。

INET_ADDRSTRLEN
: 16　　IPv4 アドレスの文字列形式の長さ

8.15 search.h

ヘッダ search.h は、以下の各種テーブル探索のための型、および関数プロトタイプ宣言を定義する。

型

ENTRY 型

構造体 entry に対する型で、その構造体は以下のメンバを含む:

```
char            *key                    検索キーとなるヌル文字で終わる文字列
void            *data                   key に対応したデータ
```

ACTION 型

検索操作を指定する値で、以下のように定義する。

```
enum {
        FIND,                   /* 検索のみ行う */
        ENTER                   /* 見つからない場合挿入 */
} ACTION;
```

VISIT 型

二分気の探索状態を示す値で、以下のように定義する。

```
enum {
        preorder,               /* ノードへの初回訪問 */
        postorder,              /* ノードへの二回目の訪問 */
        endorder,               /* ノードへの三回目の訪問 */
        leaf                    /* ノードではなく葉である */
} VISIT;
```

hsearch_data

ハッシュテーブルを表す構造体。内容は実装定義とする。
T2EXリファレンス実装では、以下のように定義している。

```
struct hsearch_data {
        void    *htable;
        int     htablesize;
};
```

関数

8.15.1 hcreate_r, hdestroy_r, hsearch_r - ハッシュテーブルの管理

【書式】

```
#include <search.h>

int     hcreate_r(size_t nel, struct hsearch_data *htab);
void    hdestroy_r(struct hsearch_data* htab);
int     hsearch_r(ENTRY item, ACTION action, ENTRY **result, struct hsearch_data *htab);
```

【解説】

これらの関数は、ハッシュテーブルを管理する。

hcreate_r() は、表のために十分な領域を割当て、ハッシュテーブルを管理するためのデータ htab を初期化する。
htab の内容は、最初に hcreate_r() を呼び出す前に 0 に初期化しておく必要がある。
アプリケーションは hsearch_r() を使用する前に、hcreate_r() が呼ばれていることを保証する必要がある。
nel は、表が含んでいるエントリの最大数の推定値である。
この値は、特定の数学的に都合のよい状況を得るために、アルゴリズムによってより大きな値に調整されることがある。

hdestroy_r() は、hcreate_r() で作成したハッシュテーブル htab に割り当てた領域を開放する。

hsearch_r() は、htab で指定したハッシュテーブルに関して、item.key と同じキーの値を持つ項目を検索し、
項目が見つかった場合、その項目へのポインタを result が指す領域に格納する。
item.data は、item.key と結びついたデータへのポインタである。
キーの比較には strcmp() を使用する。

項目が見つからない場合、ACTION で指定した以下の動作を実行する。
・ENTER は、item のコピーをテーブルの適切な場所に挿入することを指示する。
・FIND は、いかなのエントリーも操作しないことを指示する。

【戻値】

hcreate_r() は、成功した場合 0 以外の値を返す。それ以外の場合 0 を返す。

hsearch_r() は、成功した場合 0 以外の値を返す。
action が ENTER でハッシュテーブルが一杯の場合、または action が FIND で項目が見つからなかった場合 0
を返す。

hdestroy_r() は、値を返さない。

【エラー】

なし

【関連項目】

bsearch, lsearch, tsearch

8.15.2 insque, remque - キューに要素を挿入/削除

【書式】

```
#include <search.h>

void    insque(void *element, void *prev);
void    remque(void *element);
```

【解説】

insque() および reqeue() は、双方向リンクリストで構成されるキューを操作する。
キューは、循環タイプでも線形タイプでもよい。insque() または remque() を用いるアプリケーションは、
最初の二つのメンバが同じ構造体型へのポインタで、他のメンバがアプリケーション固有である構造体を定義する
必要がある。
その構造体の最初のメンバは、キュー内の次のエントリへの前方ポインタである。
第二のメンバは、キュー内の直前のエントリへの後方ポインタである。キューが線形の場合、キューは NULL で終
端される。
構造体およびポインタメンバの名前には特に規定はない。

insque() は、element が指す要素を、キュー内の prev が指す要素の直後に挿入する。

remque() は、element が指す要素を、キューから取り除く。

キューが線形リストとして使われる場合、element をキューの初期要素とすると、insque(&element, NULL) の呼
び出しは、
element の前方および後方ポインタを NULL に初期化する必要がある。

キューが循環リストとして使われる場合、アプリケーションはキューの初期要素の前方および後方ポインタをその
要素自身を
指すように初期化しておく必要がある。

【戻値】

insque() および remque() は、値を返さない。

【エラー】

なし

8.15.3 lfind, lsearch - 線形探索と更新

【書式】

```
#include <search.h>

void *lfind(const void *key, const void *base, size_t *nelp, size_t width, int (*compar)(const void *,
const void *));
void *lsearch(const void *key, void *base, size_t *nelp, size_t width, int (*compar)(const void *,
const void *));
```

【解説】

lsearch() は、表を線形探索し表内の見つかったエントリーへのポインタを返す。
エントリが見つからなかった場合、lsearch() は表の最後にエントリを追加する。
key は、表内で比較されるものと一致する要素へのポインタである。
base は、表の最初の要素を指す。
width は、要素のバイトサイズである。
nelp は、表の現在の要素数を持つ整数へのポインタである。nelp が指す整数は、表に要素が追加されたとき増加される。

compar は、アプリケーションが提供する比較関数へのポインタである。(例えば strcmp())
比較関数は、比較する二つの要素へのポインタを引数として呼び出される。
比較関数は、要素が一致した場合 0 を戻し、それ以外の場合 0 以外の値を返すようにする必要がある。

lfind() は、エントリが見つからなかった場合、表にエントリを追加しないことを除き lsearch() と等価である。

【戻値】

エントリが見つかった場合、lsearch(), lfind() ともにそのエントリへのポインタを返す。
見つからない場合、lfind() は NULL を返し、lsearch() は新たに追加した要素へのポインタを返す。
エラーが発生した場合は、どちらの関数も NULL を返す。

【エラー】

なし

【関連項目】

bsearch, hsearch_r, tsearch

8.15.4 tdelete, tfind, tsearch, twalk - 二分木(バイナリトリー)の管理

【書式】

```
#include <search.h>

void   *tdelete(const void *key, void **rootp, int(*compar)(const void *, const void *));
void   *tfind(const void *key, void *const *rootp, int(*compar)(const void *, const void *));
void   *tsearch(const void *key, void **rootp, int(*compar)(const void *, const void *));
void    twalk(const void *root, void (*action)(const void *nodep, VISIT which, int depth));
```

【解説】

tdelete(), tfind(), tsearch() および twalk() は、バイナリ探索木を操作する。
compar は、ユーザが提供する比較関数へのポインタである。
比較関数は、比較する二つの要素への各ポインタを引数として呼び出される。
比較関数は、第一引数が示すものが第二引数が示すものに対し、小さい場合負、等しい場合0、大きい場合正の値を返す必要がある。

tsearch() は、二分木を探索し構築する。key は、探索しあるいは格納される要素へのポインタである。
key が指す値と一致する要素のノードが見つかった場合、そのノードへのポインタを返す。
見つからない場合、key が指す値は二分木に挿入され(すなわち、新規ノードを作成し key 値をコピーする)、そ

して
その新規ノードへのポインタを返す。ポインタのみコピーされるので、アプリケーションは呼び出した側でデータを
格納するようにしなければならない。
rootp は、木のルートノードを指すポインタ変数へのポインタである。
rootp が指す値が NULL の場合、木は空である事を意味する。
この場合、rootp が指す変数は新しい木のルートとなったノードを指すよう設定される。

tfind() は、tsearch() 関数同様木の中のノードを探索し、見つかった場合そのノードへのポインタを返す。
見つからない場合、tfind() は NULL を返す。tfind() に対する引数は、tsearch() に対するものと同じである。

tdelete() は、二分探索木からノードを削除する。引数は、tsearch() に対するものと同じである。
rootp が指す変数は、削除されたノードが木のルートの場合変更される。
tdelete() は、削除されたノードの親へのポインタを返すか、
または、削除されたノードがルートノードの場合、不定の NULL 以外のポインタを返す。
ノードが見つからない場合、NULL を返す。

以下の場合の動作は未定義とする。
tsearch() が木に要素を追加した、または tdelete() が木から要素を削除した場合、以下の場合の動作は未定義とする。
・他のタスクでその木を同時に使用する。
・以前に実行した tfind() または tserach() が返したポインタを使用する。

twalk() は、二分探索木をトラバースする。root は、トラバースする木のルートノードへのポインタである。
(指定したノードをルートとしてその下をトラバースするために、どのノードも使用することができる。)

action は、各ノードに対して呼び出される関数で、以下の引数を持つ。

nodep は、訪問しているノードのアドレスである。nodep が指すデータ構造は不定であり、
アプリケーションが変更してはいけないが、ノードに格納された要素にアクセスするために、
nodep を要素へのポインタへのポインタにキャストすることができる。

which は、search.h で定義される、以下の列挙型の中の一つの値である。

```
typedef enum { preorder, postorder, endorder, leaf } VISIT;
```

この値は、(木の深さ優先で、左から右にトラバースして)訪問したノードが、初回、二回目、三回目か、葉かを意味する。

depth は、ノードの深さのレベルで、ルートの場合、値 0 となる。

呼び出した関数が、ルートへのポインタを変更すると結果は未定義となる。

action または compar の中で、木を変更すると結果は未定義となる。

複数タスクが同じ木にアクセスしない限り、これらの関数はスレッドセーフである。

【戻値】

ノードが見つかった場合、tserch() と tfind() は、そのノードへのポインタを返す。
見つからない時、tfind() は NULL を返し、tsearch() は、挿入した項目へのポインタを返す。

新規ノードに利用できる領域が不足した場合、tsearch() は、 NULL を返す。

tdelete(), tfind(), tsearch() は、rootp が NULL の場合、 NULL を返す。

tdelete() は、削除したノードの親へのポインタを返すか、または、削除されたノードがルートノードの場合、
不定の NULL 以外のポインタを返すか、または、ノードが見つからない場合 NULL を返す。

twalk() は、値を返さない。

【エラー】

なし

【関連項目】

bsearch, hsearch_r, lsearch

8.16 stdarg.h

ヘッダ stdarg.h は、可変個の引数リストを受け付けるポータブルな関数を記述できるようにするためのマクロ群を定義する。
ここで定義されているマクロを使用して(printf()のような)可変個の引数を持つ関数を作成しないと、コンパイラが実装している
引数の引渡し規約しだいで、正しく可変個の引数を参照できない可能性があり、移植性を失う。

型

va_list
　　可変個の引数を持つ関数の引数リストを表す型を示す。

以下のマクロを定義する。

void va_start(va_list ap, argN);
　　va_start() マクロは、va_arg(), va_end() マクロで使用される ap を初期化する。
　　そのため、va_arg(), va_end()より先に呼び出さなければならない。
　　ap は、引数リストを処理するために用いる変数である。va_start() マクロで初期化される。
　　argN は、〝...〟で表記される可変個の引数リストの直前の引数である。
　　va_start() マクロは、ap に対して va_end() マクロを呼び出すことなく、
　　その ap を再初期化するために呼び出してはならない。

type va_arg(va_list ap, type);
　　va_arg() マクロは、ap から type で指定した型を持つ引数を1つ返す。
　　すなわち、va_arg(ap, type) を最初に呼び出すと、va_start() で指定した argN の次の引数が返る。
　　va_arg() マクロを呼び出す度に ap は更新され、次回の呼び出し時に、
　　さらに次の引数が返される。
　　ap は、引数リストを処理するために用いる変数である。va_start() または va_copy() マクロで
　　初期化されている必要がある。
　　type は、取り出す引数のデータ型を指定する。
　　type は、その型のオブジェクトへのポインタの型が、型の後ろに単に'*'を置くことで得られる型名でなければならない。

　　次の引数が実際には存在しないか、type が実際の次の引数の型と互換性がない場合、
　　以下の場合を除き、動作は未定義とする。
　　・ 一方の型が符号付き整数で、もう一方の型が対応する符号無し整数型で、値が両方の型で表現できる。
　　・ 一方の型が void へのポインタで、もう一方が文字型へのポインタである。

void va_copy(va_list dest, va_list src);
　　va_copy() マクロは、dest を src のコピーとして初期化する。
　　dest に va_start マクロを適用し、現在の src の状態になるまでに src に va_arg を
　　適用したのと同じ事を、dest に対して実行したのと同じ結果となる。
　　単純に dest = src; と代入してはいけない。

　　va_copy() も va_start() マクロも、dest に対して va_end() マクロを呼び出すことなく、
　　その dest を再初期化するために呼び出してはならない。

void va_end(va_list ap);
　　va_end() マクロは、クリーンアップのために用いる。
　　これは ap を、va_start() マクロか va_copy() マクロが再度呼ばれて再初期化されるまで
　　不正にする。
　　va_start() マクロおよび va_copy() マクロの各呼び出しは、同じ関数内で対応する va_end()
　　マクロの呼び出しと1対1に対応していなければならない。
　　va_start() .... va_end() を、入れ子として使うことは可能である。

使用例:

```
void func1(int narg, ...)
{
        va_list ap;
        int             i;

        va_start(ap, narg);
        for (i = 0; i < narg; i++) {
                xxx = va_arg(ap, type);
                ...
        }
```

```
        va_end(ap);
}
```

8.17 stdbool.h

ヘッダ stdbool.h は、ブーリアン型(論理型)を扱うためのマクロを定義する。

マクロ

bool
: _Bool に展開される。

true
: 整数定数 1 に展開される。

false
: 整数定数 0 に展開される。

__bool_true_false_are_defined
: 整数定数 1 に展開される。

8.18 stddef.h

ヘッダ stddef.h は、以下の共通に使われる定数、マクロ、型を定義する。

定数

NULL
: 空のポインタを表す定数マクロ。void* にキャストした値 0 の整数定数式に展開される。

マクロ

offsetof(type, member)
: type で指定した構造体の先頭から member で示すメンバまでのバイトオフセットを返す。
値は size_t 型の整数定数式である。

型

ptrdiff_t
: 二つのポインタの差分の結果を表す符号付き整数型。

wchar_t
: システムロケールが規定する最大の文字セットの全ての文字を区別できるコード値に必要な値の範囲を持つ整数型。
例えば、ユニコードをバイト列として表現する UTF-8 ではマルチバイト文字が使われるが、その最大バイト数を表現できる整数型を意味する。
ヌル文字は、コード値 0 を持たなければならない。
文字セット中の各要素(UTF-8 であれば複数バイトの事がある)を、1文字の整数文字定数として用いるときの型であり、ワイド文字型と呼ぶ。

size_t
: sizeof 演算子の結果を表す符号無し整数型。

8.19 stdint.h

ヘッダ stdint.h は、特定の幅を持つ整数に関する型や、その限界値に対するマクロを定義する。

整数型定義

厳密な幅を持つ整数型:

int8_t
int16_t
int32_t
int64_t
: intN_t 形式の型は、パディング無しの幅 N ビットの2の補数表現の符号付き整数型を示す。
したがって、例えば int8_t は、厳密に8ビットの幅を持つ符号付き整数型となる。

uint8_t
uint16_t
uint32_t
uint64_t
: uintN_t 形式の型は、幅 N ビットの符号無し整数型を示す。
したがって、例えば uint32_t は、厳密に32ビットの幅を持つ符号無し整数型となる。

最小幅を持つ整数型:

int_least8_t
int_least16_t
int_least32_t
int_least64_t
: int_leastN_t 形式の型は、少なくとも N ビットの幅を持つ符号付き整数型を示す。
より小さいサイズの符号付き整数型であれば、少なくとも指定された幅を持つことはない。

uint_least8_t
uint_least16_t
uint_least32_t
uint_least64_t
: uint_leastN_t 形式の型は、少なくとも N ビットの幅を持つ符号無し整数型を示す。
より小さいサイズの符号無し整数型であれば、少なくとも指定された幅を持つことはない。

最速の最小幅を持つ整数型:

以下の各型は、通常最速に演算できる、少なくとも指定された幅を持つ整数型を示す。

int_fast8_t
int_fast16_t
int_fast32_t
int_fast64_t
: typedef 名前 int_fastN_t 形式の型は、少なくとも N ビットの幅を持つ最速の符号付き整数型を示す。

uint_fast8_t
uint_fast16_t
uint_fast32_t
uint_fast64_t
: typedef 名前 uint_fastN_t 形式の型は、少なくとも N ビットの幅を持つ最速の符号無し整数型を示す。

オブジェクトのポインタを格納する整数型:

intptr_t
: ポインタを格納可能な符号付き整数型を示す。
void 型へのポインタが intptr_t 型へ変換可能で、かつ void 型へのポインタに再び変換した場合、結果は元のポインタと等しくなる。

uintptr_t
: ポインタを格納可能な符号無し整数型を示す。
void 型へのポインタが uintptr_t 型へ変換可能で、かつ void 型へのポインタに再び変換した場合、結果は元のポインタと等しくなる。

最大幅を持つ整数型:

intmax_t

すべての符号付き整数型を表現可能な符号付き整数型を示す。

uintmax_t
すべての符号無し整数型を表現可能な符号無し整数型を示す。

マクロ

厳密な幅を持つ整数型の限界値:

INT8_MIN
INT16_MIN
INT32_MIN
INT64_MIN
n ビットの幅を持つ符号付き整数型の最小値を示す。
INTn_MIN に対し、最小値は、-(2 ^ (n-1)) となる('^'はべき乗を表す)。

INT8_MAX
INT16_MAX
INT32_MAX
INT64_MAX
n ビットの幅を持つ符号付き整数型の最大値を示す。
INTn_MAX に対し、最大値は、2 ^ (n-1) - 1 となる('^'はべき乗を表す)。

UINT8_MAX
UINT16_MAX
UINT32_MAX
UINT64_MAX
n ビットの幅を持つ符号無し整数型の最大値を示す。
UINTn_MAX に対し、最大値は、(2 ^ n) - 1 となる('^'はべき乗を表す)。

最小幅の整数型の限界値:

INT_LEAST8_MIN
INT_LEAST16_MIN
INT_LEAST32_MIN
INT_LEAST64_MIN
最小の n ビットの幅を持つ符号付き整数型の最小値を示す。
INT_LEASTn_MIN に対し、最小値は、-(2 ^ (n-1)) となる('^'はべき乗を表す)。

INT_LEAST8_MAX
INT_LEAST16_MAX
INT_LEAST32_MAX
INT_LEAST64_MAX
最小の n ビットの幅を持つ符号付き整数型の最大値を示す。
INT_LEASTn_MAX に対し、最大値は、2 ^ (n-1) - 1 となる('^'はべき乗を表す)。

UINT_LEAST8_MAX
UINT_LEAST16_MAX
UINT_LEAST32_MAX
UINT_LEAST64_MAX
最小の n ビットの幅を持つ符号無し整数型の最大値を示す。
UINT_LEASTn_MAX に対し、最大値は、(2 ^ n) - 1 となる('^'はべき乗を表す)。

最速の最小幅整数型の限界値:

INT_FAST8_MIN
INT_FAST16_MIN
INT_FAST32_MIN
INT_FAST64_MIN
最速の最小の n ビットの幅を持つ符号付き整数型の最小値を示す。
INT_FASTn_MIN に対し、最小値は、-(2 ^ (n-1)) となる('^'はべき乗を表す)。

INT_FAST8_MAX
INT_FAST16_MAX
INT_FAST32_MAX
INT_FAST64_MAX
最速の最小の n ビットの幅を持つ符号付き整数型の最大値を示す。
INT_FASTn_MAX に対し、最大値は、(2 ^ (n-1)) - 1 となる('^'はべき乗を表す)。

UINT_FAST8_MAX
UINT_FAST16_MAX

UINT_FAST32_MAX
UINT_FAST64_MAX
最速の最小の n ビットの幅を持つ符号無し整数型の最大値を示す。
UINT_FASTn_MAX に対し、最大値は、(2 ^ n) - 1 となる('^'はべき乗を表す)。

ポインタを格納可能な整数型の限界値:

INTPTR_MIN
ポインタを格納可能な符号付き整数型の最小値を示す。
最小値は、 -((2 ^ 31)) となる('^'はべき乗を表す)。

INTPTR_MAX
ポインタを格納可能な符号付き整数型の最大値を示す。
最大値は、 (2 ^ 31) - 1 となる('^'はべき乗を表す)。

UINTPTR_MAX
ポインタを格納可能な符号無し整数型の最大値を示す。
最大値は、 (2 ^ 32) - 1 となる('^'はべき乗を表す)。

最大幅を持つ整数型の限界値:

INTMAX_MIN
最大幅を持つ符号付き整数型の最小値を示す。
最小値は、 -((2 ^ 63)) となる('^'はべき乗を表す)。

INTMAX_MAX
最大幅を持つ符号付き整数型の最大値を示す。
最大値は、 (2 ^ 63) - 1 となる('^'はべき乗を表す)。

UINTMAX_MAX
最大幅を持つ符号無し整数型の最大値を示す。
最大値は、 (2 ^ 64) - 1 となる('^'はべき乗を表す)。

他の整数型の限界値:

PTRDIFF_MIN
ptrdiff_t 型の最小値を示す。
最小値は、 INT32_MIN となる。

PTRDIFF_MAX
ptrdiff_t 型の最大値を示す。
最大値は、 INT32_MAX となる。

SIZE_MAX
size_t 型の最大値を示す。
最大値は、 UINT32_MAX となる。

WCHAR_MIN
wchar_t 型の最小値を示す。
最小値は、 0 となる。

WCHAR_MAX
wchar_t 型の最大値を示す。
最大値は、 UINT32_MAX となる。

8.20 stdio.h

ヘッダ stdio.h は、以下の標準入出力のための型、マクロおよび関数プロトタイプ宣言を定義する。

標準Cライブラリが、入出力ストリームとしてファイルとソケットを等価に扱えるのに対し、
T2EX標準C互換ライブラリの標準入出力では、入出力ストリームとしてファイルのみを対象とし、ソケットは対象としない。

型

FILE
: ファイルを管理するための構造体の型を示す。

  その要素は実装依存であり、通常は
  ファイルティスクリプタや読み書きをバッファリングするためのバッファ、
  読み書きを行うファイル位置(ファイルオフセットとは異なる)、
  ファイル終端に達したことを示すインジケータ(ファイル終端インジケータ)、
  入出力で発生したエラー番号を記録する領域(エラー情報)、
  その他、ファイル入出力ストリームを制御するための必要なすべての情報を持っている。
  構造体内の要素は、本節で示す関数によってアクセスされ、
  ユーザが要素に直接アクセスしてはいけない。

```
typedef struct {
        /* 実装依存 */
} FILE;
```

  T2EXリファレンス実装では、実装依存部は int opaque[23]; と定義しており、
  ユーザがその要素に直接アクセスしてはいけない。

  ファイルに対する逐次的な入力および出力をストリームと呼び FILE* で表す。

errno_t
: エラー番号を表す整数型を示す。

fpos_t
: ファイル内の位置を表す型を示す。
  T2EXリファレンス実装では 32ビット符号付整数型であるが、整数型でない実装も許容する。

fpos64_t
: 64ビットでファイル内の位置を表す型を示す。
  T2EXリファレンス実装では 64ビット符号付整数型であるが、整数型でない実装も許容する。

off_t
: 32ビットでファイルサイズを表す整数型を示す。

off64_t
: 64ビットでファイルサイズを表す整数型を示す。

size_t
: sizeof 演算子の結果を表す符号無し整数型を示す。

ssize_t
: 0以上のバイト数および負のエラーコードを表す型を示す。

va_list
: 可変個の引数を持つ関数の引数リストを表す型を示す。

マクロ

BUFSIZ
: 入出力バッファのサイズ(バイト数)を示す。

ファイル入出力におけるバッファリングの状態を示すマクロ:

_IOFBF	入出力の完全バッファリングを行うことを示す。
_IOLBF	入出力の行バッファリングを行うことを示す。
_IONBF	入出力のバッファリングを行わないことを示す。

ファイルの現在のオフセットを移動する際に基準位置を示すためのマクロ:

SEEK_CUR	移動の基点がファイルの現在位置であることを示す。
SEEK_END	移動の基点がファイルの終端であること示す。
SEEK_SET	移動の基点がファイルの先頭であることを示す。

FILENAME_MAX
ファイル名の文字列の最大長(バイト数)を示す。

FOPEN_MAX
同時にオープンできるファイルの最大数を示す。

EOF
ファイルの終端を示す。

コンソール入出力のために以下の FILE* 型のデータを定義する。

stdin	標準コンソール入力を示す。
stdout	標準コンソール出力を示す。
stderr	標準コンソールエラー出力を示す。

関数

8.20.1 libc_stdio_init - 標準入出力の初期化

【書式】

```
#include <stdio.h>

ER      libc_stdio_init(void)
```

【解説】

標準入出力ライブラリの初期化を行う。ファイル管理機能の fs_main() を呼び出した後、明示的にこの API コールを呼び出す必要がある。
この API コールは、stdin, stdout, stderr が直ちに利用できるよう初期化する。
その他の標準入出力ライブラリの使用に先立って呼び出す必要がある。

【戻値】

【エラー】

戻値はエラーコードである。

E_OK	正常終了

【関連項目】

libc_stdio_cleanup

【補足事項】

libc_stdio_init() は、T2EX 独自 API コールである。

8.20.2 libc_stdio_cleanup - 標準入出力の終了

【書式】

```
#include <stdio.h>
```

```
ER      libc_stdio_cleanup(void)
```

【解説】

標準入出力ライブラリを終了する。
ファイル管理機能の fs_main() でファイル管理機能を終了させる前に、明示的にこの API コールを呼び出す必要がある。

【戻値】

【エラー】

戻値はエラーコードである。
E_OK　　正常終了

【関連項目】

libc_stdio_init

【補足事項】

libc_stdio_cleanup() は、T2EX 独自 API コールである。

8.20.3 clearerr - ストリームのエラー番号のクリア

【書式】

```
#include <stdio.h>

void    clearerr(FILE* stream);
```

【解説】

clearerr() は、stream が指すファイル終端インジケータおよびエラー情報をクリアする。

【戻値】

clearerr() は、値を返さない。

【エラー】

なし

【関連項目】

feof, ferror, fileno, fopen

8.20.4 feof - ファイル終端の検査

【書式】

```
#include <stdio.h>

int             feof(FILE* stream);
```

【解説】

feof() は、 stream が指すストリームがファイル終端まで達しているか検査する。
ファイル終端に達している場合 0 以外の値を返す。

【戻値】

feof() は、stream が、ファイル終端に達している場合 0 以外の値を返す。

【エラー】

なし

【関連項目】

clearerr, ferror, fileno, fopen

8.20.5 ferror - ストリームのエラー検査

【書式】

```
#include <stdio.h>

errno_t         ferror(FILE* stream);
```

【解説】

ferror() は、 stream が指すストリームに対して、エラー状態か検査する。stream にエラーが記録されていれば、そのエラー番号を返す。

【戻値】

ferror() は、ストリームにエラーが記録されていれば、そのエラー番号を返す。

【エラー】

なし

【関連項目】

clearerr, feof, fileno, fopen

8.20.6 fileno - ストリームのファイルディスクリプタの取得

【書式】

```
#include <stdio.h>

int             fileno(FILE* stream);
```

【解説】

fileno() は、stream が指すストリームに対応するファイルディスクリプタの値を返す。

【戻値】

fileno() は、成功した場合、stream のファイルディスクリプタ値(非負)を返す。
エラーが発生した場合、ストリーム内にエラー番号を記録し、-1 を返す。

【エラー】

エラーが発生した場合、ferror() を呼び出すと、以下を返すことがある:

EBADF   stream が不正

【関連項目】

clearerr, feof, ferror, fopen

8.20.7 fgetc, getc, getchar - ストリームから1文字読込み

【書式】

```
#include <stdio.h>

int             fgetc(FILE* stream);
int             getc(FILE *stream);
int             getchar(void);
```

【解説】

fgetc() は、stream が指すストリームから次の文字(1バイト)を unsigned char 型として取得し、int 型に変換して返す。
ストリーム内のファイル位置が存在する場合、それを一つ進める。
ストリームがファイル終端に達していた場合は、-1 を返す。

getc() は、fgetc() と等価であるが、マクロとして実装されている。
したがって stream を複数回評価する可能性がある。

getchar() は、getc(stdin) と等価である。

【戻値】

fgetc(), getc(), getchar() は、成功した場合、ストリームの現在ファイル位置から読み込んだ1文字を返す。
ストリームがファイル終端に達していた場合は、ストリーム内のファイル終端インジケータをセットし EOF を返す。
エラーが発生した場合、ストリーム内にエラー番号を記録し EOF を返す。

【エラー】

エラーが発生した場合、ferror() を呼び出すと、以下を返すことがある:

EAGAIN	ストリームに対応するファイルディスクリプタに対し、O_NONBLOCK フラグがセットされていて、読込みにより待ちが発生する
EBADF	ストリームのファイルディスクリプタが、読込みオープンされた正しいファイルディスクリプタではない
EINTR	fs_break() による中止
EIO	I/Oエラー

【関連項目】

feof, ferror, fgets, fread, ungetc, fopen

8.20.8 fgets - ストリームから1行の読込み

【書式】

```
#include <stdio.h>

char*   fgets(char* s, int size, FILE* stream);
```

【解説】

fgets() は、stream から size-1 バイトまたは、改行('¥n')が読み込まれるかファイル終端に達するまで、文字列を読込み s が指す領域に格納し最後にヌル文字を追加する。

【戻値】

fgets() は、成功した場合 s を返す。ストリームがファイル終端に達している場合、
ストリーム内のファイル終端インジケータをセットし、 NULL を返す。
エラーが発生した場合、ストリーム内にエラー番号を記録し、 NULL を返す。

【エラー】

fgetc() 参照。

【関連項目】

feof, ferror, fgetc, fread, ungetc, fopen

8.20.9 ungetc - 入力ストリームに1文字プッシュバック

【書式】

```
#include <stdio.h>

int             ungetc(int c, FILE* stream);
```

【解説】

ungetc() は、c を unsigned char に変換した文字(1バイト)を stream で指定した入力ストリームにプッシュバックする。
プッシュバックされた文字は、次の読込みで、プッシュバックされた逆の順に読み込まれる。
このストリームに対して fseek(), fsetpos(), rewind() を実行すると、プッシュバックされた文字は廃棄される。

プッシュバックしても、ストリームに対応するファイルのディスク上のデータは変化しない。

1文字のプッシュバックは保証するが、引き続くプッシュバックが成功するかは実装に依存する。
T2EXリファレンス実装では、メモリが不足しない限り何文字でもプッシュバックできる。

c の値が EOF の場合、ungetc() は失敗し、ストリームの状態は変化しない。

成功した場合には、ストリーム内のファイル終端インジケータはクリアされる。ストリーム内のファイル位置は、プッシュバックの度に -1 される。ungetc() 呼び出し前に、ファイル位置が 0 の場合は、結果のファイル位置は不定である。

【戻値】

ungetc() は、成功した場合 c の値を返す。
失敗時には EOF を返す。

【エラー】

なし

【関連項目】

fseek, fgetc, fopen, fsetpos, rewind, setbuf

8.20.10 fputc, putc, putchar - ストリームに1文字出力

【書式】

```
#include <stdio.h>
```

```
int             fputc(int c, FILE* stream);
int             putc(int c, FILE* stream);
int             putchar(int c);
```

【解説】

fputc() は、c を unsigned char に変換し、stream で指定した出力ストリームに書き出す。
ストリームにファイル位置が存在する場合、書き出しはその位置に行われ、ファイル位置を一つ進める。

ファイルが位置決めをサポートしていないか、ストリームが追加モードでオープンされている場合は、
文字をストリームの最後に追加書込みする。

putc() は、fputc() と等価であるが、マクロとして実装されている。
したがって stream を複数回評価する可能性がある。

putchar() は、putc(c, stdout) と等価である。

【戻値】

fputc(), putc(), putchar() は、成功した場合には書き込んだ文字の値を返す。
エラーが発生した場合、ストリーム内にエラー番号を記録し、EOF を返す。

【エラー】

エラーが発生した場合、ferror() を呼び出すと、以下を返すことがある:

```
EAGAIN          ストリームに対応するファイルディスクリプタに対し、O_NONBLOCK フラグがセットされて
                いて、書込みにより待ちが発生する
EBADF           ストリームに対応するファイルディスクリプタが不正
EFBIG           ファイルサイズの制限を越える
EINTR           fs_break() による中止
EIO             I/Oエラー
ENOSPC          デバイスの容量不足
```

【関連項目】

ferror, fopen, fputs, setbuf

8.20.11 fputs, puts - ストリームに文字列出力

【書式】

```
#include <stdio.h>

int             fputs(const char* s, FILE* stream);
int             puts(const char* s);
```

【解説】

fputs() は、s が示すヌル文字で終わる文字列を、stream で指定した出力ストリームに書き込む。
最後のヌル文字は出力されない。

puts() は、fputs(s, stdout) と等価である。

【戻値】

fputs(), puts() は、成功した場合には0以上の値を返す。
エラーが発生した場合、ストリーム内にエラー番号を記録し、EOF を返す。

【エラー】

fputc() 参照

【関連項目】

fputc, fopen, ferror

8.20.12 fgetpos, fgetpos64 - 現在のファイル位置の取得

【書式】

```
#include <stdio.h>

int fgetpos(FILE* stream, fpos_t* pos);
int fgetpos64(FILE* stream, fpos64_t* pos);
```

【解説】

fgetpos() は、stream で指定したストリームの現在のファイル位置を pos が示す領域に格納する。

fgetpos64() は、引数に fpos64_t 型を使用したもので、64ビットのファイル位置を扱うことができる。

【戻値】

fgetpos(), fgetpos64() は、成功した場合 0 を返す。
エラーが発生した場合、ストリーム内にエラー番号を記録し、0 以外の値を返す。

【エラー】

エラーが発生した場合、ferror() を呼び出すと、以下を返すことがある:

EOVERFLOW	現在位置が fpos_t で表現できない
EBADF	ストリームに対応するファイルディスクリプタが不正

【関連項目】

fopen, fseek, ftell, rewind, ungetc

8.20.13 fsetpos, fsetpos64 - 現在のファイル位置の設定

【書式】

```
#include <stdio.h>

int fsetpos(FILE* stream, const fpos_t* pos);
int fsetpos64(FILE* stream, const fpos64_t* pos); // 非標準機能
```

【解説】

fsetpos() は、stream で示されるストリームの現在のファイル位置を pos が示す値に設定する。
pos は以前に実行した、fgetpos() で得られた値である。

成功した場合には、ストリーム内のファイル終端インジケータをクリアし、ungetc() でプッシュバックされた文字は破棄する。

fsetpos64() は、引数に fpos64_t を使用したもので、pos の値は、fgetpos64() で得られる。

【戻値】

fsetpos(), fsetpos64() は、成功した場合は 0 を返す。
エラーが発生した場合、ストリーム内にエラー番号を記録し、0 以外の値を返す。

【エラー】

エラーが発生した場合、ferror() を呼び出すと、以下を返すことがある:

EAGAIN	ストリームに対応するファイルディスクリプタの O_NONBLOCK フラグがセットされ、書込み操作で待ちに入る

EBADF	ストリームに対応するファイルディスクリプタが不正
EFBIG	ファイルサイズの制限を越える
EINTR	fs_break() による中止
EIO	I/Oエラー
ENOSPC	デバイスの容量不足

【関連項目】

fopen, fseek, ftell, rewind, ungetc

8.20.14 fseek, fseek64, rewind, ftell, ftell64 - 現在のファイル位置の変更および取得

【書式】

```
#include <stdio.h>

int fseek(FILE* stream, long offset, int whence);
int fseek64(FILE* stream, int64_t offset, int whence);
void rewind(FILE* stream);
long ftell(FILE* stream);
int64_t ftell64(FILE* stream);
```

【解説】

fseek() は、stream で指定したストリームのファイル位置を変更する。
バイト単位の新たな位置は、whence で指定される位置に offset を追加したものとなる。
whence は、以下の値で、右側に示した位置を指定する。

SEEK_SET	ファイルの先頭位置
SEEK_CUR	ファイルの現在位置
SEEK_END	ファイルの終端位置

成功した場合は、ストリームのファイル終端インジケータをクリアし、ungetc() でプッシュバックされたデータを破棄する。

現在のファイル終端位置を越えた位置にファイル位置を設定しようとするとエラーとなる。
このとき ferror(stream) は EINVAL を返す。

コンソールのようなファイル位置が存在しないシーク不可能なファイルに対して fseek() はエラーとなり、
このとき ferror(stream) は ESPIPE を返す。

fseek64() は、位置指定のための offset が int64_t 型であり、64ビットで位置を指定できることを除き、
fseek() と等価である。

rewind() は、(void) fseek(stream, 0, SEEK_SET) と等価である。
すなわち、ストリームのファイル位置をファイルの先頭に設定する。
また、rewind() は、ストリームのエラー情報をクリアする。

ftell() は、stream で指定したストリームの現在のファイル位置を返す。

ftell64() は、戻値が 64ビットである点を除き、ftell(stream) と等価である。

【戻値】

fseek(), fseek64() は、成功した場合は 0 を返す。
エラーが発生した場合、ストリーム内にエラー番号を記録し、-1 を返す。

rewind() は、値を返さない。

ftell(), ftell64() は、成功した場合は、現在のファイル位置を返す。
エラーが発生した場合、ストリーム内にエラー番号を記録し、-1 を返す。

【エラー】

エラーが発生した場合、ferror() を呼び出すと、以下を返すことがある:

EAGAIN	ストリームに対応するファイルディスクリプタの O_NONBLOCK フラグがセットされ、書込み操作で待ちに入る

EBADF　　ストリームに対応するファイルディスクリプタが不正
EFBIG　　位置がファイルサイズの制限を越える
EINVAL　　指定位置が不正
　　- 結果のファイル位置が負
　　- ファイル終端を越える位置
EINTR　　fs_break() による中止
EIO　　I/Oエラー
ESPIPE　　シーク不可能なファイル
EOVERFLOW　　ファイル位置が結果のデータで表現できない

ftell(), ftell64() は、EBADF と EOVERFLOW のみ返すことがある。

【関連項目】

fgetpos, fsetpos, ftell, rewind, ungetc

8.20.15 fread - バイナリストリームの入力

【書式】

```
#include <stdio.h>

size_t fread(void* ptr, size_t size, size_t nmemb, FILE* stream);
```

【解説】

fread() は、stream で指定したストリームから、要素のサイズが size バイトのデータを nmemb 個、
ptr が示す領域に読み込む。
stream のファイル位置は、読み込めたデータのバイト数だけ進められる。

【戻値】

fread() は、読込みに成功した要素数を返す。読込みエラーか EOF に達した場合は、この値は nmemb より小さくなる。
size か nmemb が 0 の場合は、0 を返し、stream の状態は変化しない。
エラーが発生した場合、ストリーム内にエラー番号を記録する。

【エラー】

fgetc() 参照。

【関連項目】

fopen, fscanf, fgetc, feof, ferror

8.20.16 fwrite - バイナリストリームの出力

【書式】

```
#include <stdio.h>

size_t fwrite(const void* ptr, size_t size, size_t nmemb, FILE* stream);
```

【解説】

fwirte() は、ptr が指す領域から、要素サイズが size バイトのデータを nmemb 個、stream で指定した
ストリームに書き込む。
stream のファイル位置は、書き込めたデータのバイト数だけ進められる。

【戻値】

fwrite() は、書込みに成功した要素数を返す。書込みエラーが発生した場合は、この値は nmemb より小さくなる。
size か nmemb が 0 の場合は、0 を返し、stream の状態は変化しない。
書込みエラーが発生した場合は、ストリーム内にエラー番号を記録する。

【エラー】

fputc() 参照。

【関連項目】

fopen, fprintf, fputc, ferror

8.20.17 fflush - ストリームのフラッシュ

【書式】

```
#include <stdio.h>

int fflush(FILE* stream);
```

【解説】

fflush() は、sream で示される出力ストリームに関して、未書込みのデータがあればファイルに書き込む(フラッシュする)。

stream が NULL の場合、fflush() は、開いているすべてる出力ストリームに関して、フラッシュする。

【戻値】

fflush() は、成功した場合 0 を返す。
エラーが発生した場合、ストリーム内にエラー番号を記録し、EOF を返す。

【エラー】

エラーが発生した場合、ferror() を呼び出すと、以下を返すことがある:

EAGAIN	ストリームに対応するファイルディスクリプタに対し、O_NONBLOCK フラグがセットされていて、書込みにより待ちが発生する
EBADF	ストリームに対応するファイルディスクリプタが不正
EFBIG	ファイルサイズの制限を越える
EINTR	fs_break() による中止
EIO	I/Oエラー
ENOSPC	デバイスの容量不足

【関連項目】

setbuf, fopen

8.20.18 fprintf, printf, snprintf, sprintf - 書式付き出力

【書式】

```
#include <stdio.h>

int fprintf(FILE* stream, const char* format, ...);
int printf(const char* format, ...);
int snprintf(char* str, size_t size, const char* format, ...);
int sprintf(char* str, const char* format, ...);
```

【解説】

fprintf() は、format で示す書式にしたがって変換を行い、stream で指定した出力ストリームに書込みを行う。
printf() は、出力ストリームが標準出力に固定される。
sprintf() は、出力をヌル文字で終わる文字列として str で指定した領域に書き込む。
ユーザは、str が指す領域が十分な大きさがあることを保証する必要がある。
snprintf() は、sprintf() と等価であるが、size が str が指すバッファのバイト数を指定している。
size が 0 の場合、出力は行われず str は NULL でもよい。そうでない場合、size-1 を超える出力バイトは捨てられ、str に実際に出力された文字の最後にヌル文字が書き込まれる。
sprintf() または snprintf() の出力で、重なった領域に対するコピーが発生した場合の結果は未定義とする。
(例えば、出力先と引数で同じ領域を参照している場合等)

fprintf(), printf(), snprintf(), sprintf() は、format で示す書式にしたがって、後の引数を変換して出力する。

format は、0 個以上の指令からなる:
単純に出力ストリームにコピーされる通常文字、および、各々が 0 個以上の引数を取り出すことを指示する変換指定である。

format に対する引数が不足している場合、結果は未定義とする。引数が余ったまま format を使い尽くすと、余分な引数は関数呼び出し時に評価するだけで、これらの関数の中では無視される。

書式文字列が % 形式の変換指定を含む場合、各変換指定は引数リスト中の最初の未使用の引数を順に使用する。

各変換指定は '%' 文字で始まり、以下の1～5に列挙したものがその順番に現れる:

変換指定の形式
%[フラグ][最小フィールド幅][.精度][長さ修飾子]変換指定子

1. フラグ(順不同、省略可)
フラグは、0 個以上の変換指定の意味を修飾する文字である。

2. 最小フィールド幅(省略可)
最小フィールド幅は、最小のフィールド幅を指定する10進整数文字列か、〈アスタリスク〉('*') である。
変換された値が、最小フィールド幅より小さい場合、デフォルトで〈空白〉文字が左側に埋められる。
最小フィールド幅に左寄せフラグ('-')が与えられた場合、〈空白〉は右側に埋められる。
最小フィールド幅を〈アスタリスク〉('*')で指定した場合は、int 型の引数で最小フィールド幅を指定する。
最小フィールド幅に対応した引数が負の場合は、'-' フラグ付きの正のフィールド幅とみなす。

3. 精度(省略可)
精度は、〈ピリオド〉('.') の後に、10進数字列が続く形式か、〈アスタリスク〉('*') である。
10進数字列は省略可能で、空の場合は 0 とみなされる。
精度が他の変換指定とともに使われた場合の動作は未定義とする。
精度を〈アスタリスク〉('*')で指定した場合は、int 型の引数で精度を指定する。
精度に対応した引数が負の場合は、精度は省略されたものとみなす。

4. 長さ修飾子
長さ修飾子は、変換指定子が示す型の長さを指定する文字である。

5. 変換指定子
変換指定子は、適用する変換の型を指定する文字である。

〈アスタリスク〉('*') で、フィールド幅、精度を指定した場合、変換指定で変換される引数の前に、
最小フィールド幅、精度に対応する引数は、フィールド幅、精度の順に提供しなければならない。

フラグ文字とその意味:

-
変換結果をフィールド内で左詰めする。
このフラグが無い場合、変換結果はフィールド内で右詰めする。

+
符号付き変換結果は常に符合('+' または '-')付きとする。
このフラグが無い場合、変換結果は負の値の場合のみ符号が付けられる。

〈空白〉
符号付き変換の先頭文字が符号ではない場合、または、符号付き変換の結果が空になった場合、
〈空白〉を一つ結果の前に置く。これは、〈空白〉および '+' フラグの両方を指定した場合、
〈空白〉フラグは無視されることを意味する。

#
値を代替形式で変換することを示す。
o 変換指定子の場合:
変換結果の先頭文字が 0 でない場合、0 を追加する。必要なら精度を増やす。

x, X 変換指定子の場合:
　　変換結果が 0 以外の場合、結果の先頭に 0x または 0X(X変換の場合) を追加する。
a, A, e, E, f, F, g, G 変換指定氏の場合:
　　小数点の後に数字がなくても常に小数点を出力する。
g, G 変換指定子の場合:
　　末尾の 0 を変換結果から削除しない。
他の変換指定子に対して動作は未定義とする。

0
d, i, o, u, x ,X, a, A, e, E, f, F, g, G 変換に対し、無限大またはNaNの変換を除き、
フィールド幅を埋めるのに、空白の変わりに(符号または基数表示に続く)先頭に連続した0を用いる。
'0' と '-' の両方のフラグを用いた場合、'0' フラグは無視する。
d, i, o, u, x, X 変換に対し精度が指定されている場合、'0' フラグを無視する。
その他の変換指定に対し、動作は未定義とする。

精度とその意味:

d, i, o, u, x, X 変換指定子の場合:
　　出力する最小の桁数。
　　デフォルトは1。

a, A, e, E, f, F 変換指定子の場合:
　　小数点部の桁数。
　　デフォルトは6。

g, G 変換指定子の場合:
　　最大の有効桁数。
　　デフォルトは6。

s 変換指定子の場合:
　　出力する最大文字数(バイト数)。

長さ修飾子とその意味:

hh
d, i, o, u, x, X 変換指定子が続いた場合、変換する引数が signed char または unsigned char であることを示す。
(整数引数は、引数拡張規則にしたがって拡張されているが、その値を書式整形に先立ち signed char または unsigned char 型に変換する)。
n 変換指定が続いた場合、変換する引数が signed char へのポインタであることを示す。

h
d, i, o, u, x, X 変換指定子が続いた場合、変換する引数が short または unsigned short であることを示す。
(整数引数は、引数拡張規則にしたがって拡張されているが、その値を書式整形に先立ち short または unsigned short 型に変換する)。
n 変換指定が続いた場合、変換する引数が short へのポインタであることを示す。

l
d, i, o, u, x, X 変換指定子が続いた場合、変換する引数が long または unsigned long であることを示す。
n 変換指定が続いた場合、変換する引数が long へのポインタであることを示す。
a, A, e, E, f, F, g, G 変換指定が続いた場合、この修飾子は無視される。

ll
d, i, o, u, x, X 変換指定子が続いた場合、変換する引数が long long または unsigned long long であることを示す。
n 変換指定が続いた場合、変換する引数が long long へのポインタであることを示す。

j
d, i, o, u, x, X 変換指定子が続いた場合、変換する引数が intmax_t または uintmax_t であることを示す。
n 変換指定が続いた場合、変換する引数が intmax_t へのポインタであることを示す。

z
d, i, o, u, x, X 変換指定子が続いた場合、変換する引数が size_t または対応する符号付き整数型であることを示す。
n 変換指定が続いた場合、変換する引数が size_t または対応する符号付き整数型へのポインタであることを示す。

t
d, i, o, u, x, X 変換指定子が続いた場合、変換する引数が ptrdiff_t または対応する符号無し

整数型であることを示す。
n 変換指定が続いた場合、変換する引数が ptrdiff_t または対応する符号無し整数型へのポインタであることを示す。

L
a, A, e, E, f, F, g, G 変換指定子が続いた場合、変換する引数が long double であることを示す。

上記した以外の変換指定に対して長さ修飾子を指定した場合の動作は未定義とする。

変換指定子とその意味:

d,i
int 型引数を、″[-]dddd″ 形式の符号付き10進数字に変換する。
精度は現れる最小桁数を指定する。
値がより少ない桁数で表現できる場合は、その精度まで先行する 0 で埋める。
デフォルトの精度は 1 である。精度 0 で変換結果が 0 の場合は、何も出力されない。

o
unsigned 型引数を、″dddd″ 形式の符号無し8進数字に変換する。
精度は、現れる最小桁数を指定する。
値がより少ない桁数で表現できる場合、その精度まで先頭から 0 で埋める。
デフォルトの精度は 1 である。精度 0 で変換結果が 0 の場合、何の文字も出力されない。

u
unsigned 型引数を、″dddd″ 形式の符号無し10進数字に変換する。
精度は、現れる最小桁数を指定する。
値がより少ない桁数で表現できる場合、その精度まで先頭から 0 で埋める。
デフォルトの精度は 1 である。精度 0 で変換結果が 0 の場合、何の文字も出力されない。

x
unsigned 型引数を、″dddd″ 形式の符号無し16進数字に変換する(10～15を表す文字は ″abcdef″ を用いる)。
精度は、現れる最小桁数を指定する。
値がより少ない桁数で表現できる場合、その精度まで先頭から 0 で埋める。
デフォルトの精度は 1 である。精度 0 で変換結果が 0 の場合、何の文字も出力されない。

X
文字 ″abcdef″ の代わりに大文字の ″ABCDEF″ を使うことを除き、x 変換指定子と等価である。

f,F
double 型引数を、″[-]ddd.ddd″ 形式の10進数字表記に変換する。ここで、小数点の後の桁数は、精度指定に一致する。精度を省略すると、6 とみなす。
精度が 0 で、'#' フラグが指定されていない場合、小数点は現れない。
小数点が現れる場合は、その前に一桁以上の数字が現れる。
下位の桁は、T2EXリファレンス実装では四捨五入するが、実装が定義した別な丸めを行ってもよい。

無限大の double 型引数は、T2EXリファレンス実装では、″[-]inf″ に変換されるが、
実装によっては ″[-]infinity″ に変換してもよい。
NaN を表す double 型引数は、″[-]nan(<n文字列>)″ または ″[-]nan″ に変換される。
<n文字列> については、nan() を参照。
F 変換指定子は、″inf″, ″infinity″, ″nan″ の変わりに、各々 ″INF″, ″INFINITY″, ″NAN″を生成する。

e,E
double 型引数を、″[-]d.ddde±dd″ の形式に変換する。ここで、引数が0でない場合、小数点の前に一桁の数字が存在し、小数点後の数値桁数は精度で指定した桁数となる。
精度を省略すると 6 とみなす。
精度がゼロで、'#' フラグが存在しない場合、小数点は現れない。
下位の桁は、実装が定義した方法により丸められる。
E 変換指定子は、指数部の先頭文字として 'e' の代わりに 'E' が使われる。
指数部は常に 2 桁以上である。値が 0 の場合、指数は「00」になる。

無限大および NaN を表す double 引数は、f および F 変換指定子の場合と同じ方法で変換される。

g,G
double 型の引数を、変換された値と精度に応じて、f または e (G 変換指定子の場合は F または E) 形式に変換する。
小数点後の数字の桁数は、精度で指定した桁数となる。
精度に 0 を指定した場合、1 とみなされる。
変換された値によって、e または f 形式のどちらかになる。
変換結果の指数部が -4 より小さいか、精度以上の場合 e 形式が使われる。
変換結果の小数部の末尾の 0 は、取り除かれる。
小数点文字が出力されるのは、少数点後に数字が一つ以上ある場合か、# フラグが指定された場合だけである。

無限大および NaN を表す double 引数は、f および F 変換指定子の場合と同じ方法で変換される。

a,A
double 型の引数を、″[-]0xh.hhhhp±d″ の形式に変換する。ここで、小数点の前に16進数一桁が存在し、小数点後の16進桁数は精度で指定した桁数となる。
少数点の前の数字は、正規化された浮動小数点数の場合は0以外となるが、正規化されていない場合は不定である。

精度指定が無く、FLT_RADIX が 2 のべき乗の場合、精度は値を正確に表現するのに必要な値となる。
精度指定が無く、FLT_RADIX が 2 のべき乗以外の場合、精度は double 型の値を正確に表現できる値となる。
最後に 0 が連続する場合は、連続した 0 は省略される。

精度が 0 で '#' フラグが指定されていない場合、小数点は現れない。
a 変換指定子に対しては、文字 ″abcdef″ が使われ、A 変換指定子に対しては大文字の ″ABCDEF″ が変換に使われる。
A 変換は、'x' および 'p' の代わりに、'X' 変換の形式で、数字を生成する。
指数は、常に一桁以上で、2 の指数を10進表現するのに必要な最小の桁数となる。

無限大および NaN を表す double 引数は、f および F 変換指定子の場合と同じ方法で変換される。

c
int 型引数を unsigned char に変換し、その結果の文字(1バイト)を出力する。

s
引数は、char 型の配列を指しているものとする。配列の先頭から終端のヌル文字までを(ヌル文字は含めずに)出力する。
精度が指定された場合は、精度を超えて文字を出力しない。
精度が指定されていないか配列のサイズより大きい場合、アプリケーションは配列がヌル文字を含むことを保証する必要がある。

p
引数は、void 型へのポインタでなければならない。
ポインタ値を %#x または %#lx で変換したかのように16進数に変換する。

n
引数は、整数へのポインタでなければならない。この fprintf() の呼び出しで、これまでに出力されたバイト数をその整数に格納する。
引数の変換は行わない。

%
文字 '%' を出力する。完全な変換指定子は %% となる。

変換指定子が上記形式のどれにも一致しない場合、動作は未定義とする。
引数が変換指定に対応した正しい型でない場合、動作は未定義とする。

最小フィールド幅が存在しないか小さくても、フィールドを切り詰めることはない。
変換結果が最小フィールド幅より大きい場合、フィールドは変換結果を含むまで拡張される。
fprintf() および printf() で生成される文字は、それぞれ、あたかも fputc() および putc() が呼ばれたようにして出力される。
したがって fputc() で発生し得るエラーが、fprintf() でも発生し、putc() で発生し得るエラーが printf() でも発生する。

a および A 変換指定に対し、FLT_RADIX が2のべき乗ではなく、結果が与えられた精度で正確に表現できない場合、結果は、与えられた精度の16進浮動小数点形式の二つの隣接する数値のどちらか一つでなければならない。
その場合、誤差は現在の丸め方向に対し、正しい符号をもたなければならない。

e, E, f, F, g および G 変換指定子に対し、有効桁数が DECIMAL_DIG 以下の場合、結果は正しく丸められなければならない。

【戻値】

fprintf() および printf() は、成功した場合、出力したバイト数を返す。

sprintf() は、成功した場合、s に書き出した終端のヌル文字を含まないバイト数を返す。

snprintf() は、成功した場合、size が十分大きい場合に str に書き出されるはずの終端のヌル文字を含まないバイト数を返す。

出力エラーが発生した場合、printf(), fprintf(), snprintf(), sprintf() は、負の値を返す。

snprintf() は、size の値が 0 の場合何も書き出さないが、size が十分大きい場合に
str に書き出されるはずの終端のヌル文字を含まないバイト数を返す。
このとき str は NULL でもよい。

【エラー】

エラーが発生した場合、ferror() を呼び出すと、以下を返すことがある:

fputc() で発生するエラー番号
EOVERFLOW　　オーバーフロー
　　- snprintf()で、size の値が INT_MAX より大きい
　　- 最後のヌル文字を除いた出力を保持するのに必要なバイト数が INT_MAX を超えた

【関連項目】

fputc, fscanf

【補足事項】

マルチバイト文字のライブラリを持たないため、wchar_t とマルチバイトの変換に伴う指定はない。

8.20.19 vfprintf, vprintf, vsnprintf, vsprintf - 可変個の引数リストによる書式付き出力

【書式】

```
#include <stdio.h>

int vfprintf(FILE *stream, const char *format, va_list ap);
int vprintf(const char *format, va_list ap);
int vsnprintf(char *str, size_t size, const char *format, va_list ap);
int vsprintf(char *str, const char *format, va_list ap);
```

【解説】

vfprintf(), vprintf(), vsnprintf(), vsprintf() は、可変個の引数の代わりに、va_list 型の
引数リストを引数にとる点を除き、各々 fprintf(), printf(), snprintf(), sprintf() と等価である。

vfprintf(), vprintf(), vsnprintf(), vsprintf() は、va_end マクロを呼び出さない。
vfprintf(), vprintf(), vsnprintf(), vsprintf() は、va_arg マクロを呼び出すため、終了時の ap の値は不定
となる。

【戻値】

fprintf() を参照。

【エラー】

fprintf() を参照。

【関連項目】

fprintf

8.20.20 fscanf, scanf, sscanf - 書式付き入力変換

【書式】

```
#include <stdio.h>

int fscanf(FILE* stream, const char* format, ...);
int scanf(const char* format, ...);
int sscanf(const char* str, const char* format, ...);
```

【解説】

fscanf() は、stream で指定した入力ストリームから読み込みを行う。
scanf() は、標準入力から読み込みを行う。
sscanf() は、str で指定した文字列から読み込みを行う。

これらの関数は、指定したストリームから1バイトずつ読込み、それらを format で示す書式にしたがって変換を行い、
結果を引数が示す領域に格納する。

これらの関数は、format で指定する制御文字列、および変換した入力の格納場所を示すポインタを可変個の引数として持つ。
format で指定される書式が必要とする引数の数よりも、関数が呼び出された時の実際の引数の数が少ない場合、結果は未定義とする。
引数が余ったまま、format を使い尽くすと、余分な引数は、関数呼び出し時に評価するだけで、
fscanf(), scanf(), sscanf() の中では無視される。

fscanf() は、あたかも fgetc() が呼ばれたかのようにしてストリームから入力を行う。
したがって fgetc() で発生し得るエラーが、fscanf() でも発生する。

format は、0 個以上の指令からなる文字列である。
各指令は、以下の(a),(b),(c)のいずれか一つである:

(a) 一つ以上の空白文字(〈スペース〉, 〈タブ〉, 〈改行〉, 〈垂直タブ〉, または 〈改ページ〉)
(b) ('%' でも空白文字でもない)通常の文字
(c) 変換指定

fscanf() は、format 内の各指令を順に実行する。指令に失敗するとそこで関数は終了する。
指令の失敗は、(利用可能な入力が無い)入力の失敗、または、(不適切な入力による)照合の失敗に分けられる。

(a) の一つ以上の空白文字からなる指令は、入力がなくなるか、非空白文字に達するまで読み飛ばされる。
このとき、到達した非空白文字は、未読込み状態となる。

(b) の通常文字からなる指令は、以下のように実行される:
次の文字が入力から読み込まれ、指令を構成する文字と比較される。
一致した場合は、その文字を読み飛ばす。
一致しなかった場合は、指令は照合の失敗となり、その文字も含めた以後の文字は読み取られずに残る。
ファイル終端、または読込みエラーが発生した場合、文字は読み込まれずに、指令は入力の失敗となる。

(c) の変換指定は、'%' 文字で始まり、その後に以下のものが番号順に続く:

1. 代入抑止文字 '*' (省略可能)
2. 最大フィールド幅を指定するゼロでない10進整数(省略可能)
3. 受け入れるオブジェクトのサイズを指定する長さ修飾子(省略可能)
4. 適用する変換のタイプを指定する変換指定文字

変換指定の形式
　　%[代入抑止文字][最大フィールド幅][長さ修飾子]変換指定子

1. 代入抑止文字(省略可)
   代入抑止文字は、〈アスタリスク〉('*') である。

2. 最大フィールド幅(省略可)
   最大フィールド幅は、最大のフィールド幅を指定する0でない10進整数文字列である。

3. 長さ修飾子(省略可)
   長さ修飾子は、変換指定子が示す型のサイズを指定する文字である。

4. 変換指定子
   変換指定子は、適用する変換の型を指定する文字である。

変換指定からなる指令は、各変換指定文字に対し、入力と照合すべき文字列の集合を定義する。

変換指定は、以下のステップで実行される。

変換指定が '[' 'c' 'C' 'n' のいずれか一つを含まないならば、入力内の空白文字は読み飛ばす。

変換指定が n を含まないならば、まず一つの項目が入力から読み込まれる(入力項目)。
入力項目とは、入力の先頭から変換指定で示される照合すべき文字列集合と照合して、最大フィールド幅に達するまでに一致した最長の入力文字列と定義する。

入力項目の直後のバイトは、(もしあれば)読み込まれずに残る。
入力項目の長さが 0 の場合、変換指定の実行は失敗する。
この状況は、もし、ファイル終端、または読込みエラーといった入力失敗でない場合、照合の失敗である。

% 変換指定の場合を除き、入力項目(%n 変換指定の場合は入力バイト数)は、変換指定で指定された型に変換される。
入力項目が、変換指定が示す照合文字列集合に含まれない時、変換指定の実行は失敗する。この状況は、照合の失敗である。

'*' による代入抑止が指示されていない場合、変換指定が % で開始されている場合、変換指定に応じた変換を行い、変換結果は、format 引数に続くまだ変換結果を受け取っていない最初の引数が指すオブジェクトに格納される。
このオブジェクトが、適切な型ではないか、変換結果が引数が提供する領域では表現できない場合、動作は未定義とする。
'*' による代入抑制が指示された場合は、変換指定に応じた変換は行うが、ポインタ引数は使用せず、変換結果は廃棄する。

長さ修飾子とその意味:

hh
: d, i, o, u, x, X, n 変換指定子が続いた場合、引数が signed char または unsigned char へのポインタ型であることを示す。

h
: d, i, o, u, x, X, n 変換指定子が続いた場合、引数が short または unsigned short へのポインタ型であることを示す。

l
: d, i, o, u, x, X, n 変換指定子が続いた場合、引数が long または unsigned long へのポインタ型であることを示す。
a, A, e, E, f, F, g, G 変換指定子が続いた場合、引数が double へのポインタ型であることを示す。

ll
: d, i, o, u, x, X, n 変換指定子が続いた場合、引数が long long または unsigned long long へのポインタ型であることを示す。

j
: d, i, o, u, x, X, n 変換指定子が続いた場合、引数が intmax_t または uintmax_t へのポインタ型であることを示す。

z
: d, i, o, u, x, X, n 変換指定子が続いた場合、引数が size_t へのポインタ型であることを示す。

t
: d, i, o, u, x, X, n 変換指定子が続いた場合、引数が ptrdiff_t または対応する unsigned 型へのポインタ型であることを示す。

L
: a, A, e, E, f, F, g, G 変換指定子が続いた場合、引数が long double へのポインタ型であることを示す。

上記以外の変換指定子に長さ修飾子が現れた場合、動作は未定義とする。

変換指定子とその意味:

d
: 符号付き10進整数に変換する。
base を10とした strtol() の変換対象と同じ形式の符号付き10進整数と一致する。
長さ修飾子がない場合、対応する引数は int 型へのポインタでなければならない。

i
: 符号付き8,10,16進整数に変換する。
base を0とした strtol() の変換対象と同じ形式の符号付き整数と一致する。
長さ修飾子がない場合、対応する引数は int 型へのポインタでなければならない。

o
: 符号付き8進整数に変換する。
base を8とした strtol() の変換対象と同じ形式の符号付き8進整数と一致する。
長さ修飾子がない場合、対応する引数は unsigned 型へのポインタでなければならない。

u
	符号無し10進整数に変換する。
	base を10とした strtol() の変換対象と同じ形式の符号無し10進整数と一致する。
	長さ修飾子がない場合、対応する引数は unsigned 型へのポインタでなければならない。

x
	符号無し16進整数に変換する。
	base を16とした strtol() の変換対象と同じ形式の符号無し16進整数と一致する。
	長さ修飾子がない場合、対応する引数は unsigned 型へのポインタでなければならない。

a,e,f,g
	符号付き10進浮動小数点数に変換する。
	strtod() の変換対象と同じ形式の浮動小数点定数、無限大, または NaN と一致する。
	長さ修飾子がない場合、対応する引数は float 型へのポインタでなければならない。

s
	空白文字を含まない文字列に変換する。
	空白文字ではない、文字列と一致する。
	アプリケーションは、対応する引数がその文字列と終端のヌル文字を格納するのに十分な大きさの
	char, signed char 型または unsigned char 型の配列の先頭へのポインタであることを
	保証する必要がある。

[走査文字の並び],[^走査文字の並び]
	期待される文字の集合(走査文字集合)に変換する。
	期待される文字の集合(走査文字集合)からなる空でない文字列と一致する。
	この場合、通常の、空白文字のスキップは行われない。
	アプリケーションは、対応する引数が、その文字列と自動的に付く終端のヌル文字を格納するのに
	十分な大きさの char, signed char 型または unsigned char 型の配列の先頭へのポインタで
	あることを保証する必要がある。

	変換指定子は、format 文字列内の、対応する ']' を含めたそこまでの文字列を全て含んでいる。
	'[' の直後の文字が '^' ではない場合、'[' と ']' 間の文字列(走査文字の並び)は、
	走査文字集合を構成する。例えば [ABCD] と [A-D] は、同じ、A,B,C,D の文字集合を意味する。
	'[' の直後の文字が '^' の場合、走査文字集合は、'^' と ']' の間の走査文字の並びに現れない
	文字となる。
	変換指定子が "[]" または "[^]" で始まる場合、']' は走査文字の並びに含まれ、次の ']' までが、
	変換指定子となる。それ以外の場合は、最初の ']' で変換指定子は終わる。

	例えば、"[0-9]" は、'0'から'9' までの数字の集合を意味する。
	"[^]0-9-]" は、']', '0'から'9', '-' の3種類を除く全ての文字の集合を意味する。

c
	空白文字を含む文字に変換する。
	最大フィールド幅(最大フィールド幅の指定がないときは1)で指定した数の文字列と一致する。
	ヌル文字は追加されない。
	この場合、通常の、空白文字のスキップは抑制される。
	アプリケーションは、対応する引数がその文字列を受け入れるのに十分な大きさの
	char, signed char 型または unsigned char 型の配列へのポインタであることを保証する必要がある。

p
	ポインタに変換する。
	対応する fprintf(), printf(), snprintf(), sprintf() の %p 変換指定で生成されるポインタ値の
	文字列表現と一致する。
	アプリケーションは、対応する引数が void 型へのポインタへのポインタであることを
	保証する必要がある。
	入力項目の解釈は、実装定義である。
	入力項目が、同じプログラムの実行の間に、以前に変換された値の場合、結果のポインタは、その値と
	一致しなければならない。それ以外の場合、%p 変換の動作は、未定義とする。
	T2EXリファレンス実装では、ポインタの16進表現(%#x)を解釈する。NULL は、0x0 で表す。

n
	入力したバイト数に変換する。
	入力を消費しない。
	%n が来るまでの入力から読み込んだ文字数(バイト数)を、対応する引数に格納する。
	アプリケーションは、対応する引数が整数へのポインタであることを保証する必要がある。
	%n 変換指定を実行しても、この関数の戻値である入力項目数は増加しない。
	変換指定が、代入抑止文字か最大フィールド幅を含む場合、動作は未定義とする。

%
	'%'に変換する。
	一つの '%' 文字と一致する。変換も代入も行われない。完全な変換指定子は、%% である。

変換指定子が不正な場合、動作は未定義とする。

変換指定子 A, E, F, G, X は、各々 a, e, f, g, x と等価である。

入力中にファイル終端に達すると、変換は終了する。
%n を除く現在の変換指定と一致する(先行する空白文字を除き)文字が読み込まれる前にファイル終端に達した場合は、現在の変換指定の実行は入力失敗として終了する。
現在の変換指定の実行が照合の失敗で終了しない場合は、続く変換指定の実行は入力の失敗として終了する。

sscanf() で文字列の終端に達した場合は、fscanf() でファイル終端に達した場合と等価とみなす。

変換が、不適切な入力文字により終了した場合、その(不適切な)入力文字は読み込まれずに入力ストリームに残る。
変換指定で照合に一致しない(<改行>文字を含む)後続する空白は、読み取られずに入力ストリームに残る。

【戻値】

fscanf(), scanf(), sscanf() は、成功した場合、照合に成功して代入を行った入力項目数を返す。
この数は、最初に照合に失敗すると 0 となる。
照合または変換の失敗の前に、入力が終了すると EOF を返す。
入力ストリームのエラーが発生した場合、ストリーム内にエラー番号を記録し EOF を返す。

【エラー】

エラーが発生した場合、ferror() を呼び出すと、以下を返すことがある:

fgetc() で発生するエラー番号
ENOMEM　　メモリ不足
EINVAL　　引数が不足

【関連項目】

fgetc, fprintf

【補足事項】

マルチバイト文字のライブラリを持たないため、c, s 変換指定子に対し、l 長さ修飾は使えない。

8.20.21 vfscanf, vscanf, vsscanf - 可変個の引数リストによる書式付き入力

【書式】

```
#include <stdio.h>

int vfscanf(FILE* stream, const char* format, va_list ap);
int vscanf(const char* format, va_list ap);
int vsscanf(const char* str, const char* format, va_list ap);
```

【解説】

vfscanf(), vscanf(), vsscanf() は、可変個の引数の代わりに、stdarg.h で定義される引数リストで呼び出されることを除き、各々、fscanf(), scanf(), sscanf() と等価である。
vfscanf(), vscanf(), vsscanf() は、va_end マクロを呼び出さない。
vfscanf(), vscanf(), vsscanf() は、va_arg マクロを呼び出すため、終了時の ap の値は不定となる。

【戻値】

fscanf() 参照。

【エラー】

fscanf() 参照。

【関連項目】

fscanf

8.20.22 setbuf, setvbuf - ストリームのバッファの設定

【書式】

```
#include <stdio.h>

void setbuf(FILE* stream, char* buf);
int setvbuf(FILE* stream, char* buf, int mode, size_t size);
```

【解説】

setvbuf() は、stream で指定したストリームに関するバッファリングを buf に設定する。
setvbuf() は、ストリームをオープンした後、他の操作を行う前に実行する必要がある。
mode により、バッファリングのタイプを指定する。
        _IOFBF   入出力ともに、完全なバッファリングを行う。
        _IOLBF   入出力ともに、行単位のバッファリングを行う。
        _IONBF   入出力ともに、バッファリングを行わない。
buf が NULL でない場合、buf が指す size バイトの領域をストリームのためのバッファとして使用する。
buf が NULL の場合は、setvbuf() は、size バイトのバッファを内部で割り当てる。

setbuf() は、
buf が NULL でない場合、
        setvbuf(stream, buf, _IOFBF, BUFSIZ) と等価である。
buf が NULL の場合は、
        setvbuf(stream, buf, _IONBF, BUFSIZ) と等価である。

【戻値】

setvbuf() は、成功した場合 0 を返す。
mode の値が不正な場合、または実行に失敗した場合は、0 以外の値を返す。
setbuf()は、値を返さない。

【エラー】

エラーが発生した場合、ferror() を呼び出すと、以下を返すことがある:

EBADF   ストリームに対応するファイルディスクリプタが不正

【関連項目】

fopen

8.20.23 fopen, fopen_eno - ストリームのオープン

【書式】

```
#include <stdio.h>

FILE* fopen(const char* path, const char* mode);
FILE* fopen_eno(const char* path, const char* mode, errno_t* eno); // 追加機能
```

【解説】

fopen(), fopen_eno() は、ファイルをオープンしストリームと関連付ける。
path で示すファイル名のファイルを mode で示すオープンモードで開き、ストリームと関連付け、そのストリームを返す。

mode は、オープンモードを示す文字列で、以下で示す値を指定する。
        "r"
                テキストファイル読込みモードでオープンする。
        "rb"
                バイナリファイル読込みモードでオープンする。

"w"
テキストファイル書込みモードでオープンする。
ファイルが存在しない場合は、新規作成し、存在する場合は、サイズを 0 にする。
"wb"
バイナリファイル書込みモードでオープンする。
ファイルが存在しない場合は、新規作成し、存在する場合は、サイズを 0 にする。
"a"
テキストファイル追加書込みモードでオープンする。
ファイルが存在しない場合は、新規作成し、存在する場合は、
ファイルオフセットをファイル終端に設定する。
"ab"
バイナリファイル追加書込みモードでオープンする。
ファイルが存在しない場合は、新規作成し、存在する場合は、ファイルオフセットを
ファイル終端に設定する。
"r+"
テキストファイル更新(読み書き)モードでオープンする。
"rb+", "r+b"
バイナリファイル更新(読み書き)モードでオープンする。
"w+"
テキストファイル更新(読み書き)モードでオープンする。
ファイルが存在しない場合は、新規作成し、存在する場合は、サイズを 0 にする。
"wb+", "w+b"
バイナリファイル更新(読み書き)モードでオープンする。
ファイルが存在しない場合は、新規作成し、存在する場合は、サイズを 0 にする。
"a+"
テキストファイル追加更新(読み書き)モードでオープンする。
ファイルが存在しない場合は、新規作成し、存在する場合はファイルオフセットを
ファイル終端に設定する。
"ab+", "a+b"
バイナリファイル追加更新(読み書き)モードでオープンする。
ファイルが存在しない場合は、新規作成し、存在する場合はファイルオフセットを
ファイル終端に設定する。

文字 'b' は、バイナリモードであることを意味する。
'b' の指定がない mode はテキストモードを意味する。
テキストモード時の読み書きを特別扱いするシステムが存在するために設けられているが、
テキストモードの特別扱いは行わないので、'b' による効果の違いは発生しない。

mode に、上記以外の指定が行われた場合の動作は、未定義とする。

追加モード('a' で始まるモード)でオープンしたファイルに対する書込み操作は、fseek() を呼び出しても、常にファイル終端に追加する形で行われる。

更新モード('+' を含むモード)でファイルをオープンした時、そのストリームに対して入力も出力も行う事ができる。

・出力の後に入力を行う場合は、2つの処理の間に fflush() または、fseek(), fseek64(), fsetpos(), rewind() を呼び出す必要がある。
・入力の後に出力を行う場合は、2つの処理の間に fseek(), fseek64(), fsetpos(), rewid() を呼び出す必要がある。

オープンされた場合、コンソールのように対話型デバイスではない限り、ストリームは完全なバッファリングを行うモードになっている。ストリームのエラー情報およびファイル終端インジケータはクリアされる。

fopen_eno() は、エラーが発生した場合、eno が NULL でないとき、eno が指す領域にエラー番号を格納する。

fopen() は、fopen_eno(path, mode, NULL) と等価である。

【戻値】

fopen(), fopen_eno() は、成功した場合、ストリームに対応するポインタを返す。
エラーが発生した場合、 NULL を返す。

【エラー】

エラーが発生した場合、 eno に以下のエラー番号が設定される。

EACCES	読込み専用ファイルを書込みまたは更新モードでオープンしようとした
EINTR	fs_break() による中止
EISDIR	path がディレクトリで、mode が書込みまたは更新モード
EMFILE	オープン中のストリーム数が、FOPEN_MAX を超えた
ENAMETOOLONG	ファイル名が長すぎる - path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX)

- path 全体の長さが長すぎる(最大PATH_MAX)
ENFILE　　システムでオープン可能なファイル数を超えた
ENOENT　　ファイルが存在しない
- path のファイルが存在しない
- path が空文字列
ENOMEM　　メモリ不足
EROFS　　読込み専用ファイルシステム上のファイルを書込みオープンしようとした

【関連項目】

fclose, fdopen, freopen

8.20.24 fdopen, fdopen_eno - ストリームをオープンする

【書式】

```
#include <stdio.h>

FILE* fdopen(int fd, const char* mode);
FILE* fdopen_eno(int fd, const char* mode, errno_t* eno); // 追加機能
```

【解説】

fdopen(), fdopen_eno() は、既にオープン済みのファイルディスクリプタに関連付けられたストリームを生成し、そのストリームを返す。

mode の指定は、fopen(), fopen_eno() 参照。
ただし、'w' を使用した場合、ファイルサイズを 0 にすることはしない。

mode の指定は、fs_open(), fs_creat() でオープンしたファイルディスクリプタ fd の読み書きモードと互換でなければならない。
新しく作成されたストリームのファイル位置は、ファイルディスクリプタ fd のファイルオフセットの値に設定される。
ストリームのエラー情報およびファイル終端インジケータはクリアされる。

エラーが発生した場合、eno が NULL でなければ、eno が指す領域にエラー番号を格納する。

fdopen() は、fdopen_eno(fd, mode, NULL) と等価である。

【戻値】

fdopen(), fdopen_eno() は、成功した場合、ストリームに対応するポインタを返す。
エラーが発生した場合、 NULL を返す。

【エラー】

エラーが発生した場合、eno に以下のエラー番号が設定される。

EMFILE　　オープン中のストリーム数が、FOPEN_MAX を超えた
EBADF　　ファイルディスクリプタが不正
EINVAL　　mode が不正
ENOMEM　　メモリ不足

【関連項目】

fopen, fclose, freopen

8.20.25 freopen, freopen_eno - ストリームを再オープン

【書式】

```
#include <stdio.h>

FILE* freopen(const char* path, const char* mode, FILE* stream);
FILE* freopen_eno(const char* path, const char* mode, FILE* stream, errno_t* eno); // 追加機能
```

【解説】

freopen_eno() は、最初に fflush(stream) が呼ばれたように、stream で指定したストリームをフラッシュする。
ストリームのフラッシュに関する失敗は無視される。
path が NULL でない場合、freopen_eno() は、ストリームに関連づれられたファイルディスクリプタをクローズする。
ファイルディスクリプタのクローズに関する失敗は、無視される。
ストリームのエラー情報とファイル終端インジケータはクリアされる。

freopen_eno() は、次に path で指定した名前のファイルをオープンし、stream で指定したストリームに関連付ける。
mode は、この場合 fopen() とまったく同じに使われる。

最後のオープンが成功するかどうかに関わらず、最初に指定したストリームはクローズされる。

path が NULL の場合、freopen_eno() は、現在ストリームに結び付けられているファイル名が使われたかのように、
ストリームのモードを mode で指定したモードに変更しようとする。この場合、freopen_eno() の呼び出しが成功する場合、ストリームに関連付けられたファイルディスクリプタはクローズされる必要は無い。
どのような状況で、どのようなモード変更が許されているかは、実装定義である。
T2EXリファレンス実装では、モードの変更は許容しない。

freopen_eno() は、エラーが発生した場合、eno が NULL でない時、eno が指す領域にエラー番号を格納する。

freopen(path, mode, stream) は、freopen_eno(path, mode, stream, NULL) と等価である。

【戻値】

freopen(), freopen_eno() は、成功した場合、stream の値を返す。
エラーが発生した場合、 NULL を返す。

【エラー】

エラーが発生した場合、eno には以下のエラー番号が設定される。

EACCES	ファイルに対し、mode で指定したアクセスが許可がない
EBADF	ストリームに対応するファイルディスクリプタが不正
EINTR	fs_break() による中止
EISDIR	path がディレクトリで、mode に書込み要求を指定した
EMFILE	オープン中のストリーム数が、FOPEN_MAX を超えた
ENAMETOOLONG	ファイル名が長すぎる - path の長さが PATH_MAX を超えている - path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX)
ENFILE	システムでオープン可能なファイル数を超えた
ENOENT	ファイルが存在しない - path のファイルが存在しない - path が空文字列である
ENOTDIR	path のプレフィクス部にディレクトリではないものがある
EROFS	読込み専用ファイルシステム上のファイルを書込みオープンしようとした

【関連項目】

fopen, fdopen, fclose

8.20.26 fclose_eno, fclose - ストリームのクローズ

【書式】

```
#include <stdio.h>

int fclose(FILE* fp);
```

```
int fclose_eno(FILE* fp, errno_t* eno); // 追加機能
```

【解説】

fclose_eno() は、stream で指定したストリームをフラッシュし、ストリームに関連付けられたファイルをクローズする。
ストリームのバッファ内の未書込みデータはファイルに書き込まれ、バッファ内の未読込みデータは破棄される。
呼び出しの成功の如何にかかわらず、ストリームとファイルは関連付けは解除され、setbuf() または setvbuf() で
設定されたバッファは、ストリームから切り離される。バッファが自動で割り当てられていた場合は、解放される。

fclose_eno() は、エラーが発生した場合、eno が NULL でない場合、eno が指す領域にエラー番号を格納する。

fclose(fp) は、fclose_eno(fp, NULL) と等価である。

【戻値】

fclose_eno(), fclose() は、成功した場合、0 を返す
エラーが発生した場合、EOF を返す。

【エラー】

エラーが発生した場合、eno には以下のエラー番号が設定される。

EAGAIN	ストリームに対応するファイルディスクリプタに対し、O_NONBLOCK フラグがセットされていて、書込みにより待ちが発生する
EBADF	ストリームに対応するファイルディスクリプタが不正
EFBIG	ファイルサイズの制限を越える
EINTR	fs_break() による中止
EIO	I/Oエラー
ENOSPC	デバイスの容量不足

【関連項目】

fopen

8.21 stdlib.h

ヘッダ stdlib.h は、以下のマクロおよび型を定義する。

マクロ

NULL
: 空のポインタを表す定数マクロ。(void*) にキャストした値 0 の整数定数式に展開される。

RAND_MAX
: 乱数を返す関数 rand_r() が返す擬似乱数整数の最大値を示す。
この値は、少なくとも 32767 以上をとらなければならない。
T2EXリファレンス実装では、0x7fffffff とする。

型

div_t
: div() が返す構造体型を示す。
この構造体は、int 型の除算の商と剰余からなる。

ldiv_t
: ldiv() が返す構造体型を示す。
この構造体は、long 型の除算の商と剰余からなる。

lldiv_t
: lldiv() が返す構造体型を示す。
この構造体は、long long 型の除算の商と剰余からなる。

size_t
: sizeof 演算子の結果を表す符号無し整数型を示す。

wchar_t
: システムロケールが規定する最大の文字セットの全ての文字を区別できるコード値に必要な値の範囲を持つ整数型。
例えば、ユニコードをバイト列として表現する UTF-8 ではマルチバイト文字が使われるが、その最大バイト数を
表現できる整数型を意味する。
ヌル文字は、コード値 0 を持たなければならない。
文字セット中の各要素(UTF-8 であれば複数バイトの事がある)を、1文字の整数文字定数として用いる場合の型であり、ワイド文字型と呼ぶ。

関数

8.21.1 abort - システムの異常終了

【書式】

```
#include <stdlib.h>

void abort(void);
```

【解説】

abort() は、システムを異常終了させる。標準ではエラーメッセージの出力と tm_monitor() が実行されるが、setabort() を実行することでシステムの異常発生時の処理を変更することができる。

【戻値】

なし

【エラー】

なし

【関連項目】

setabort

8.21.2 abs, labs, llabs - 整数の絶対値

【書式】

```
#include <stdlib.h>

int             abs(int i);
long            labs(long i);
long long       llabs(long long i);
```

【解説】

abs(), labs(), llabs() は、整数 i の絶対値を計算する。結果が表現できない場合、動作は未定義とする。

【戻値】

整数の絶対値を返す。

【エラー】

なし

【関連項目】

fabs

8.21.3 atof - 文字列から double 型数値への変換

【書式】

```
#include <stdlib.h>

double  atof(const char *str);
```

【解説】

atof() は、文字列 str で示す浮動小数点文字列を double 値に変換する。
atof() は、エラーの扱いが異なる点を除き、以下と等価である。

```
strtod(str, (char**) NULL);
```

値が表現できない場合、動作は未定義とする。

【戻値】

atof() は、値が表現できた場合変換された値を返す。

【エラー】

なし

【関連項目】

strtod

8.21.4 atoi, atol, atoll - 文字列から整数値への変換

【書式】

```
#include <stdlib.h>

int             atoi(const char *str);
long            atol(const char *str);
long long       atoll(const char *str);
```

【解説】

atoi(), atol(), atoll() は、文字列 str で示す10進整数文字列を整数値に変換する。
atoi(), atol() は、エラーの扱いが異なる点を除き、以下と等価である。

```
strtol(str, (char**) NULL, 10);
```

atoll() は、エラーの扱いが異なる点を除き、以下と等価である。

```
strtoll(str, (char**) NULL, 10);
```

値が表現できない場合、動作は未定義とする。

【戻値】

atoi(), atol(), atoll() は、値が表現できた場合変換された値を返す。

【エラー】

なし

【関連項目】

strtol

8.21.5 bsearch - 二分木探索

【書式】

```
#include <stdlib.h>

void    *bsearch(const void *key, const void *base, size_t nel, size_t width,
                 int (*compare)(const void *, const void *));
```

【解説】

bsearch は, 先頭の要素を base が指す nel 個のオブジェクトの配列内で,
key が指すオブジェクトに一致する要素を探索する。その配列の各要素の大きさを width で指定する。
nel が 0 の場合、compare で示される比較関数は呼ばれず、何も見つからない結果となる。

compare で示される比較関数は、二つの引数を持つ関数で、key オブジェクト、配列の要素という順の引数で呼び出される。

アプリケーションは、比較関数で配列の内容を変更しないようにする必要がある。bsearch() は、比較関数を呼び出した間に配列要素の順序を変更する可能性があるが、各要素の内容は変更しない。

(配列内の現在位置に関係なく、width バイトからなる)同じオブジェクトが比較関数に複数回渡される場合、
結果は、互いに一致しなければならない。すなわち、同じオブジェクトは常に同じ方法で key と比較する。

比較関数は、key オブジェクトが配列要素より、小さい場合は負の整数を、一致する場合は0を、大きい場合は正の整数を返す必要がある。
アプリケーションは、指定した配列が、比較関数で決められる大小関係にしたがって小さい順にソートされているようにする必要がある。

【戻値】

bsearch() は、配列内の一致する要素へのポインタを返す。一致する要素がない場合 NULL を返す。
一致する要素が複数ある場合、どの要素が返るかは不定である。

【エラー】

なし

【関連項目】

lsearch, qsort

8.21.6 calloc - ユーザレベルメモリの割当

【書式】

```
#include <stdlib.h>

void    *calloc(size_t nelem, size_t elsize);
```

【解説】

calloc() は、elsize バイトの要素を nelem 個からなる配列のための未使用領域を割当て、全て 0 で初期化する。
割り当てられるメモリ領域の保護レベルは、ユーザレベルである。

割当てに成功した場合返すポインタは、適切に整列されており、いかなる型のオブジェクトへのポインタに代入しても
(領域を解放するか、再割当てがされるまで)その割り当てられた領域のオブジェクトの配列にアクセスできる。この割当て
によるポインタは、他のいかなるオブジェクトとも区別されたオブジェクトへのポインタである。

返されるポインタは、割り当てた領域の先頭を指している。領域を割り当てられなかった場合、 NULL を返す。
要求した領域のサイズが 0 の場合、動作は実装依存とする。返す値は NULL かユニークなポインタとする。
T2EXリファレンス実装では、領域のサイズが 0 の場合、NULL を返す。

【戻値】

nelem と elsize がともに 0 でない場合、割当に成功したら、割り当てた領域へのポインタを返す。
nelem と elsize のどちらかが 0 の場合、 NULL または free() に渡すことが出来るユニークなポインタを返す。
それ以外の場合 NULL を返す。

【エラー】

なし

【関連項目】

malloc, realloc, free

8.21.7 div, ldiv, lldiv - 整数の除算の商と剰余

【書式】

```
#include <stdlib.h>

div_t           div(int n, int d);
ldiv_t          ldiv(long n, long d);
lldiv_t         lldiv(long long n, long long d);
```

【解説】

div(), ldiv(), lldiv() は、n を d で割ったときの商(quot)と剰余(rem)を計算する。
割り切れない場合、結果の商は、代数的な商に最も近い小さい方の整数とする。
結果が表現できない場合の動作は、未定義とする。そうでなければ、quot * d + rem = n が成立する。

【戻値】

div(), ldiv(), lldiv() は、div_t, ldiv_t, lldiv_t 型の構造体に結果を返す。
div_t, ldiv_t, lldiv_t 型の構造体は以下の要素を含む。

quot	商: 関数に応じた int, long, long long 型
rem	剰余: 関数に応じた int, long, long long 型

【エラー】

なし

【関連項目】

ldiv

8.21.8 free - メモリの解放

【書式】

```
#include <stdlib.h>

void    free(void *ptr);
```

【解説】

free() は、ptr が指すメモリ領域を解放する。
ptr が NULL の場合、何もしない。
ptr が NULL でない場合は、以前に calloc(), malloc(), realloc() 関数で割り当てた領域でなければならない。
それ以外の場合、動作は未定義とする。

解放された領域を参照しているポインタの値は、未確定である。

【戻値】

free() は、値を返さない。

【エラー】

なし

【関連項目】

calloc, malloc, realloc

8.21.9 malloc - ユーザレベルメモリ領域の割当

【書式】

```
#include <stdlib.h>

void    *malloc(size_t size);
```

【解説】

malloc() は、size バイトのオブジェクトのための領域を割り当てる。領域の内容は不定とする。

割り当てられるメモリ領域の保護レベルは、ユーザレベルである。

割当てに成功した場合返すポインタは、適切に整列されており、いかなる型のオブジェクトへのポインタに代入しても(領域を解放するか、再割当てがされるまで)その割り当てられた領域のオブジェクトにアクセスできる。
この割当てによるポインタは、他のいかなるオブジェクトとも区別されたオブジェクトへのポインタである。
返されるポインタは、割り当てた領域の先頭を指している。領域を割り当てられなかった場合、 NULL を返す。
要求した領域のサイズが 0 の場合、動作は実装依存とする。このとき返す値は NULL かユニークなポインタとする。

【戻値】

size が 0 でない場合、割当てに成功するとその領域へのポインタを返す。
size が 0 の場合、 NULL または free() に渡すことが出来るユニークなポインタを返す。
それ以外の場合、 NULL を返す。
T2EXリファレンス実装では、size が 0 の場合、NULL を返す。

【エラー】

なし

【関連項目】

calloc, realloc, free

8.21.10 qsort - 配列のソート

【書式】

```
#include <stdlib.h>

void    qsort(void *base, size_t nel, size_t width, int (*compare)(const void *, const void *));
```

【解説】

qsort() は、先頭を base が指している enel 個のオブジェクトの配列をソートする。
各オブジェクトのバイトサイズは、width で指定される。
nel の値が 0 の場合、compare で指定された比較関数は呼び出されず並び替えも行われない。

アプリケーションは、比較関数が配列の内容を変更しないようにする必要がある。
qsort() は、比較関数の呼び出しの間に要素を並べ替えることがあるが、各要素の内容は変更しない。

(配列内の現在位置に関係なく width バイトからなる)同じオブジェクトが比較関数に複数回渡される場合、結果は互いに一致しなければならない。すなわち、それらは配列の全順序を定義する。

配列の内容は、比較関数にしたがって昇順にソートされる。compare は、比較関数のポインタである。
比較関数は、比較される要素を指す二つの引数で呼び出される。
比較関数は、最初の引数が二番目に対して、小さい場合負の整数を、等しい場合0を、大きい場合正の整数を返す必要がある。
二つの要素が等しい場合、ソートされた配列内での順序は不定とする。

【戻値】

qsort() は、値を返さない。

【エラー】

なし

8.21.11 rand_r - 擬似乱数整数列の生成

【書式】

```
#include <stdlib.h>

int     rand_r(unsigned int *seed);
```

【解説】

rand_r() は、[0, RAND_MAX] の範囲の擬似乱数整数列を計算する。

rand_r() は、 seed に擬似乱数整数列の初期値を設定する。rand_r() の返す擬似乱数整数列は、seed によって決まる。同じ seed では、同じ擬似乱数整数列となる。

【戻値】

rand_r() は、擬似乱数整数を返す。

【エラー】

なし

【関連項目】

drand48_r

8.21.12 drand48_r, lrand48_r, mrand48_r, srand48_r, seed48_r, lcong48_r - 一様分布する擬似乱数の生成

【書式】

```
#include <stdlib.h>

double  drand48_r(struct rand48_data *buffer);
long    lrand48_r(struct rand48_data *buffer);
long    mrand48_r(struct rand48_data *buffer);
void    srand48_r(long int seedval, struct rand48_data *buffer);
void    seed48_r(unsigned short int seed16v[3], struct rand48_data *buffer, unsigned short oldxi[3]);
void    lcong48_r(unsigned short int param[7], struct rand48_data *buffer);
```

【解説】

drand48_r(), lrand48_r(), mrand48_r(), srand48_r(), seed48_r(), lcong48_r() は、線形合同法と48ビット整数演算
を用いて擬似乱数を生成する。

drand48_r() は、[0.0, 1.0) の範囲に一様に分布する非負の倍制度浮動小数点値を返す。
lrand48_r() は、[0, 2 ^ 31) の範囲に一様に分布する非負の整数を返す(' ^ 'はべき乗を表す)。
mrand48_r() は、[-2 ^ 31, 2 ^ 31) の範囲に一様分布する符号付き整数を返す(' ^ 'はべき乗を表す)。

srand48_r(), seed48_r() および lcong48_r() は、その中の一つが、drand48_r(), lrand48_r() または mrand48_r()
を呼ぶ前に、rand48_data 構造体の初期化を行う関数である。

drand48_r(), lrand48_r(), mrand48_r(), srand48_r(), seed48_r(), lcong48_r() は、48ビット整数列 Xi を以下の線形合同式として生成することで動作している。
n 番目の X に対し、n+1 番目の X は、次の式で求める。

```
        Xn+1 = (a * Xn + c) mod m     n>=0
```

パラメータ m は 2 ^ 48 であるため、48ビット整数演算が行われている(' ^ 'はべき乗を表す)。
lcong48_r() が呼ばれない限り、乗算値 a と加算値 c は以下の値が使われる。

```
        a = 0x5DEECE66D
        c = 0xB
```

drand48_r(), lrand48_r(), mrand48_r() で返される値は、
最初に整数列内の次の48ビットXiを生成することで求められる。次に、戻値の型に応じた適切なビット数の数値が、
Xi の最上位ビットからコピーされ、戻値の型に変換される。

構造体 rand48_data は、以下のようなデータを含んでいる。

```
struct rand48_data {
        unsigned short  xi[3];          /* 現在の Xi 値 : xi[0] が最下位16bit */
        unsigned short  mult[3];        /* 乗算数 a : mult[0] が最下位16bit */
        unsigned short  add;            /* 加算数 c */
        /* 実装依存の要素 */            /* T2EXリファレンス実装ではこの要素はない */
};
```

乱数の計算には上記 rand48_data を用いる。したがって、最初に srand48_r(), seed48_r(), lcong48_r() を用いて、
rand48_data 構造を初期化する必要がある。

drand48_r(), lrand48_r() および mrand48_r() は、rand48_data で与えられた Xi, m , a を用いて、
        Xi+1 = (a * Xi + c) mod m
を計算する。生成された最後の48ビット値を次の Xi として buffer 内の xi[3] に格納する。

初期化関数 srand48_r() は、rand48_data の Xi の値を以下のように初期化する。
xi の上位32ビットを、 seedval の下位32ビットに設定する。
xi の下位16ビット(すなわち xi[0])は、0x330E に設定する。
mult, add は上記デフォルト値に設定する。

初期化関数、seed48_r() は、rand48_data の Xi の値を、 seed16v[3] で指定した値に設定する。
Xi の下位16ビット(xi[0])は、seed16v[0] の下位16ビットに設定する。
Xi の中間位16ビット(xi[1])は、seed16v[1] の下位16ビットに設定する。
Xi の上位16ビット(xi[2])は、seed16v[2] の下位16ビットに設定する。
加えて、Xi の以前の値(呼び出す前の xi の値)は、oldxi が指す領域にコピーする。
mult, add は上記デフォルト値に設定する。

初期化関数 lcong48_r() は、rand48_data の、初期 Xi、乗算数 a および加算数 c を指定した値に設定する。
 param[0-2] は Xi を、param[3-5] は乗算数 a を、そして、param[6] は16ビットの加算数 c を指定する。

【戻値】

drand48_r(), lrand48_r(), mrand48_r(), srand48_r(), seed48_r(), lcong48_r() は、上記説明で述べた値を返す。

【エラー】

なし

【関連項目】

rand_r

8.21.13 realloc - ユーザレベルメモリの再割当

【書式】

```
#include <stdlib.h>

void    *realloc(void *ptr , size_t size);
```

【解説】

realloc() は、ptr が指す保護レベルがユーザレベルのメモリオブジェクトのサイズを、size で指定したサイズに変更する。
メモリオブジェクトの内容は、旧サイズと size の何れか小さい方までは、変化しない。
メモリオブジェクトの移動が必要になった場合、オブジェクトに以前に割当てられていた領域は、解放される。
サイズを大きくした場合、オブジェクトの新たに割当てられた部分の内容は不定である。
サイズが 0 で、ptr が NULL でない場合、ptr が指す領域は解放される。
領域が割当てられなかった場合、オブジェクトは解放されずにそのまま残る。

ptr が NULL の場合、realloc() は、malloc(size) と等価である。

ptr が、以前に実行した calloc(), malloc() または realloc() の戻値ではない場合、または領域が free() ま

たは
realloc() により解放されている場合の動作は、未定義とする。

割当てに成功した場合返すポインタは、適切に整列されており、いかなる型のオブジェクトへのポインタに代入しても
(領域を解放するか、再割当てがされるまで)その割り当てられた領域のオブジェクトの配列にアクセスできる。
この割当てによるポインタは、他のいかなるオブジェクトとも区別されたオブジェクトへのポインタである。
返されるポインタは、割当てた領域を先頭を指している。領域を割当てられなかった場合、 NULL を返す。
割り当てられるメモリ領域の保護レベルは、ユーザレベルである。

【戻値】

realloc() は、size が 0 以外で割当に成功した場合、割当てられた領域へのポインタを返す。
size が 0 の場合、 NULL または free() に渡すことが出来るユニークなポインタを返す。
メモリが不足した場合、NULL を返す。
T2EXリファレンス実装では、size が 0 の場合、NULL を返す。

【エラー】

なし

【関連項目】

calloc, malloc, free

8.21.14 realpath_eno - パス名の正規化

【書式】

```
#include <stdlib.h>

char    *realpath_eno(const char *path, char *resolved_path, errno_t* eno);
```

【解説】

realpath_eno() は、path で指定されたパス名の文字列から、'.' や '..' および余計な '/' を含まない絶対パス名を生成する。
生成されたパス名は、resolved_path で示される最大 PATH_MAX バイトの領域にヌル終端された文字列として格納される。
resolved_path が NULL の場合、生成されたパス名はヌル文字で終わる文字列として malloc() で割り当てたバッファに格納される。

エラーが発生した場合、eno が NULL でない場合、eno が指す領域にエラー番号を格納する。

【戻値】

realpath_eno() は、成功した場合、生成されたパス名を含むバッファのポインタを返す。
失敗した場合、 NULL を返す。

resolved_path が NULL の場合、realpath_eno() が返したポインタは free() に渡すことが出来る。
resolved_path が NULL ではない場合、realpath_eno() が失敗した場合、resolved_path が指すバッファの内容は不定とする。

【エラー】

eno が指す領域に以下のエラー番号が格納されることがある。

EINVAL	path が NULL
EIO	I/Oエラー
ENAMETOOLONG	ファイル名が長すぎる - path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX) - path 全体の長さが長すぎる(最大PATH_MAX)
ENOENT	path のファイルが存在しない
ENOTDIR	path のプレフィクス部にディレクトリではないものがある

【関連項目】

realpath2_eno

8.21.15 realpath2_eno - パス名の正規化

【書式】

```
#include <stdlib.h>

char *realpath2_eno(const char *path1, const char* path2, char *resolved_path, errno_t *eno);
```

【解説】

realpath2() は、path1 を現在の作業ディレクトリとみなし、path2 で指定されたパス名の文字列から、
'.' や '..' および余計な '/' を含まない絶対パス名を生成する。
生成されたパス名は、resolved_path で示される最大 PATH_MAX バイトの領域に、ヌル終端された文字列として格納される。
resolved_path が NULL の場合、生成されたパス名は、ヌル文字で終わる文字列として malloc() で割り当てたバッファに格納される。
エラー発生時、eno が NULL でない場合、eno が指す領域にエラー番号を格納する。

【戻値】

realpath2_eno() は、成功した場合、生成されたパス名を含むバッファのポインタを返す。
失敗した場合、 NULL を返す。

resolved_path が NULL の場合、realpath2_eno() が返したポインタは free() に渡すことが出来る。
resolved_path が NULL ではない場合、realpath2_eno() が失敗した場合、resolved_path が指すバッファの内容は不定とする。

【エラー】

eno が指す領域に以下のエラー番号が格納されることがある。

EINVAL	path が NULL
EIO	I/Oエラー
ENAMETOOLONG	ファイル名が長すぎる - path に含まれるディレクトリやファイル名部分が長すぎる(最大NAME_MAX) - path 全体の長さが長すぎる(最大PATH_MAX)
ENOENT	path のファイルが存在しない
ENOTDIR	path のプレフィクス部にディレクトリではないものがある

8.21.16 setabort - システムの異常終了処理を行う関数の登録

【書式】

```
#include <stdlib.h>

int setabort(void (*func)(void));
```

【解説】

abort() により発生したシステムの異常終了時に実行される関数 func を登録する。
すでに登録されている関数の登録は解除され、最後に登録された関数のみが有効となる。

func が NULL である場合、ライブラリ標準の異常処理関数の登録を意味する。
標準の異常処理関数ではエラーメッセージの出力と tm_monitor() の実行が行われる。

【戻値】

setabort() は、常に成功し 0 を返す。

【エラー】

なしT2EXリファレンス実装
abort

【補足事項】

この関数は、T2EX の独自関数である。

8.21.17 strtod, strtof, strtold - 文字列から double, float, long double 型数値への変換

【書式】

```
#include <stdlib.h>

double          strtod(const char *nptr, char **endptr);
float           strtof(const char *nptr, char **endptr);
long double     strtold(const char *nptr, char **endptr);
```

【解説】

strtod(), strtof(), strtold() は、nptr が指す文字列の先頭部分を、それぞれ double, float, long double 型の値に変換する。

strtod(), strtof(), strtold() は、最初に入力文字列を以下の三つの部分に分解する:
(1). isspace() が真となる空白文字の並び。これは、空でも良い。
(2). 浮動小数点定数、または無限大あるいは NaN を表現する変換対象文字列。
(3). ヌル文字を含む一つ以上の変換対象とならない文字列。

次に、(2)の変換対象文字列部を浮動小数点数値に変換し、結果を返す。

変換対象文字列部は、オプションの'+'または'-'の符号に続いて、以下のいずれかの形式を持つ:
・空でない10進数字列(中に小数点が一つあってもよい)と、オプションの指数部。
・"0x" または "0X" に続く空でない16進数字列(中に小数点が一つあってもよい)と、オプションの指数部。
・"INF" または "INFINITY"。大文字小文字は無視する。
・"NAN" または "NAN(<n文字列>)"。NAN 部分の大文字小文字は無視する。
　　<n文字列> については nan() 参照。

指数部は、以下の何れかの形式を持つ:
・10進指数部は以下の形式を持つ:
　　'e' または 'E' の後に、オプションの'+'または'-'の符号、その後に指数値を表す空でない数字列。
　　指数値は、10の何乗であるかを意味する。
・2進指数部は以下の形式を持つ:
　　'p' または 'P' の後に、オプションの'+'または'-'の符号、その後に指数値を表す空でない数字列。
　　指数値は、2の何乗であるかを意味する。

変換対象文字列部は、最初の非空白文字で始まる、期待する形式の入力文字列の最長の並びとして定義される。
入力文字列が期待されている形式ではない場合、変換対象文字列は文字を含まない。

INF または INFINITY は、返す型で表現できる場合、無限大と解釈される。そうでない場合、返す型の範囲を超えた大きな浮動小数点数として扱う。

"NAN" または "NAN(<n文字列>)" は、戻値の型がサポートしている場合 NaN と解釈される。
それ以外の場合、それは期待された形式ではない変換対象文字列部である。

endptr が NULL でない場合、これらの関数は、最後の文字列へのポインタを endptr が指す領域に格納する。

変換対象文字列が16進形式で、FLT_RADIXが2のべき乗の場合、変換結果は正しく丸められる。

変換対象文字列が空または期待する形式ではない場合、変換は行われない。endptr が NULL でない場合、これらの関数は、nptr の値を endptr が指す領域に格納する。

【戻値】

strtod(), strtof(), strtold() は、成功した場合、変換した値を返す。

変換が行われなかった場合、0 を返す。

正しい値が表現できる値の範囲外の場合、HUGE_VAL, HUGE_VALF, または HUGE_VALL に符号をつけて返す。

正しい値がアンダーフローを起こす場合、戻値は、表現できるもっとも小さな正の値(通常 0)を返す。

【エラー】

なし

【関連項目】

strtol, fscanf, isspace

8.21.18 strtol, strtoll - 文字列から long, long long 型への変換

【書式】

```
#include <stdlib.h>

long            strtol(const char *str, char **endptr, int base);
long long       strtoll(const char *str, char **endptr, int base);
```

【解説】

strtol(), strtoll() は、str が指す文字列の先頭部分を、それぞれ long, long long 型の値に変換する。
strtol(), strtoll() は、最初に入力字列を三つの部分に分割する。
(1). isspace() が真となる空白文字の並び。これは、空でもよい。
(2). base で指定された基数での整数表現と解釈される、変換対象文字列。
(3). 一つ以上の変換対象とならない、入力文字列の最後の文字列。(ヌル文字列も含む)

strtol(), strtoll() は、(2)の変換対象文字列を整数に変換し、結果を返す。

base が 0 の場合、変換対象文字列は、10進、8進、または16進定数である。この場合、先頭に'+'または'-'の符号がついてもよい。
・10進定数は、0 以外の数字で始まる10進数字列である。
・8進定数は、'0' で始まる'0'から'7'の数字列である。
・16進定数は、 0x または 0X で始まる、10進数字または 'a'(または'A')から'f'(または'F')までの文字で 0 から 15 の値を持つ文字の列である。

base が 2 から 36 の場合、変換対象文字列の形式は、base で指定した基数を持つ数値を表現する文字または数字の並びであり、先頭に'+'または'-'の符号が付いてもよい。
'a'(または'A')から'z'(または'Z')は、10から35の値を表す。base 以下の値の文字のみ許される。

base が 16 の場合、0x または 0X の文字列が変換対象文字列の先頭に(符号がある場合は、その直後に)あってもよい。

変換対象文字列は、最初の非空白文字で始まる期待された形式の入力文字列の最長の並びとして定義される。
入力文字列が空か全て空白文字、または最初の非空白文字が符号でも数字でも数字を表現する文字でもない場合、変換対象文字列は空となる。

変換対象文字列が期待される形式で、base の値が 0 の場合、最初の数字で始まる文字の並びは、整数定数と解釈される。
変換対象文字列が期待される形式で、base の値が 2 から 36 の場合、base の値を変換の基数として用い、各文字は上記で説明した値を持つものとする。
変換対象文字列がマイナス符号で始まる場合、変換結果の値の符合を反転する。

endptr が NULL でない場合、これらの関数は、最後の文字列へのポインタを endptr が指す領域に格納する。

変換対象文字列が空または期待する形式ではない場合は、変換は行われない。endptr が NULL でない場合、これらの関数は、str の値を endptr が指す領域に格納する。

【戻値】

strtol(), strtoll() は、成功した場合、変換した値を返す。

変換が行われなかった場合、0 を返す。

正しい値が表現できる範囲外の場合、値の符号に応じて LONG_MIN, LONG_MAX, LLONG_MIN または LLONG_MAX を返す。

【エラー】

なし

【関連項目】

strtod, fscanf, isalpha

8.21.19 strtoul, strtoull - 文字列から unsigned long, unsigned long long 型への変換

【書式】

```
#include <stdlib.h>

unsigned long           strtoul(const char *str, char **endptr, int base);
unsigned long long      strtoull(const char *str, char **endptr, int base);
```

【解説】

strtoul(), strtoull() は、str が指す文字列の先頭部分を、それぞれ unsigned log, unsigned long long 型の値に変換する。
strtoul(), strtoull() は、最初に入力文字列を三つの部分に分割する。
(1). isspace() が真となるで空白文字列。これは、空でもよい。
(2). base で指定された基数での整数表現と解釈される、変換対象文字列。
(3). 一つ以上の認識されない文字の、ヌル文字も含めた、入力文字列の最後の文字列。

strtoul(), strtoull() は、(2)の変換対象文字列を符号無し整数に変換し、結果を返す。

base が 0 の場合、変換対象文字列は、10進定数、8進定数、または16進定数である。この場合、先頭に'+'または'-'の符号が付いてもよい。
・10進定数は、0 以外の数字で始まる10進数字列である。
・8進定数は、'0' で始まる'0'から'7'の数字列である。
・16進定数は、 0x または 0X で始まる、10進数字または 'a'(または'A')から'f'(または'F')までの文字で 0 から 15 の値を持つ文字の列である。

base が 2 から 36 の場合、変換対象文字列の形式は、
base で指定した基数を持つ数値を表現する文字または数字の並びであり、先頭に'+'または'-'の符号が付いてもよい。
'a'(または'A')から'z'(または'Z')は、10から35の値を表す。base 以下の値の文字のみ許される。

base が 16 の場合、0x または 0X の文字列が変換対象文字列の先頭に(符号がある場合は、その直後に)あってもよい。

変換対象文字列は、最初の非空白文字で始まる期待された形式の入力文字列の最長の並びとして定義される。
入力文字列が空か全て空白文字、または最初の非空白文字が符号でも数字でも数字を表現する文字でもない場合、変換対象文字列は空となる。

変換対象文字列が期待される形式で、base の値が 0 の場合、最初の数字で始まる文字の並びは、整数定数と解釈される。
変換対象文字列が期待される形式で、base の値が 2 から 36 の場合、base の値を変換の基数として用い、各文字は上記で説明した値を持つものとする。
変換対象文字列がマイナス符号で始まる場合、変換結果の値の符合を反転し符号無しの型にキャストして返す。

endptr が NULL でない場合、これらの関数は、最後の文字列へのポインタを endptr が指す領域に格納する。

変換対象文字列が空または期待する形式ではない場合は、変換は行われない。endptr が NULL でない場合、これらの関数は、str の値を endptr が指す領域に格納する。

【戻値】

strtol(), strtoll() は、成功した場合、変換した値を返す。

変換が行われなかった場合、0 を返す。

正しい値が表現できる範囲外の場合、LUONG_MAX または ULLONG_MAX を返す。

【エラー】

なし

【関連項目】

strtod, strtol, fscanf, isalpha

8.22 string.h

ヘッダ string.h は、以下の文字列を操作する関数を定義する。
T2EX では、標準Cライブラリの文字列操作関数群に対し、スレッドセーフではない関数 strerror, strtok は排除している。

関数

8.22.1 memccpy - メモリ領域の指定文字までのコピー

【書式】

```
#include <string.h>

void    *memccpy(void *dst, const void *src, int c, size_t n);
```

【解説】

memccpy() は、src が示すメモリ領域から dst が示すメモリ領域に、最大で n バイトのコピーを行う。
n バイトをコピーする前に、文字 c が現れると、そこでコピーは中止する。

オーバーラップしている領域間でコピーが行われた場合の動作は未定義とする。

【戻値】

memccpy() は、dst 内にコピーした c の次のバイトへのポインタを返す。
ただし、n バイトの間に c が見つからなかった場合、NULL を返す。

【エラー】

なし

【関連項目】

bcopy, memmove, memccpy

8.22.2 memchr - メモリ領域内の文字の検索

【書式】

```
#include <string.h>

void    *memchr(const void *s, int c, size_t n);
```

【解説】

memchr() は、s が指すメモリ領域の先頭から n バイトの中から(unsigned char に変換した)文字 c を探す。

【戻値】

memchr() は、見つかったバイトへのポインタを返す。
ただし、 n バイトの間に c が見つからなかった場合は、 NULL を返す。

【エラー】

なし

【関連項目】

strchr, index

8.22.3 memcmp - メモリ領域の比較

【書式】

```
#include <string.h>

int     memcmp(const void *s1, const void *s2, size_t n);
```

【解説】

memcmp() は、s1 が示す n バイトの領域と s2 が示す n バイトの領域を(各バイトを unsigned char とみなして)比較する。

【戻値】

s1 < s2 の場合は、負の整数を返す。
s1 = s2 の場合は、0 を返す。
s1 > s2 の場合は、正の整数を返す。

【エラー】

なし

【関連項目】

strcmp

8.22.4 memcpy - メモリ領域のコピー

【書式】

```
#include <string.h>

void    *memcpy(void *dst, const void *src, size_t n);
```

【解説】

memcpy() は、src が示すメモリ領域から dst が示すメモリ領域に、n バイトのコピーを行う。
各領域がオーバーラップしている場合の動作は未定義とする。

【戻値】

memcpy() は、dst を返す。

【エラー】

なし

【関連項目】

bcopy, memmove

8.22.5 memmove - オーバーラップするメモリ領域のコピー

【書式】

```
#include <string.h>

void    *memmove(void *dst, const void *src, size_t n);
```

【解説】

memmove() は、src が示すメモリ領域から dst が示すメモリ領域に、n バイトのコピーを行う。
コピーは、src が指す n バイトの領域を、src, dst が指す n バイトの領域と重ならない一時的な領域にコピーしてから dst が指す領域にコピーしたかのように行われる。
コピー元とコピー先のメモリ領域がオーバーラップしている場合でも、正常にコピーが行われる。

【戻値】

memmove() は、dst を返す。

【エラー】

なし

【関連項目】

bcopy, memcpy

8.22.6 memset - メモリ領域の設定

【書式】

```
#include <string.h>

void    *memset(void *s, int c, size_t n);
```

【解説】

memset() は、s が指す n バイトの領域の各バイトを、全て(unsigned char に変換した c)で埋める。

【戻値】

memset() は、s を返す。

【エラー】

なし

【関連項目】

bzero

8.22.7 strcat - 文字列の連結

【書式】

```
#include <string.h>

char    *strcat(char *dst, const char *src);
```

【解説】

strcat() は、dst が指す文字列の最後に、src が指す文字列をコピーして追加する。
src の先頭バイトは、dst の ヌル文字を上書きする。二つの文字列がオーバーラップしている場合の動作は未定義とする。

【戻値】

strcat() は、dst を返す。

【エラー】

なし

【関連項目】

strncat, strcpy, strncpy

8.22.8 strchr - 文字列中の文字検索

【書式】

```
#include <string.h>

char    *strchr(const char *s, int c);
```

【解説】

strchr() は、s が指す文字列中に最初に現れる(char 型に変換した) c を探す。終端の ヌル文字は、
文字列の一部とみなす。

【戻値】

strchr() は、文字が見つかった場合、その文字へのポインタを返す。
それ以外の場合 NULL を返す。

【エラー】

なし

【関連項目】

strrchr, memchr, index

8.22.9 strcmp - 文字列の比較

【書式】

```
#include <string.h>

int     strcmp(const char *s1, const char *s2);
```

【解説】

strcmp() は、s1 が指す文字列を s2 が指す文字列と比較する。

【戻値】

s1 < s2 の場合は、負の整数を返す。
s1 = s2 の場合は、0 を返す。
s1 > s2 の場合は、正の整数を返す。

【エラー】

なし

【関連項目】

bcmp, memcmp, strncmp

8.22.10 strcoll - システムロケールに基づく文字列の比較

【書式】

```
#include <string.h>

int     strcoll(const char *s1, const char *s2);
```

【解説】

strcoll() は、s1 が指す文字列を s2 が指す文字列と、システムロケールが規定する文字照合順序に基づいて比較する。

【戻値】

s1 < s2 の場合は、負の整数を返す。
s1 = s2 の場合は、0 を返す。
s1 > s2 の場合は、正の整数を返す。

【エラー】

なし

【関連項目】

bcmp, memcmp, strcmp

8.22.11 strcpy - 文字列のコピー

【書式】

```
#include <string.h>

char    *strcpy(char *dst, const char *src);
```

【解説】

strcpy() は、src が指す文字列を dst が指す領域にコピーする。
オーバーラップする領域間でコピーした場合、その動作は未定義とする。

【戻値】

strcpy() は、dst を返す。

【エラー】

なし

【関連項目】

bcopy, memcpy, memmove, strncpy

8.22.12 strcspn - 文字列中の指定文字を含まない先頭部分の長さの取得

【書式】

```
#include <string.h>

size_t  strcspn(const char *s1, const char *s2);
```

【解説】

strcspn() は、s1 が指す文字列の先頭から、s2 に含まれない文字だけで構成される、最長部分の長さ(バイト数)を計算する。

【戻値】

strcspn() は、s1 が指す文字列で、 s2 にない文字からなる部分の先頭からの長さを返す。

【エラー】

なし

【関連項目】

strspn, memchr, index, strchr, strstr

8.22.13 strdup - 文字列の複製

【書式】

```
#include <string.h>

char    *strdup(const char *s);
```

【解説】

strdup() は、s が指す文字列を複製した、新しい文字列へのポインタを返す。
返されたポインタは、free() に渡すことが出来る。
新しい文字列が生成できない場合、 NULL を返す。

【戻値】

strdup() は、成功した場合、新しい文字列へのポインタを返す。それ以外の場合、 NULL を返す。

【エラー】

なし

【関連項目】

malloc, calloc, realloc, free

8.22.14 strlen - 文字列の長さの取得

【書式】

```
#include <string.h>

size_t  strlen(const char *s);
```

【解説】

strlen() は、s が指す文字列の長さ(バイト数)を求める。ただし、終端のヌル文字は含まない。

【戻値】

strlen() は、文字列の長さを返す。

【エラー】

なし

8.22.15 strncat - 指定文字数分の文字列の連結

【書式】

```
#include <string.h>

char    *strncat(char *dst, const char *src, size_t n);
```

【解説】

strncat() は、dst が指す文字列の後に src が指す文字列を追加する。
ただし、src が指す文字列長が n バイトを超える場合は最大 n バイトまでを追加する。
src の先頭バイトは、dst の ヌル文字に上書きされる。dst の終端にはヌル文字を追加する。

【戻値】

strncat() は、dst を返す。

【エラー】

なし

【関連項目】

strcat, strcpy, strncpy

8.22.16 strncmp - 指定文字数分の文字列の比較

【書式】

```
#include <string.h>

int     strncmp(const char *s1, const char *s2, size_t n);
```

【解説】

strncmp() は、最大 n バイトまで、s1 が指す文字列を s2 が指す文字列と比較する(ヌル文字以降のバイトは比較しない)。

【戻値】

s1 < s2 の場合は、負の整数を返す。
s1 = s2 の場合は、0 を返す。
s1 > s2 の場合は、正の整数を返す。

【エラー】

なし

【関連項目】

strcmp, memcmp, bcmp

8.22.17 strncpy - 指定文字数分の文字列のコピー

【書式】

```
#include <string.h>
```

```
char    *strncpy(char *dst, const char *src, size_t n);
```

【解説】

strncpy() は、最大 n バイトまで src が指す文字列を dst が指す領域にコピーする(ヌル文字以後の文字はコピーしない)。

src が指す文字列が n バイトより短い場合、dst が指す領域は、n バイトに達するまでヌル文字が追加される。

オーバーラップするオブジェクト間でコピーされた場合の動作は未定義とする。

【戻値】

strncpy() は、dst を返す。

【エラー】

なし

【関連項目】

strcpy, bcopy, memcpy, memmove

8.22.18 strpbrk - 文字列中の文字位置検索

【書式】

```
#include <string.h>

char    *strpbrk(const char *s1, const char *s2);
```

【解説】

strpbrk() は、s1 が指す文字列の中で、s2 が指す文字列中のいずれかの文字が、最初に現れる位置を求める。

【戻値】

成功した場合、strpbrk() は、見つかった文字へのポインタを返す。見つからない場合、 NULL を返す。

【エラー】

なし

【関連項目】

strchr, strrchr, strspn

8.22.19 strrchr - 文字列の最後からの文字を検索

【書式】

```
#include <string.h>

char    *strrchr(const char *s, int c);
```

【解説】

strrchr() は、s が指す文字列の最後から先頭に向かい、(char に変換した) c が現れる位置を求める。
終端のヌル文字は、文字列の一部とみなす。

【戻値】

strrchr() は、見つかった文字へのポインタを返す。c が文字列中に存在しない場合、 NULL を返す。

【エラー】

なし

【関連項目】

strchr, index, rindex

8.22.20 strspn - 文字列中の指定文字を含む先頭部分の長さの取得

【書式】

```
#include <string.h>

size_t  strspn(const char *s1, const char *s2);
```

【解説】

strspn() は、s1 が指す文字列の先頭から、s2 が指す文字列の各文字だけで構成される最長部分の長さ(バイト数)を求める。

【戻値】

strspn() は、求めた長さを返す。

【エラー】

なし

【関連項目】

strcspn, strchr, strpbrk, strstr

8.22.21 strstr - 文字列中の文字列検索

【書式】

```
#include <string.h>

char    *strstr(const char *s1, const char *s2);
```

【解説】

strstr() は、s1 が指す文字列の先頭から、(終端のヌル文字を除いた)s2 が指す文字列が現れる位置を探す。

【戻値】

strstr() は、見つかった位置へのポインタを返す。見つからない場合、 NULL を返す。

【エラー】

なし

【関連項目】

strspn, strpbrk

8.22.22 strtok_r - 文字列からトークン分割

【書式】

```
#include <string.h>

char    *strtok_r(char *str, const char *sep, char **lasts);
```

【解説】

strtok_r() は、繰り返し呼び出すことにより、str が指す文字列を、sep が指す文字列の中のいずれかの文字で区切られるトークンに分割する。

strtok_r() の最初の呼び出しでは、str はヌルで終わる文字列で、二回目以降の呼び出しでは、
str は、 NULL とする。sep が指す区切り文字用の文字列は、各呼び出し毎に変えることができる。

最初の呼び出しでは、str が指す文字列から、sep が指す現在の区切り文字列に含まれていない最初の文字を探す。
そのような文字が見つからなかった場合、str が指す文字列には、トークンは存在せず、strtok_r() は、
NULL を返す。そのような文字が見つかった場合、それは最初のトークンの先頭文字となる。

strtok_r() は、次に、その位置から区切り文字列に含まれている文字を探す。文字が見つからない時、
現在のトークンは str が指す文字列の最後までとなり、次のトークンの検索では NULL を返す。
文字が見つかった場合、現在のトークンを最終位置の文字には、ヌル文字が上書きされ、現在の
トークンを終了する。strtok_r() は、次のトークンの検索のための情報を lasts が指す領域に格納する。

続く二回目以降の一連の呼び出しは、lasts に保存された情報に基づき、前回解析した位置の直後から検索を開始し、上記と同様に振舞う。

二回目以降の一連の呼び出しでは、str は NULL とし、lasts は以前の呼び出しから変更してはいけない。

【戻値】

strtok_r() は、見つかったトークンの先頭へのポインタ返す。トークンが見つからない場合、strtok_r() は、
NULL を返す。

【エラー】

なし

8.22.23 strxfrm - 文字列の変換

【書式】

```
#include <string.h>

size_t  strxfrm(char *dst, const char *src, size_t n);
```

【解説】

strxfrm() は、src が指す文字列を変換し、dst が指す領域に格納する。

文字列の変換はシステムロケール情報のLC_COLLATEカテゴリに相当する値に基づいて行う。
(T2EX ではロケールを自由に設定する機能はないので、この値はシステムロケールで決められた固定値である)

変換は、以下のように行う:
二つの変換した文字列を strcmp() で比較した結果と、元の文字列を strcoll() で比較したときの結果は同じになる。

dst には、終端のヌル文字も含め、 n バイトを超えて格納しない。
n が 0 の場合、dst は NULL でもよい。
オーバーラップしたオブジェクト間でコピーが行われる時、動作は未定義とする。

【戻値】

strxfrm() は、成功した場合、(終端のヌル文字を含めない)変換した文字列の長さを返す。
戻値が n 以上の場合、dst の内容は不定とする。

【エラー】

なし

【関連項目】

strncpy

8.22.24 strerror_r - エラー番号のエラーメッセージの取得

【書式】

```
#include <string.h>

int             strerror_r(int errnum, char *buf, size_t buflen);
```

【解説】

strerror_r() は、errnum で指定したエラー番号に対応した、システムロケールで表現されたエラーメッセージ文字列を、buf で指定した長さ buflen バイトの領域に取得する。

【戻値】

成功した場合、strerror_r() は、0 を返す。それ以外の場合正のエラー番号を返す。

【エラー】

EINVAL	不正な(あるいは未知の)エラー番号
ERANGE	buf の領域がメッセージを格納するのに不足

8.22.25 strercd_r - エラーコードのエラーメッセージの取得

【書式】

```
#include <string.h>

int             strercd_r(ER ercd, char *buf, size_t buflen);
```

【解説】

strercd_r() は、ercd で指定したエラーコードに対応した、システムロケールで表現されたエラーメッセージ文字列を buf で指定した、長さ buflen バイトの領域に格納する。

【戻値】

成功した場合、strercd_r() は、0 を返す。それ以外の場合、正のエラー番号を返す。

【エラー】

EINVAL	不正な(あるいは未知の)エラーコード
ERANGE	buf の領域がメッセージを格納するのに不足

【補足事項】

strercd_r() は、T2EX 独自関数である。

8.23 strings.h

ヘッダ strings.h は、以下の関数を定義する。

最新のPOSIX(IEEE Std 1003.1-2008)仕様からは削除されている関数であっても、まだ UNIX 系OSで一般的に使われているものについては、移植性の観点から削除せずに提供する。
/* LEGACY */ とコメントしている関数が、それらに相当する。

関数

8.23.1 bcmp - バイト列の比較

【書式】

```
#include <strings.h>

int     bcmp(const void *s1, const void *s2, size_t n); /* LEGACY */
```

【解説】

bcmp() は、s1 が指すバイト列と s2 が指すバイト列の先頭から n バイトを比較する。等しい場合 0 を返す。

【戻値】

bcmp() は、二つのバイト列の先頭 n バイトが等しい場合、または n が 0 の場合 0 を返す。
それ以外の場合、0 以外の値を返す。

【エラー】

なし

【関連項目】

memcmp, strcmp

【補足事項】

bcmp() は、POSIX(IEEE Std 1003.1-2008)仕様では削除されている。
代わりに memcmp() を使うのが望ましい。

8.23.2 bcopy - バイト列のコピー

【書式】

```
#include <strings.h>

void    bcopy(const void *dst, void *src, size_t n); /* LEGACY */
```

【解説】

bcopy() は、src が指す n バイトの領域を、dst が指す領域にコピーする。
二つの領域がオーバーラップしていても、正しくコピーする。

【戻値】

bcopy() は、値を返さない。

【エラー】

なし

【関連項目】

memmove, memcpy, strcpy

【補足事項】

bcopy() は、POSIX(IEEE Std 1003.1-2008) 仕様では削除されている。
代わりに、memmove() を使うのが望ましい。

8.23.3 bzero - メモリ領域の操作

【書式】

```
#include <strings.h>

void    bzero(void *s, size_t n); /* LEGACY */
```

【解説】

bzero() は、s が指す先頭から n バイトの領域を、値 0 のバイトで埋める。

【戻値】

bzero() は、値を返さない。

【エラー】

なし

【関連項目】

memset

【補足事項】

bzero() は、POSIX(IEEE Std 1003.1-2008) 仕様では削除されている。
代わりに、memset() を使うのが望ましい。

8.23.4 ffs - 最初にセットされているビットの検索

【書式】

```
#include <strings.h>

int     ffs(int i);
```

【解説】

ffs() は、i の中で、(最下位ビットから始めて)最初にセットされているビットを探し、そのビットの
位置を返す。最下位ビットの値は、1 である。

【戻値】

ffs() は、最初にセットされているビットの位置を返す。
i のどのビットにもセットされていない場合(i が 0)は、 0 を返す。

【エラー】

なし

8.23.5 index, rindex - 文字列中の文字位置の特定

【書式】

```
#include <strings.h>

char    *index(const char *s, int c); /* LEGACY */
char    *rindex(const char *s, int c); /* LEGACY */
```

【解説】

index() は、s が指す文字列中に、最初に(char に変換した) c が現れる位置へのポインタを返す。
rindex() は、s が指す文字列中に、最後に(char に変換した) c が現れる位置へのポインタを返す。

【戻値】

index(), rindex() は、見つかった文字へのポインタを返す。
見つからなかった場合、 NULL を返す。

【エラー】

なし

【関連項目】

strchr, strrchr

【補足事項】

index(), rindex() は、POSIX(IEEE Std 1003.1-2008) 仕様では削除されている。
代わりに、strchr(), strrchr() を使うのが望ましい。

8.23.6 strcasecmp, strncasecmp - 大文字小文字を区別しない文字列の比較

【書式】

```
#include <strings.h>

int     strcasecmp(const char *s1, const char *s2);
int     strncasecmp(const char *s1, const char *s2, size_t n);
```

【解説】

strcasecmp() は、s1 が指す文字列と s2 が指す文字列を、大文字小文字を区別せずに比較する。
strncasecmp() は、s1 が指す文字列と s2 が指す文字列を、最大 n バイトまで、大文字小文字を区別せずに比較する。
比較は、システムロケールに基づいた文字照合順序で、大文字小文字の違いを無視して行う。

【戻値】

s1 < s2 の場合は、負の整数を返す。
s1 = s2 の場合は、0 を返す。
s1 > s2 の場合は、正の整数を返す。

【エラー】

なし

【関連項目】

strcmp, strncmp

8.24 time.h

ヘッダ time.h は、以下の時刻に関する型や関数を定義する。

以下の関数で参照しているタイムゾーンは、常にシステムタイムゾーンである。
システムタイムゾーンは、自動的に設定されることはない。
予めカレンダ機能の dt_setsystz() を実行してシステムタイムゾーンを設定しないと、
システムタイムゾーンの初期値は実装定義の値となる。

以下の型を定義する。

struct tm 構造体
　　カレンダ時刻を要素別に表す構造体を示す。
　　カレンダ機能の、struct tm 構造体参照。

time_t
　　秒単位の時刻を表す整数型。
　　カレンダ時刻(1970年1月1日0時(UTC)からの通算秒数)を表すのに用いる。

関数

8.24.1 asctime_r - 時刻から文字列への変換

【書式】

```
#include <time.h>

char    *asctime_r(const struct tm *tm, char *buf);
```

【解説】

asctime_r() は、tm が指す構造体の要素別の時刻を文字列形式に変換する。
文字列形式は、「曜日 月 日 時:分:秒 西暦」の形式で、以下のようなものである。

```
        "Thu Dec 16 09:24:58 2011¥n"
```

文字列の終りには、'¥n'が追加される。

曜日には、"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" を用いる。
月には、"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" を用いる。

変換した文字列は、buf が指す領域(26 バイト以上必要)に格納し buf を返す。
tm 内の要素別時刻の曜日(tm_wday)または月(tm_mon)が範囲外の値の場合、または tm_year - 1900 が INT_MAX を
超える場合、または終端のヌル文字も含む結果の長さが26を超える場合の動作は未定義とする。

【戻値】

asctime_r() は、成功した場合 buf を返す。変換できなかった場合 NULL を返す。

【エラー】

なし

【関連項目】

ctime_r, gmtime_r, localtime_r

8.24.2 ctime_r - カレンダ時刻から文字列への変換

【書式】

```
#include <time.h>

char    *ctime_r(const time_t *clock, char *buf);
```

【解説】

ctime_r() は、clock が指すカレンダ時刻(1970年1月1日0時(UTC)からの通算秒数)をシステムタイムゾーンでのローカル時刻として文字列形式に変換し buf が指す領域(26 バイト以上必要)に格納する。

文字列形式については asctime_r() を参照。

【戻値】

ctime_r() は、成功した場合 buf を返す。変換できなかった場合 NULL を返す。

【エラー】

なし

【関連項目】

asctime_r, gmtime_r, localtime_r

8.24.3 gmtime_r_eno, gmtime_r - カレンダ時刻から要素別UTC時刻への変換

【書式】

```
#include <time.h>

struct tm *gmtime_r_eno(const time_t *clock, struct tm *result, errno_t *enop);
struct tm *gmtime_r(const time_t *clock, struct tm *result);
```

【解説】

gmtime_r_eno(), gmtime_r() は、clock が指すカレンダ時刻(1970年1月1日0時(UTC)からの通算秒数)を、協定世界時(UTC)での要素別時刻に変換し
 result が指す struct tm 型データに格納する。
変換できない場合、enop 指す領域にエラー番号を格納する。
enop を NULL とした場合、エラー番号は格納されない。

gmtime_r() は、gmtime_r_eno(clock, tm, NULL) と等価である。

【戻値】

gmtime_r_eno(), gmtime_r() は、成功した場合 result を返す。変換できなかった場合、エラー番号を enop が指す領域に格納し NULL を返す。

【エラー】

enop が示す領域に戻されるエラー番号:
EOVERFLOW　　結果が表現できる範囲にない

【関連項目】

asctime_r, ctime_r, localtime_r

8.24.4 localtime_r_eno, localtime_r - カレンダ時刻から要素別ローカル時間への変換

【書式】

```
#include <time.h>

struct tm *localtime_r_eno(const time_t *clock, struct tm *result, errno_t *enop);
struct tm *localtime_r(const time_t *clock, struct tm *result);
```

【解説】

localtime_r_eno() は、clock が指すカレンダ時刻(1970年1月1日0時(UTC)からの通算秒数)をシステムタイムゾーンでの
要素別のローカル時刻に変換し、result が指す struct tm 型データに格納し result を返す。
変換できない場合、enop 指す領域にエラー番号を格納する。
enop を NULL とした場合、エラー番号は格納されない。

localtime_r() は、localtime_r_eno(clock, result, NULL) と等価である。

【戻値】

localtime_r_eno(), localtime_r() は、成功した場合 result を返す。変換できなかった場合 NULL を返す。

【エラー】

enop が示す領域に戻されるエラー番号:
EOVERFLOW　　結果が表現できる範囲にない

【関連項目】

asctime_r, ctime_r, gmtime_r

8.24.5 mktime_eno, mktime - 要素別ローカル時刻からカレンダ時刻への変換

【書式】

```
#include <time.h>

time_t  mktime_eno(struct tm *, errno_t* enop);
time_t  mktime(struct tm *tm);
```

【解説】

mktime_eno() は、tm が指すローカル時刻で表現された要素別の時刻をカレンダ時刻(1970年1月1日0時(UTC)からの通算秒数)に変換する。
tm 内の tm_wday と tm_yday の初期値は、無視する。他の要素の初期値も正しい範囲に入っている必要はない。
変換に成功すると、tm_wday と tm_yday は適切な値に設定し、他の要素は正しい範囲内になるように設定される。
tm_mon と tm_year が決定されるまで tm_mday は設定されない。

変換できない場合、enop 指す領域にエラー番号を格納する。
enop を NULL とした場合、エラー番号は格納されない。

mktime() は mktime_eno(tm, NULL) と等価である。

【戻値】

mktime_eno(), mktime() は、カレンダ時刻(1970年1月1日0時(UTC)からの通算秒数)を返す。
カレンダ時刻(1970年1月1日0時(UTC)からの通算秒数)が表現できない場合 ((time_t)(-1)) を返す。

【エラー】

enop が示す領域に戻されるエラー番号:
EOVERFLOW　　結果が表現できる範囲にない

【関連項目】

ctime_r, gmtime_r, localtime_r

8.24.6 time - 現在時刻の取得

【書式】

```
#include <time.h>

time_t  time(time_t *tloc);
```

【解説】

time() は、現在時刻を、1970年1月1日0時(UTC)からの通算秒数で返す。
引数 tloc が NULL でない場合、tloc が指す領域にも戻値を格納する。

【戻値】

time() は、成功した場合現在時刻を返す。失敗した場合 ((time_t)(-1)) を返す。

【エラー】

なし

8.24.7 difftime - カレンダ時刻の時間差の計算

【書式】

```
#include <time.h>

double  difftime(time_t time1, time_t time0);
```

【解説】

difftime() は、二つのカレンダ時刻の差(time1 - time0)を求める。

【戻値】

difftime() は、秒数の差分を double 型で返す。

【エラー】

なし

8.24.8 strftime - 日付および時刻から文字列への変換

【書式】

```
#include <time.h>

size_t  strftime(char *s, size_t max, const char *format, const struct tm *tm);
```

【解説】

strftime() は、tm が指す時刻を format で指定した書式にしたがって文字列に変換し、
ヌル文字も含めて最大 max 文字まで変換し文字列 s に書き込む。

format は変換書式を表す文字列で、0 個以上の変換指定文字列(変換指示子)と通常の文字を含む。
format の詳細は、6章 カレンダ機能の dt_strftime() の format と同じである。

【戻値】

strftime() は、成功した場合、s に格納したヌル文字を含まない結果のバイト数を返す。

それ以外の場合、0 を返し s が指す領域は不定となる。

【エラー】

なし

【関連項目】

dt_strftime

8.24.9 strptime - 文字列から日付および時刻への変換

【書式】

```
#include <time.h>

char    *strptime(const char *str, const char *format, struct tm *tm);
```

【解説】

strptime() は、str で示す文字列を format で指定した書式にしたがってtm構造体の時刻に変換し tm に格納する。
format 詳細は、6章 カレンダ機能の dt_strptime() の format と同じである。

【戻値】

strptime() は、成功した場合、最後に解析した文字の次の文字へのポインタを返す。それ以外の場合 NULL を返す。

【エラー】

なし

【関連項目】

dt_strptime

8.25 wchar.h

ヘッダ wchar.h は以下のワイド文字に関する定義を提供する。

以下の型のみを定義する。T2EX ではワイド文字に関連したライブラリ関数を提供しないため、ワイド文字型のみを規定する。
その他の標準Cライブラリで規定されている多くの定義は、T2EX では規定しない。

wchar_t
: システムロケールの拡張文字セットの全てのメンバを区別できるコードを表現する値の範囲を持つ整数型を示す。
stddef.h でも定義されている。

Appendix

A.1 システム構成情報

T2EX では、システムに関する設定情報を保持・管理するために、T-Kernel 2.0 のシステム構成情報管理機能を利用している。本節では、T2EXによって標準定義されるシステム管理情報を示す。

システム構成情報管理機能そのものについては、T-Kernel 2.0仕様書の 5.7 節を参照すること。

○ T-Kernel 2.0

T-Kernel 2.0 の仕様では、TSVCLimit によってシステムコールの呼び出し可能なレベル・Kmalloc/Vmallocによって割り当てられるメモリの保護レベルの両方が設定される。T2EXでは、前者のみを TSVCLimit で設定されるものとし、後者については新たなパラメータ TKMallocLvl によって独立して設定されるように変更する。

N	TSVCLimit	T-Kernel 2.0 および T2EX システムコールの呼び出し可能な最も低い保護レベル T2EX ではこの値を 2 とし、値を変更した場合の動作は保証しない。
N	TKmallocLvl	Kmalloc/Vmalloc によって割り当てられるメモリの保護レベル T2EX ではこの値を 1 とし、値を変更した場合の動作は保証しない。 本パラメータが設定されていない場合は、TSVCLimit の値が用いられる。

他の T-Kernel 2.0 に由来する標準システム構成情報については、T-Kernel 2.0 仕様書の 5.7 節を参照すること。

○ ファイル管理機能

ファイル管理機能全体のシステム構成情報

N	FsMaxFile	同時にオープン可能なファイル数(システム全体)
N	FsMaxFIMP	同時に登録可能なファイルシステム実装部数
N	FsMaxCON	同時に接続可能な接続数
N	FsAccessTime	最終アクセス時刻の更新の可否

FATファイルシステム実装部のシステム構成情報

N	FiFAT_TskPri	タスク優先度
N	FiFAT_StkSz	タスクスタックサイズ
N	FiFAT_FCacheSz	FATキャッシュサイズ(バイト数)
N	FiFAT_FCacheNs	FATキャッシュ単位セクタ数
N	FiFAT_RCacheSz	ルートディレクトリキャッシュサイズ(バイト数)
N	FiFAT_RCacheNs	ルートディレクトリキャッシュ単位セクタ数
N	FiFAT_DCacheSz	データキャッシュサイズ(バイト数)
N	FiFAT_DCacheNs	データキャッシュ単位セクタ数

○ ネットワーク通信機能

N	SoMaxSocket	同時にオープン可能な最大ソケット数
N	SoTaskBasePri	ネットワーク通信機能で生成されるタスクの最大優先度 SoTaskBasePri から (SoTaskBasePri+4) の範囲の優先度をもつ複数のタスクが生成される。
N	SoDrvRxBufNum	LANドライバに登録する受信バッファ数
N	SoTcpTxBufSz	デフォルトのTCP/IPの送信キューのバッファサイズ(バイト数)
N	SoTcpRxBufSz	デフォルトのTCP/IPの受信キューのバッファサイズ(バイト数)
N	SoUdpTxBufSz	デフォルトのUDP/IPの送信キューのバッファサイズ(バイト数)
N	SoUdpRxBufSz	デフォルトのUDP/IPの受信キューのバッファサイズ(バイト数)
N	SoRawIPTxBufSz	デフォルトのSOCK_RAW型ソケット(AF_INET)の送信キューのバッファサイズ(バイト数)
N	SoRawIPRxBufSz	デフォルトのSOCK_RAW型ソケット(AF_INET)の受信キューのバッファサイズ(バイト数)
N	SoRawTxBufSz	デフォルトのSOCK_RAW型ソケット(AF_ROUTE)の送信キューのバッファサイズ(バイト数)
N	SoRawRxBufSz	デフォルトのSOCK_RAW型ソケット(AF_ROUTE)の受信キューのバッファサイズ(バイト数)
N	SoTcpDoAutoTx	TCP/IPの送信バッファサイズの自動リサイズの許可 0以外の値の場合に自動リサイズを許可する。
N	SoTcpIncAutoTx	TCP/IPの送信バッファサイズの自動リサイズの増加サイズ(バイト数)
N	SoTcpMaxAutoTx	TCP/IPの送信バッファサイズの自動リサイズ後の最大サイズ(バイト数)
N	SoTcpDoAutoRx	TCP/IPの受信バッファサイズの自動リサイズの許可 0以外の値の場合に自動リサイズを許可する。
N	SoTcpIncAutoRx	TCP/IPの受信バッファサイズの自動リサイズの増加サイズ(バイト数)

N	SoTcpMaxAutoRx	TCP/IPの受信バッファサイズの自動リサイズ後の最大サイズ(バイト数)

○ プログラムロード機能

N	PmMaxProg	同時にロード可能な最大プログラム数

A.2 ブレークAPIコール(fs_break, so_break)の利用例

本節では、ファイル管理機能・ネットワーク通信機能におけるブレークAPIコール(fs_break, so_break)の利用方法の例を示す。

ブレークAPIコールは、ファイルやネットワークへの入出力処理による待ちを解除し、入出力処理を安全に中止させるための機能である。ユーザ操作による処理のキャンセル、電源残量の低下やシャットダウンなどのシステム要因による処理のキャンセルの際に、タスクが実行している入出力処理を安全に終了させるために用いられる。T-Kernel 2.0 API の tk_rel_wai に相当する機能であるが、それぞれファイル管理機能・ネットワーク管理機能による待ちに限って解除を行う。

ブレークAPIコールの利用例として、cnt バイトの読込みが行われるまでソケットディスクリプタ sd からの読み込みを行う処理に対する中止処理を示す。A.2.1 節に読込み処理本体の実装例を示し、その処理に対する中止処理の実装例を A.2.2 節に示す。

A.2.1 読込み処理の実装例

この例では、ソケットからの読込み処理が優先度 SOCKET_READ_TASK_PRI を持つタスクで実行されると想定する。以下に示すスタートアップ関数の実装例では、外部からの中止をサポートするため、中止の有無を示す変数 interrupted および中止処理の完了を示す変数 finished を定義し用いている。

```
LOCAL volatile BOOL finished = FALSE;
LOCAL volatile BOOL interrupted = FALSE;

void socketReadTask( INT stacd, void* exinf )
{
    int sd = (int)stacd;
    int pos;
    ER er;

    for (pos = 0; pos < cnt && !interrupted; ) {
        /* so_read により、待ちに入る可能性がある */
        er = so_read(sd, buf + pos, cnt - pos);
        if (er <= 0)
            break;
        pos += er;
    }

    /* 中止処理が正しく実行されたことを、中止処理側に通知する */
    finished = TRUE;

    /* タスクを終了する */
    tk_ext_tsk();
}
```

A.2.2 中止処理の実装例

A.2.1 節の処理に対し、ブレークAPIコールを用いた中止処理は以下のように実装することができる。

```
void interruptSocketReadTask(ID tskid)
{
    ER er;

    /* 中止処理を実行するタスクの優先度を下げる
       (中止対象のタスクが待ちに入っているときだけ so_break を実行する) */
    tk_chg_pri(TSK_SELF, SOCKET_READ_TASK_PRI + 1);

    /* タスクの処理を中止させる */
    while (!finished) {
        interrupted = TRUE;
        so_break(tskid);
    }
}
```

基本的な考え方としては、so_break を用いて so_read の待ちを解除させて処理を中止すればよい。
ただし、so_break による待ち解除要求はキューイングされないため、so_read を実行する前後でブレークAPIコールが呼び出された場合には効果を及ぼさないことに注意が必要である。

本節での実装例では、変数 finished が真となり、処理が正しく中止されたことが確認できるまで、so_break を繰り返し実行する。これによって、確実に対象となる処理が中止できることが保証される。

A.3 一般プログラムモジュール機能の利用例

本節では、一般プログラムモジュール機能の利用例として、データ構造としてのスタックを一般プログラムモジュールとして実装する例を示す。

A.3.1 一般プログラムモジュールの定義例

int 型の要素に対するスタックの push, pop 操作を実現する一般プログラムモジュールの定義例を示す。簡単のためエラー処理等は省くが、実際のプログラムモジュールではエラーの扱いも正しく行うことが推奨される。

○ stack.h

```
/*
 * スタックモジュール外部インタフェース定義 (stack.h)
 */
#ifndef DEFINE_H_STACK
#define DEFINE_H_STACK

typedef struct stack_module {
    /* モジュールの初期化の際のパラメータ */
    int size;

    /* モジュールのインタフェース関数 */
    void (*push)(int data);
    void (*pop)(int* pdata);
} STACK_MODULE;

#endif
```

○ stack.c

```
/*
 * スタックモジュールの実装 (stack.c)
 */
#include <tk/tkernel.h>
#include "stack.h"

LOCAL FastLock lock;
LOCAL int*     stack;
LOCAL int      sp;

LOCAL void stack_push(int data)
{
    Lock(&lock);
    stack[sp++] = data;
    Unlock(&lock);
}

LOCAL void stack_pop(int* pdata)
{
    Lock(&lock);
    *pdata = stack[--sp];
    Unlock(&lock);
}

EXPORT int module_main(BOOL startup, void* arg)
{
    STACK_MODULE* sm = (STACK_MODULE*)arg;

    if (startup) {
        /* 初期化処理 */
        CreateLock(&lock, "stkl");
        stack = (int*)malloc(sm->size * sizeof(int));
```

```
        sp = 0;
        sm->push = stack_push;
        sm->pop = stack_pop;
    }
    else {
        /* 終了処理 */
        free(stack);
        DeleteLock(&lock);
    }

    return 0;
}
```

A.3.2 一般プログラムモジュールの利用例

A.3.1 節で定義されたスタックモジュールをアプリケーションプログラムから利用する際のコード例を示す。

○ sample.c

```
/*
 * スタックの利用方法の説明 (sample.c)
 */
#include <basic.h>
#include <t2ex/load.h>
#include "stack.h"

LOCAL ID smId = E_NOEXS;
LOCAL STACK_MODULE smif;
LOCAL pm_entry_t* sm_main = NULL;

EXPORT void sample()
{
    ER er;
    int data1, data2;
    struct pm target = {
        .pmatr = PM_FILE,
        .pmhdr = "/sysdsk/module/stack.obj",
    };

    /* モジュールのロード*/
    er = pm_load(&target, &sm_main);
    if (er < E_OK) {
        fprintf(stderr, "pm_load error: %d¥n", er);
        return;
    }
    smId = er;

    /* モジュールの初期化 */
    smif.size = 100;
    sm_main(TRUE, &smif);

    /* モジュールを利用 */
    smif.push(1);
    smif.push(2);
    smif.pop(&data1);
    smif.push(3);
    smif.pop(&data2);

    /* モジュールの終了処理 */
    sm_main(FALSE, &smif);

    pm_unload(smId);
    sm_main = NULL;
    smId = E_NOEXS;
}
```