T-Engine フォーラム
ミドルウェア流通 WG

ミドルウェア開発者のための T-Engine 入門

Version 1.01

2005 年 9 月

はじめに

本書は、T-Engine ミドルウェアの入門書の位置づけとして、T-Engine フォーラム ミドルウェア流通 WG においてまとめられたものである。

T-Engine を利用したミドルウェア開発に本書が活用され、T-Engine が世の中に深く浸透することを期待するものである。

編纂に当たり、ミドルウェア流通 WG メンバに多大なご協力を頂いたことに感謝したい。

T-Engine フォーラム

改訂履歴

版数	改版日	内容	備考
0.90	2005/3/22	新規作成	
1.00	2005/7/21	誤字脱字の修正等	
1.01	2005/9/13	細部修正･公開	

注意事項

・個人の私的使用に限り、講習テキストを自由に利用することができます。
・原則として、講習テキストの再配布を行うことはできません。
・講習テキストの著作権は T-Engine フォーラムに帰属します。
・講習テキストに記されている内容は、将来予告無く変更されることがあります。
・T-Engine フォーラムおよび講習テキスト執筆協力者は、講習テキストに記載されている内容について、明示的にも黙示的にも、また法律で定められているか否か問わず、一切の保証をいたしません。
・T-Engine フォーラムおよび講習テキスト執筆協力者は、講習テキストに関連して発生したいかなる損害もしくは不利益についても一切責任を負いません。

・TRON は、"The Real-time Operating system Nucleus"の略称です。
・ITRON は、"Industrial TRON"の略称です。
・μITRON は、"Micro Industrial TRON"の略称です。
・TRON、BTRON、ITRON、μITRON、eTRON、T-Engine、μT-Engine、T-Monitor、T-Kernel、T-JV、T-Dist はコンピュータの仕様に対する名称であり、特定の商品を指すものではありません。
・Linux は、Linus Torvalds の米国およびその他の国における商標または登録商標です。
・UNIX は、TheOpenGroup の米国およびその他の国における商標または登録商標です。
・Red Hat は、Red Hat,Inc.の米国およびその他の国における商標または登録商標です。
・Microsoft、Windows は、Microsoft Corporation の米国およびその他の国における商標または登録商標です。
・その他の本書に記載されている会社名、商品名などは、各社の商標または登録商標です。
本書では、『™』、『®』マークを明記しておりません。

本書の構成

T-Kernelの概要

本章の概要および前提知識

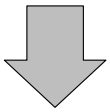
- 本章の概要
  - T-Kernelの概要について紹介する
- 本章を読むために必要な前提知識
  - 組み込みシステムの基本的な知識

T-Kernelとは

- T-Engineプロジェクトから生まれた新世代のリアルタイムOS
- ソースコードが公開されたオープンなOS
- 比較的小規模な組み込み機器から、より大規模で高度なシステムまで対応可能なリアルタイムOS

T-Kernel仕様策定の経緯

- 従来の組み込みOS（μITRON）
  - 標準的なリアルタイムOSとして利用されている
    - 携帯電話、デジカメなどの最新の情報機器、各種家電製品、制御機器などに広く使われている
    - 80年代の設計のため、近年の高度化した組み込みシステムの要求に対応しきれていない



- T-Kernel
  - μITRONの優れた面を継承しつつ、新たな要求に対応
  - 古い仕様に縛られること無く新しいOSとして設計

2005.09.13 version 1.01 4

μITRONリアルタイムカーネル仕様（μITRON仕様書）は、トロンプロジェクトのホームページ
- http://www.tron.org/

からダウンロード可能です。

現在のμITRONの最新バージョンは、『μITRON4.0仕様』です。

T-Kernelの特徴

1. ミドルウェア流通のための強い標準化
2. プログラムのダイナミックロード
3. MMUを用いたメモリ管理
4. より高度なシステムへの拡張性

ミドルウェア流通のための強い標準化

- 従来の組み込みOS（μITRON）
  - 『弱い標準化』
    - ソフトの互換性に問題
    - 結果的にミドルウェアの流通を阻害
- T-Kernel
  - 『強い標準化』
    - ソフトウェアの互換性・ソフトウェアの流通を目的とする
    - 異なるT-Engineでも再コンパイルするだけで移植可能
  - 実装依存を極力排除
  - ソースコードを一本化

・“弱い標準化”とは

- 標準的な仕様は定めてオープンにするが、その実装や利用方法についてはユーザ側に多くが任される

プログラムのダイナミックロード

- 従来の組み込みOS（μITRON）
  - スタティックロード
    - OSもアプリケーションプログラムも、ミドルウェアも全てリンクし、一つのオブジェクトコードにする
    - プログラムコードのメモリ上の配置はスタティックに決定
      - 小規模な組み込みシステムではこれで充分
      - ただしソフトウェア部品を独立したオブジェクトとして扱うのは難しい
- T-Kernel
  - ダイナミックロード
    - メモリなどの資源のダイナミックな管理
    - タスクや通信・同期オブジェクトなどのID番号の自動割り当て

MMUを用いたメモリ管理

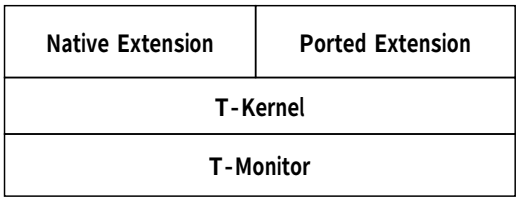
- 従来の組み込みOS（μITRON）
  - 物理メモリモデルが一般的
  - 近年、組み込みシステムでもMMUを備えた高機能なCPUが使用されるようになった
- T-Kernel
  - MMUを用いることで、モジュール間のメモリ保護が可能
  - プログラムコードのダイナミックなロードが容易
  - 仮想メモリやプロセス対応は、Extensionでサポート

より高度なシステムへの拡張性

- 従来の組み込みOS（μITRON）
  - 一部のミドルウェアはオプション的な扱い
    - ファイルシステムやネットワーク機能等
  - システムの高度化と共にOSとして標準化が望まれている
- T-Kernel
  - T-Kernel ExtensionというOSの機能を拡張する仕組みを設けた
    - 全ての機能をT-Kernelに取り込むことは、OSの巨大化を招くため

T-Kernel Extension

- T-Kernelの機能を拡張し、より高度なOSの機能を実現するためのプログラム
- 各種のExtensionが用意されている
  1. Native Extension
     - T-Engineフォーラムにより定められる標準のExtension
  2. Ported Extension
     - 各ベンダが開発したExtension
       - T-Linux, T-JVなど



Native Extension

- SE: Standard Extension
  - MMUベースの、組込みシステムに必要な標準ミドルウェアプロファイル
  - ファイルシステム
  - 仮想メモリまで対応したメモリ管理機能
  - プロセス管理
- TE: Tiny Extension
  - 実メモリベースの、コンパクトなハードリアルタイムシステム向けの標準ミドルウェアプロファイル
- EE: Enterprise Extension
  - セキュリティ機能を強化した、サーバシステム用T-Kernelの標準ミドルウェアプロファイル

Ported Extension

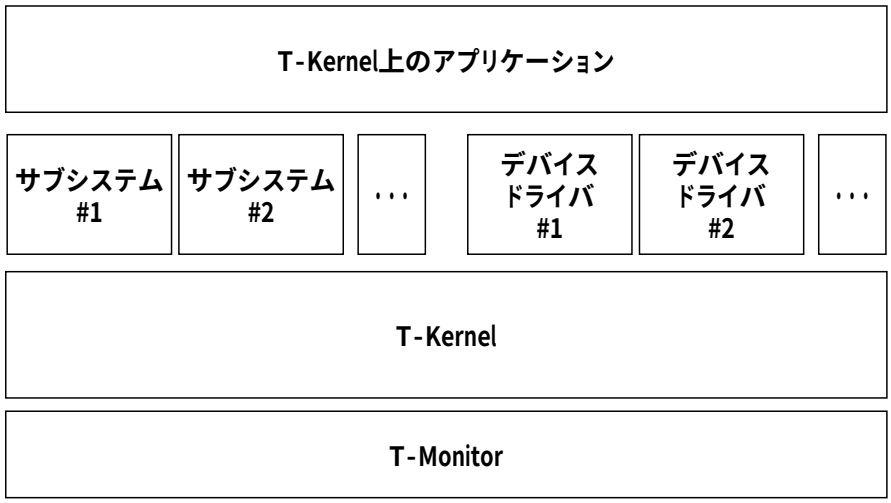
- T-Wireless
  - T-Kernel上の第三世代携帯電話のためのミドルウェア群
- T-JV
  - T-Kernel上で動作するJava実行環境
- T-Linux
  - T-Kernel上で動作するLinuxカーネル
- T-Integrator
  - T-Kernel上で動作するNexWave社の情報家電用ミドルウェア
- Windows CE .NET
  - T-Kernel上で動作するWindows CE .NET環境

T-Kernelのソフトウェア構成

本章の概要および前提知識

- 本章の概要
  - T-Kernelのソフトウェア構成について紹介する
- 本章を読むために必要な前提知識
  - 組み込みシステムの基本的な知識
  - 前章で述べたT-Kernelの概要に関する知識

T-Kernelのソフトウェア構成



T-Kernelのソフトウェア構成図（上位システムを含まないシンプルな例）

T-Monitor

- T-Engineのハードウェアに対する基本的な操作を実現するプログラム
  - ハードウェアの初期化、およびシステムの起動
  - ユーザに基本的なデバッグ機能を提供
    - Dump　　メモリ内容の表示
    - OutputByte　　I/Oポートへの出力　　など
  - アプリケーションにサービス関数を提供
    - tm_getchar()　　コンソールから1文字入力
    - tm_putchar()　　コンソールへ1文字出力
    - tm_getline()　　コンソールから1行入力
    - tm_putstring()　　コンソールへ文字列出力　など

デバイスドライバ

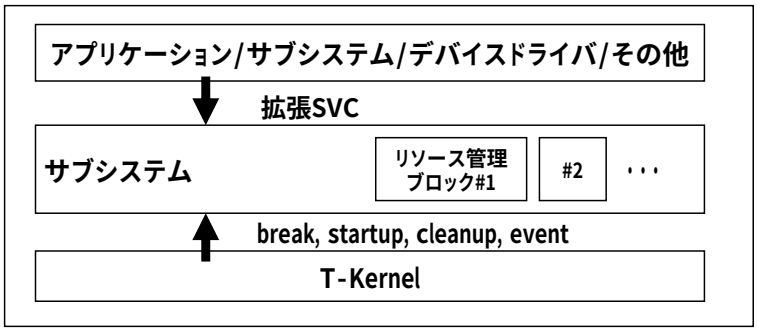
- ハードウェアの制御を行うソフト
- T-Kernel/SMがデバイスドライバを管理、アプリケーションにシステムコールを提供

T-Kernel

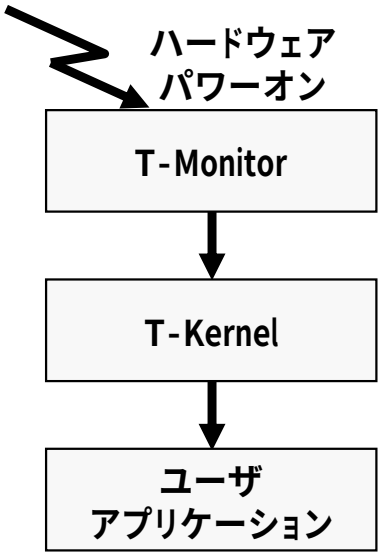
- T-Kernel/OS(Operating System)
  - リアルタイムOSとして基本的な機能
  - ITRONの機能にほぼ相当（狭義のT-Kernel）
- T-Kernel/SM(System Manager)
  - より上位のシステム管理機能
- T-Kernel/DS(Debugger Support)
  - 開発環境（デバッガ）のための機能
  - 一般ユーザは使用してはいけない

サブシステム

- システムに機能を追加するプログラム
  - 下記から構成される
    - 拡張ＳＶＣハンドラ
    - OSからの要求を処理する関数群
    - リソース管理ブロック



T-Kernelの起動



ハードウェアの初期化
T-Kernelを起動

初期化タスクを起動し、
サブシステム・
アプリケーションを起動

必要なタスクを生成・
起動し、処理スタート

T-Kernelの基本概念と動作

本章の概要および前提知識

- 本章の概要
  - T-Kernelの基本概念および動作について紹介する
- 本章を読むために必要な前提知識
  - 組み込みシステムの基本的な知識
  - T-Kernelの基本的な知識
  - T-Kernelのソフトウェア構成に関する知識

タスク･ディスパッチ･スケジューリング

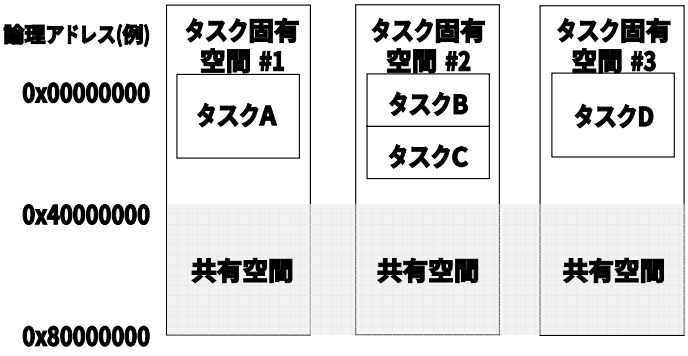
- タスク
  - プログラムの並列実行単位
    - 並列というのは、アプリケーションから見た概念
    - 実装上はカーネル制御のもと、それぞれのタスクが時分割に実行される
- ディスパッチ
  - プロセッサが実行するタスクを切り替えること
- スケジューリング
  - 次に実行すべきタスクを決定する処理
  - 一般的には、システムコール処理の中で行う

API・システムコール・オブジェクト

- API
  - T-Kernelが提供する機能の呼び込みI/Fの総称をAPI(Application Program Interface)と呼ぶ
- システムコール
  - OS機能を直接呼び出すAPIをシステムコールと呼ぶ
- オブジェクト
  - カーネルが操作対象とする資源をオブジェクトと呼ぶ
  - オブジェクトの例
    - タスク
    - メモリプール・セマフォ・メールボックスなどの同期通信機能
    - タイムイベントハンドラ

アドレス空間

- アドレス空間
  - メモリのアドレス空間は、共有空間とタスク固有空間に区別される
  - 共有空間は、全てのタスクからアクセス可能
  - タスク固有空間は、タスクが属しているタスク固有空間のみアクセスが可能。ただし、1つのタスク固有空間に複数のタスクが属している場合がある



保護レベル

- 保護レベル
  - T-Kernelは、0～3の4つのメモリ保護レベルをもつ
  - レベル0が最も高い特権を持ち、3が最も低い
  - 実行中の保護レベル以下のメモリにのみアクセス可能
    - 保護レベル2で実行している場合は、保護レベル2および3のメモリにアクセス可能
  - MMUの無いシステムは、すべて保護レベル0として扱う

非常駐メモリ

- 非常駐メモリ
  - 非常駐メモリは、そのメモリにアクセスすることにより、ディスクなどからメモリへデータ転送を行う
    - そのため、デバイスなどが動作できる状況でなければならない
  - 非常駐メモリのアクセスによるディスクなどからの転送は、T-Kernelでは行わない
    - 通常は、仮想記憶管理などを行うサブシステムがT-Kernelに組み合わされて行われる

メモリには、常駐メモリと非常駐メモリがあります。

タスク状態(1)

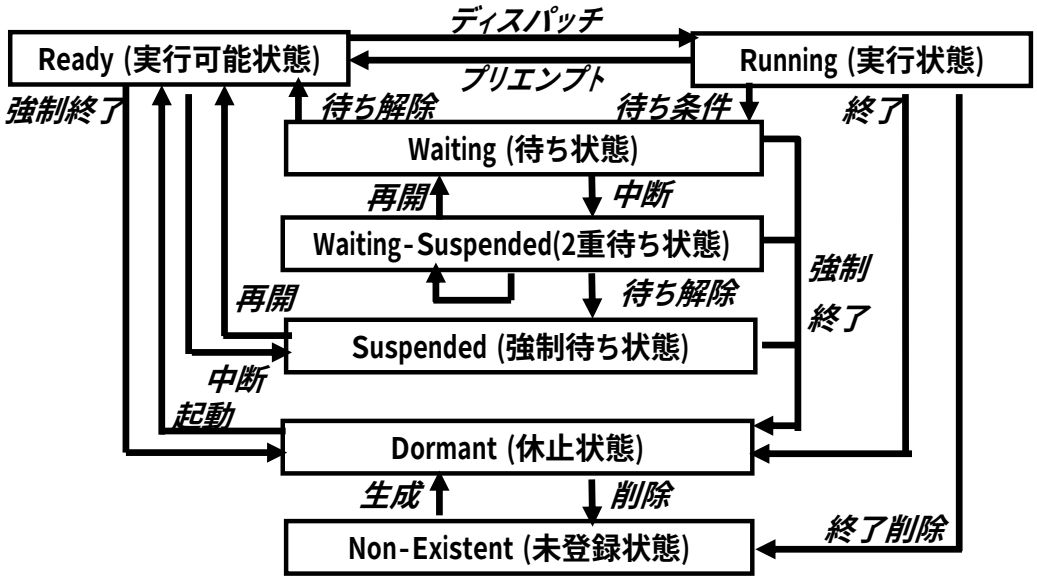
- 未登録状態 (Non-Existent)
  - タスクがまだ生成されていないか、削除された状態
- 休止状態 (Dormant)
  - タスクがまだ起動していないか、実行を終了した後の状態
- 実行可能状態 (Ready)
  - そのタスクを実行する準備は整っているが、優先度の高い他タスクが実行中のため、実行できない状態
- 実行状態 (Running)
  - 現在そのタスクが実行中である状態

タスク状態(2)

- 待ち状態(Waiting)
  - システムコールを呼び出したことにより、何らかの条件が整うまで実行が中断された状態。言いかえると、条件が満たされるのを待っている状態
- 強制待ち状態(Suspended)
  - 他タスクにより強制的に実行を中断させられた状態
- 二重待ち状態(Waiting & Suspended)
  - 待ち状態中に強制待ち状態への移行要求があった状態

タスクの状態遷移

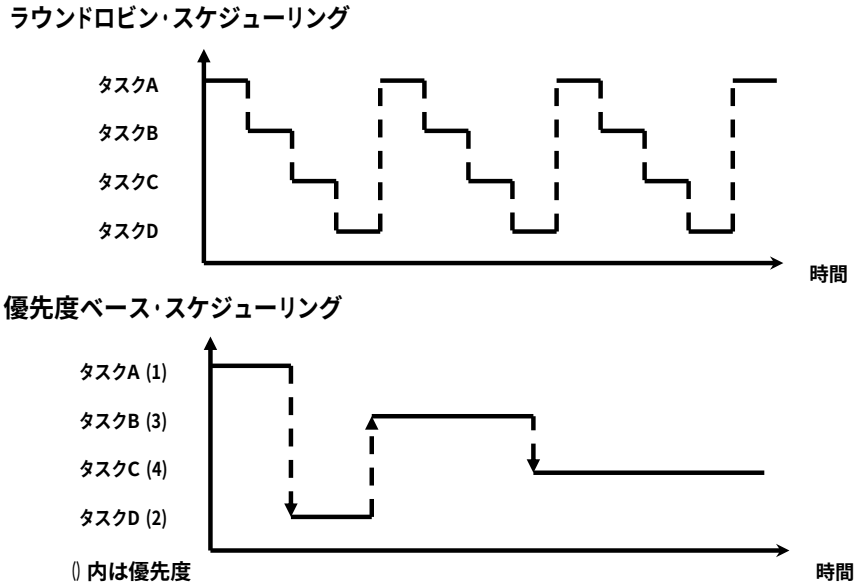
※斜体字はシステムコールに相当



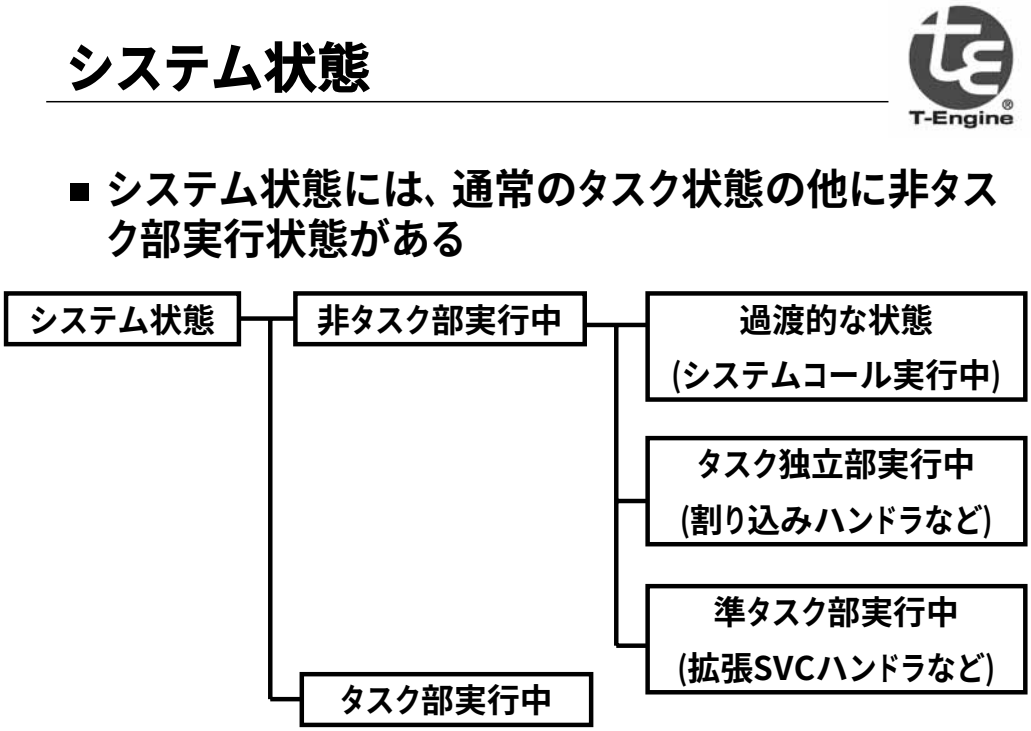
タスク スケジューリング

- 実行可能状態のタスクの中で、最も高い優先度のタスクを実行する
  - タスク優先度に基づくプリエンプティブな優先度ベーススケジューリング方式
  - 複数のタスクを公平に実行しようとするTSS (Time Sharing System) のスケジューリング方式とは異なる
- 優先度の高いタスクが実行状態または実行可能状態にある限り、それより優先度の低いタスクは実行されない
  - ただし、同じ優先度を持つタスク間の優先順位は、システムコールを用いて変更可能である。
  - 同じ優先度を持つタスク間では、先に実行できる状態 (実行状態または実行可能状態) になったタスクの方が、高い優先順位を持つ。

タスクスケジューリング(図解)



システム状態

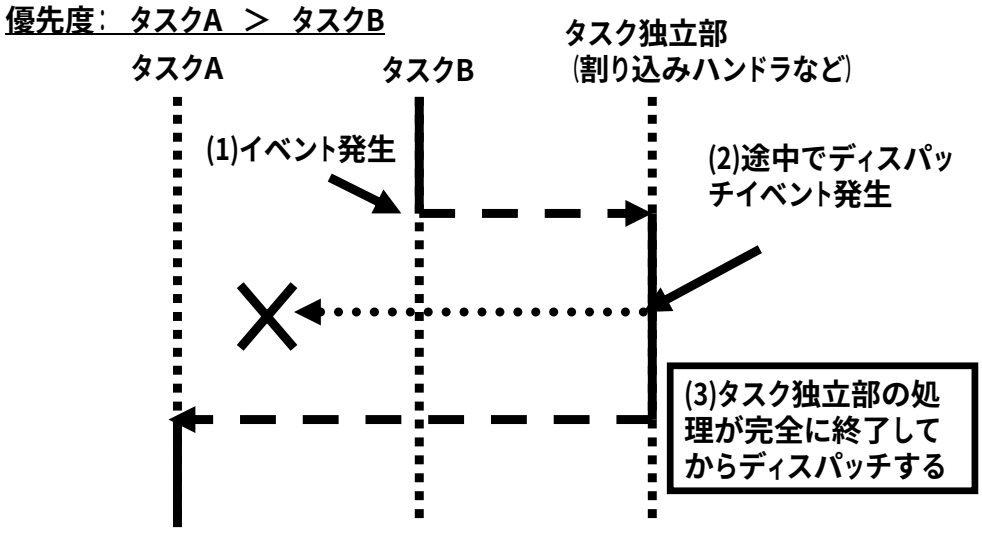


- システム状態には、通常のタスク状態の他に非タスク部実行状態がある

タスク独立部

- タスク独立部
  - タスク独立部とは、割り込みハンドラやタイムイベントハンドラなどのことを指す
  - 自タスクという概念が存在しない
  - タスク独立部の中から、自タスクを指定するシステムコールは発行できない
  - タスク独立部は、現在実行中のタスクを特定できない。そのため、タスク切り替え（ディスパッチ）が発生せず、ディスパッチが必要になっても、それはタスク独立部から抜けるまで遅延する。これを「遅延ディスパッチ」の原則と呼ぶ

遅延ディスパッチ



準タスク部

- 準タスク部
  - 準タスク部とは、拡張SVCハンドラ（OS拡張部で、サブシステムのAPIとなる部分）のことを指す
  - 準タスク部は、通常タスクの実行中と同様にディスパッチが発生する

T-Engine開発キットを利用した ソフトウェア開発方法

本章の概要および前提知識

- 本章の概要
  - T-Engine開発キットの概要、メリット、製品内容
  - T-Engine開発キットを用いたプログラム開発方法
  - サンプルプログラムのコンパイルと実行方法
  - Cygwinを利用したWindows版開発環境
- 本章を読むために必要な前提知識
  - LinuxやWindowsの操作に関する基本的な知識

・本章では、まずT-Engine開発キットの概要、メリット、製品内容などを説明します。その後、T-Engine開発キットを用いたプログラムの開発方法やサンプルプログラムのコンパイルと実行方法について説明し、最後にCygwinを利用したWindows版開発環境について説明します。

本章でご紹介する内容

- T-Engine開発キットとは
  - T-Engine開発キットの特長/内容
- T-Engine開発キットのソフトウェア構成
  - ターゲット用のソフトウェア
  - ソフトウェアの分類
    - T-Kernelベースのソフトウェア
    - プロセスベースのソフトウェア
  - 開発キット付属のソフトウェアツール
    - IMS (Initial Monitor System)
    - CLI (Command Line Interpreter)
- GNU開発環境の実際
  - [1] 開発の準備
  - [2] 開発の実際
  - [3] サンプルプログラムのコンパイルと実行
- Windows版開発環境
  - Cygwin利用のメリット
  - Cygwin環境の実際

・本章では、大きく分けて、開発キットの特長、構成、実際といった3つの内容をご説明します。

T-Engine開発キットとは

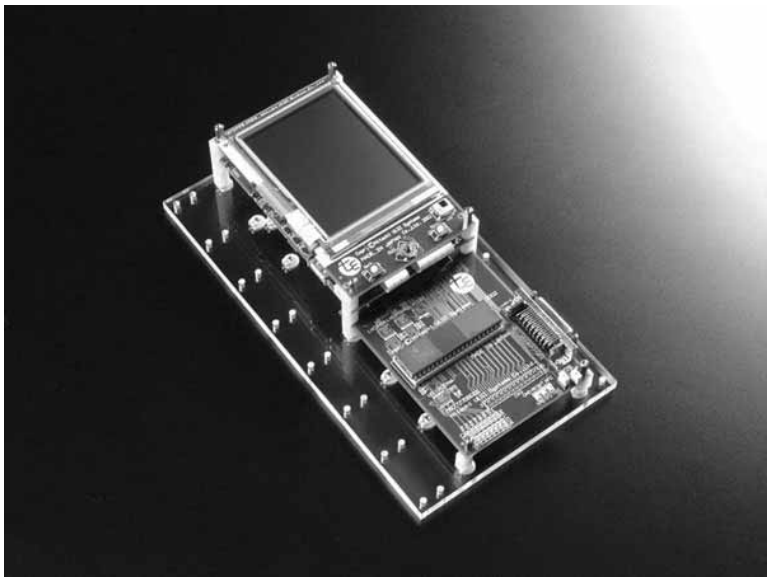
- T-Engine開発キットの特長
  - 買って来てすぐに使える、安価な開発用キット
  - ボード、OS、GNU開発環境、ドキュメントをパッケージ化
- 数多くのCPUをサポート
- 豊富なオプション製品
  - ハードウェア
    - LCDボード、デバッグボード、拡張LANボード
    - 拡張ユニバーサルボード、拡張FPGAボード
    - T-Engine開発ベンチ、拡張バス専用コネクタ
  - ソフトウェア、ミドルウェア
    - T-Linux/SH7751R評価キット &T299262;
    - PMC T-Shell開発キット (多漢字GUIミドルウェア)
    - KASAGO for T-Engineバイナリ評価ライセンス
    - Finger Attestor for T-Engine ...
- 問い合わせ先: パーソナルメディア株式会社
  - te-sales@personal-media.co.jp
  - http://www.personal-media.co.jp/te/

・T-Engine開発キットでは、ボードからソフトまでパッケージ化されているので、これを購入するだけですぐに開発を始められます。

・用途に応じて CPU を選べます。ソフト開発手法は各機種共通です。複数の T-Engine 開発キット間で、ソフトウェア開発のソースを共通にすることもできます。

・各社とのパートナ契約により、豊富なハードウェア、ミドルウェアをご提供します。

T-Engine/SH7727開発キットの外観



・これはT-Engine開発キットのハードウェアの写真です。このほか、ソフトウェアやドキュメントを入れた CD-ROM が提供されます。

T-Engine開発キット一覧

- **発売済みのT-Engine開発キット**

  SH-series and M32 CPU (RENESAS):
  T-Engine/SH7727, T-Engine/SH7751R, T-Engine/SH7760,
  μT-Engine/M32104

  MIPS CPU (NEC and TOSHIBA):
  T-Engine/VR5500, μT-Engine/VR4131, T-Engine/TX4956

  ARM CPU:
  T-Engine/ARM720-S1C (EPSON), T-Engine/ARM920-MX1 (Freescale),
  T-Engine/ARM926-MB8 (FUJITSU), μT-Engine/ARM7-LH79532 (SHARP),
  T-Engine/ARM720-LH7 (SHARP), T-Engine/ARM922-LH7 (SHARP)

- **今後発売予定のT-Engine開発キット**

  Nios CPU　μT-Engine/Nios II 開発キット (ALTERA)
  SH-series　T-Engine/SH7720開発キット (RENESAS)
  V-series　　μT-Engine/V850E-MA3開発キット (NEC)

・T-Engine開発キットは、組込み系の主要なCPU系列を網羅しています。

・同一アーキテクチャを持つ系列でも、CPU 性能や周辺デバイス等のバリエーションから選択可能です。

・標準 T-Engine のほか、μT-Engine の機種も増えてきています。μT-Engine は機器制御などに向いています。

T-Engine開発キットの内容

- ハードウェア
  - CPUボード、電源アダプタ、シリアルケーブル、(LCDボード)
- ターゲット用ソフトウェア
  - T-Monitor、T-Kernel、T-Kernel Extension、標準デバイスドライバ(ソース付)、サンプルアプリケーション
- 開発マシン用ソフトウェア
  - GNU開発環境(ソースデバッガgdbを含む)
- ドキュメント
  - 取扱説明書、ハードウェア説明書(回路図を含む)、T-Kernel仕様書など
- 開発用ホスト(PC-Linux機)があれば、T-Engine用のミドルウェアやアプリケーションを開発できる

・標準ドライバやサンプルプログラムはソースが付属しています。
・ドキュメントは PDF で提供されます。英語版のご用意もあります。
・開発用ホストマシンとしては、PC-Linux のほか、Windows (+ Cygwin) も利用できます。

開発キットのソフトウェア構成(1)

- ターゲット用のソフトウェア
  - T-Kernel (T-Kernel/OS, /SM, /DSを含む)
    - T-Kernel仕様書に準拠したリアルタイムカーネル
  - T-Monitor
    - T-Monitor仕様書に準拠したモニタ
  - T-Kernel Extension
    - T-Kernel/OS のサブシステム機能を利用して構築したOSの拡張部分
      - この拡張により仮想記憶OSを実現しており、開発環境としてファイルやプロセスを利用できる。BTRON3仕様に準拠した以下の機能が含まれる
        - プロセス/タスク管理、メッセージ管理、プロセス/タスク間同期通信管理、グローバル名管理、メモリ管理、ファイル管理、イベント管理、デバイス管理、時計管理、システム管理

・Extension は TRON ファイルシステムのほかに FAT ファイルシステムもサポートしています。
・MMU の無い μT-Engine の場合、Extension は利用できません。

開発キットのソフトウェア構成(2)

- デバイスドライバ
  - T-Kernel/SMのデバイス管理機能に基づいた下記のデバイスドライバが含まれている
    - PCカードマネージャ、USB マネージャ (バスドライバ)、
    - 時計(RTC)、コンソール(シリアル)、
    - システムディスク(ATA カード、USB)、
    - KB/PD(キーパッド、タッチパネル、USB KB/MOUSE)、
    - スクリーン(内蔵LCD)
- アプリケーション
  - 下記アプリケーションを提供
    - IMS (Initial Monitor System)
    - CLI (Command Line Interpreter)
    - フォーマッタなど

・フォーマッタでは、CF カードやUSB ストレージのフォーマットを行います。

ソフトウェアの分類

- 開発対象となるソフトウェアを以下の3つに分類
  - それぞれ、開発方法やオブジェクト形式などが異なる
    - モニタベースのソフトウェア
    - T-Kernel ベースのソフトウェア
    - プロセスベースのソフトウェア

・モニタベースのソフトウェアでは、T-Kernel を使いません。(他のOSを動かす場合など、特殊な用途でのみ利用します)

T-Kernelベースのソフトウェア

- T-Kernelの機能を使用するデバイスドライバやサブシステムなどのリアルタイムソフトウェア
- MMUを使用した環境で、システムの共通空間に常駐して動作する
- CLIまたはIMSから実行し、ロード時に自動的にリロケートされる

・T-Kernelベースのソフトウェアの場合、T-Kernel の API (tk_xxx_yyy) 、割込みハンドラ、I/O などが直接使用可能です。

・一つのメモリ空間内に複数のモジュールを動的にロードするため、ロード時に自動的にリロケートされます。

・MMU を使ってページングを行うため、効率良く物理メモリを利用でき、断片化(フラグメンテーション)が生じません。

プロセスベースのソフトウェア

- T-Kernel Extension上のプロセスとして、仮想メモリ上で動作するソフトウェア
- 一般のアプリケーションソフトウェアや、ライブラリ相当のミドルウェアなどが相当
- MMUを使用した環境で、プロセス固有の空間 (ローカル空間)にロードされて動作する
- 実行時の保護レベルはユーザモード(非特権モード)
  - T-Kernelの機能を直接利用することはできない
  - I/O空間を直接アクセスすることもできない

  → 逆に言えば、システムはアプリケーションから保護される。

・プロセスベースのソフトウェアの場合、各プロセスごとに独立したメモリ空間が用意されます。
・MMU を使ってページングを行うため、効率良く物理メモリが使用でき、断片化は生じません。
・T-Kernel や I/O を使えないかわりに、ミドルウェア(サブシステムやデバイスドライバや Extension)の機能を呼び出すことによってプログラミングを行います。

開発キット付属のソフトウェアツール(1)

- IMS (Initial Monitor System)
  - T-Kernelの初期タスクとして起動されるプログラムであり、次の機能を持つ
    - コマンドによる T-Kernel の各種状態の参照/操作
    - システムプログラム(サブシステム)のロード/アンロード
    - システムプログラム(プロセス)の実行
    - コマンドファイルの実行
    - システムの初期起動コマンドファイル
      - STARTUP.CMDの自動実行

・IMS は T-Kernel ベースのモニタです。一方、次ページで説明する CLI はプロセスベースのモニタです。

・タスク、セマフォなどのオブジェクトの参照機能を利用できます。

・lodspg(ロードシステムプログラム)コマンドにより、プログラムモジュールをロードします。

・STARTUP.CMD を書き替えれば、電源起動時に自動的にロード、実行するモジュールを指定できます。

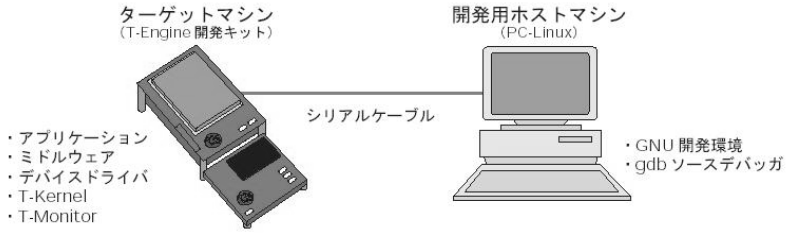
開発キット付属のソフトウェアツール(2)

- CLI (Command Line Interpreter)
  - システムの1プロセスとして起動されるプログラム。UNIX の sh に相当し、次の機能を持つ
    - コマンドによるファイルを中心とした各種操作
    - システムプログラム(サブプロセス)のロード/アンロード
    - アプリケーションプログラムの実行
    - コマンドファイルの実行

・CLIはプロセスベースのモニタです。
・プロセスの参照機能があります。
・UNIXライクな、豊富なファイル管理機能(ls, rm, etc)を利用できます。
・recv(レシーブ)コマンドによりホストからプログラムなどのファイルを転送します。

GNU開発環境の実際

- 開発用マシンとターゲットマシンが別(クロス開発)
  - PC-LinuxまたはWindows (Cygwin) 上で動作する
    - ◆ GNUベースの開発環境をご提供
      - RedHat Linux 7.1 / 7.3 / 8.0 / 9.0 および
      - RedHat Professional Workstation, Cygwin で動作
  - このほか、perl と GNU make も必要
  - (プラットフォームにインストール済のものを利用)
  - デバッグにはソースデバッガgdbを利用



・T-Engine開発キットには、T-Engine用に調整したGNU開発環境が含まれます。これは、既存のGNUに対して、ライブラリの追加やOS環境の相違に対する修正などを行ない、T-Engine 用の開発環境として構築しなおしたものです。

・gdb によってソースレベルのデバッグが可能です。

開発の準備(1)

[1] 開発の準備

(1)GNU開発環境のインストール

CD-ROM の GNU 開発環境パッケージ
+Linux 用プラットホーム共通部分
te.Linux-i686.common.tar.gz
+Linux 用 SH 対応部分
te.Linux-i686.sh.tar.gz
+T-kernelリソース部分
te.resource.sh.tar.gz

を /usr/local/te などで展開し、環境変数を設定し、サンプルのコンパイルで動作確認

・基本的には、ファイル展開の作業を行うだけですので、わずか数分の簡単な操作です。
・環境変数の設定は、/usr/local/te 以外の場所に開発環境を展開した場合に必要です。

開発の準備(2)

(2)T-Engine Board とデバッグコンソールの接続

- 付属シリアルケーブルと、開発用Linuxマシンを接続。
- この状態でCLI, IMS, T-Monitorを操作可能。

(3)作業用ディスクの準備

- ATA, CF, USB ストレージデバイスなど、バイナリプログラムを保存しておくためのディスクを準備する。

・シリアルは最高 115200bps で動作します。
なお、さらに高速でファイル転送する方法も用意されています(CF 経由、ftp など)。

・開発したプログラムは、ホスト側のPCからいったん T-Engine 側の作業用ディスク(CFなど)に転送し、それをメモリ空間上にロードして実行、デバッグします。

開発の実際

[2] 開発の実際

- T-Kernelベースのソフトウェアの場合
  - リロケータブルなオブジェクトが生成
  - バイナリを CLI の recv コマンドで作業用ディスクにファイルとして保存
  - CLI または IMS の lodspg コマンドでロードと実行を行う
- プロセスベースのソフトウェアの場合
  - リロケータブルなオブジェクトが生成
  - バイナリを CLI の recv コマンドで作業用ディスクにファイルとして保存
  - CLI または IMS で、直接プログラム名を指定してロードと実行を行う

・recv は「レシーブ」と読みます。
・lodspg は「ロード システム プログラム」と読みます。
・unlspg 「アンロード システム プログラム」コマンドにより、アンロードも可能です。

・recv は「レシーブ」と読みます。
・プロセスは、終了後に自動的にアンロードされます。
・各サブシステムは、プロセス単位でリソース管理を行います。このため、プロセスが終了する時には、自動的にリソースのクリーンアップが行われます。

サンプル(1)

[3] サンプルプログラムのコンパイルと実行
【drawsamp編】

drawsamp(簡単なお絵描きプログラム)をコンパイル、ロード、実行した
あと、ソースの一部を変更して、再コンパイル、ロード、実行を行う。

1.開発環境上で /usr/local/te/kappl/drawsamp/sh7727 に移動
>$cd /usr/local/te/kappl/drawsamp/sh7727

2.一旦 make clean したあと make して、オブジェクトを生成
>$make clean
>$make

3. gterm を起動
>$/usr/local/te/tool/etc/gterm -B -l/dev/ttyS0

・gterm「ジーターム」はシリアル回線用の端末ソフトであり、T-Engine開発キットに付属しています。
gterm を使えば、recv (レシーブ)コマンドで自動的にファイル転送を行うなど、操作が簡単になります。

・gterm の代わりに他の端末ソフトを使うこともできます。

サンプル(2)

4. 必要であれば作業用ディスクをアタッチして、そこに移動
   >[/SYS]% att pca0 work
   >[/SYS]% cd /work
   >[/work]%
5. recv コマンドでオブジェクトをロード
   >[/work]% recv -d /usr/local/te/kappl/drawsamp/sh7727/drawsamp
6. lodspg コマンドで実行
   >[/work]% lodspg drawsamp
7. 開発環境上で vi や emacs を使い、
   /usr/local/te/kappl/drawsamp/src/drawsamp.c を開き、
   draw_line の引数の BLACK を col16(0xFF,0x00,0x00) に
   変更して保存
8. 上記同様に、再度コンパイル、転送、実行

- 「アタッチ」とはファイルシステムの接続を行うものであり、UNIXの mount に相当する操作です。
- CF 等にシステムを格納した「起動ディスク」であれば、上記のアタッチの操作は不要です。システムを含まない「作業ディスク」の場合は、上記のようにアタッチの操作が必要です。一般的には、「起動ディスク」を作成して利用する方が何かと便利です。
- 7. 8. はソースを修正して再度実行する例です。

Windows版開発環境

- Cygwin: Windows版のGNU実行環境を利用
- Linux版開発環境との違い
  - Linux版
    - T-Engine ←→ Linux Host [標準開発環境]
    - エディタ[vi,emacs]
    - (←→ Windows Host ICE)
  - Cygwin版
    - T-Engine ←→ Windows Host Cygwin [標準開発環境]
    - Windows用エディタ
    - (ICE)

・ICE をつなぐ場合、多くの ICE は Windows にのみ対応していますので、開発用ホストを Linux とした場合には、別に Windows マシンを用意する必要があります。

・また、一部のシリアルデバイス等は、Windows からは使えても、Linux からはドライバの問題により使えない場合があります。

Cygwin利用のメリット

- Linuxマシンが不要になる
  - T-Engineの開発環境として、新たなLinuxマシンを用意する必要がない
  - ICE等を使用する場合にも、別のWindowsマシンを用意する必要がない
- Linux固有の知識が不要
  - ユーザー管理、ネットワーク、シェルスクリプトなど、T-Engineのプログラム開発に必要なLinux固有の設定作業が不要になる

- Windowsには前ページで説明したようなメリットがあるほか、Linux よりも Windows に慣れている技術者が多いため、T-Engine開発キットの発売当初より、Windows 上で動作する開発システムが望まれていました。そのようなご要望に対応したのが Cygwin です。

Cygwin環境の実際

- Cygwin + Cygterm + TeraTermPro
  - Cygterm を利用することにより TeraTermPro との間を Telnet で接続する
  - 2つのフリーソフトウェアの追加で済む
  - TeraTermPro は T-Engine のターミナルとしても利用可

    T-Engine ←→ Windows host TeraTermPro

    ↑

    Cygwin[Cygterm]

    Cygwin[標準開発環境]

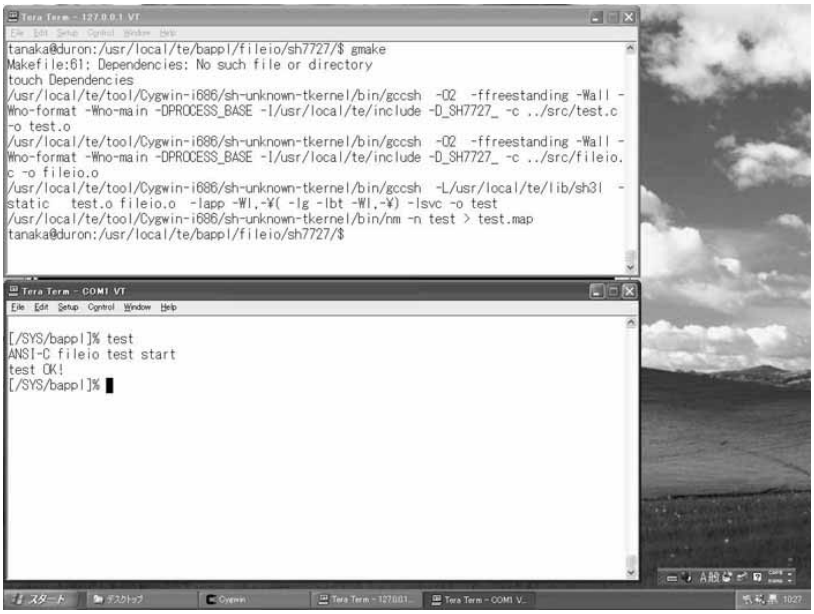
  - エディタ等は使いなれた Windows のものを(EUC 対応)

・Cygwin は Windows 上でGNU環境をエミュレーションするものです。

・Cygwin 版の開発環境における開発のスタイルは、Linux 版とほとんど同じです。

・標準の端末ソフトを使っても構いませんが、漢字コードなどの面で充分でない場合があるので、Cygterm を間に介して TeraTermPro を使う方法を推奨しています。

Windows上での開発画面



・上側の TeraTerm は Cygwin + GNU 開発環境でコンパイルしているところです。
・下側の TeraTerm は T-Engine と接続して CLI を使っているところです。

T-Kernel利用によるソフトウェア開発方法

本章の概要および前提知識

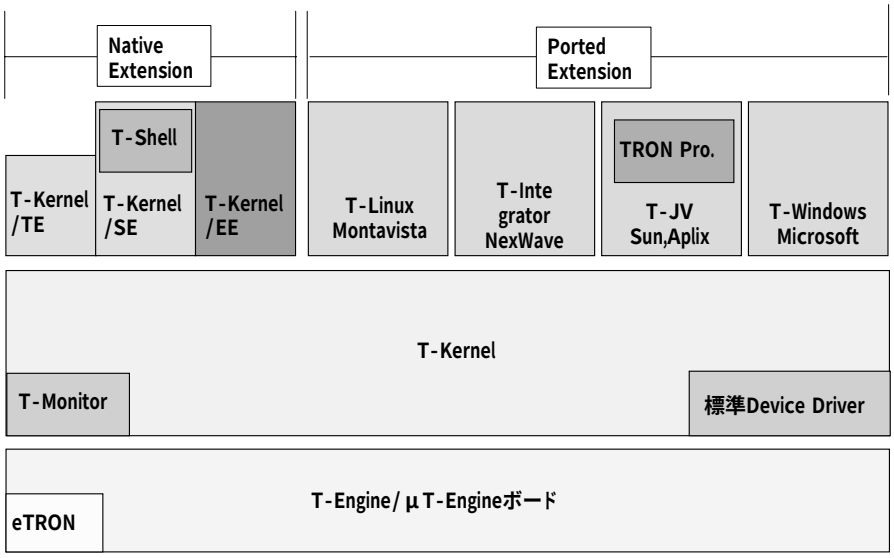
- 本章の概要
  - T-Kernelのソースコードをどのように使うか
  - T-Licenseと製品への組み込み
  - T-Kernelの移植事例
  - まとめ

- 本章を読むために必要な前提知識
  - リアルタイムOSを使えること
  - C言語のプログラムの読み書きができること
  - 組み込みソフトウェア開発ができること
    - プログラミング⇒ビルド⇒デバッグ

T-Kernelソースコードをどのように使うか

- 組み込み製品への適応
- リアルタイムOSの研究･開発
- 教育用のベースOSとして
- 新規マイコンのエントリOSとして

T-Engineソフトウェアアーキテクチャ



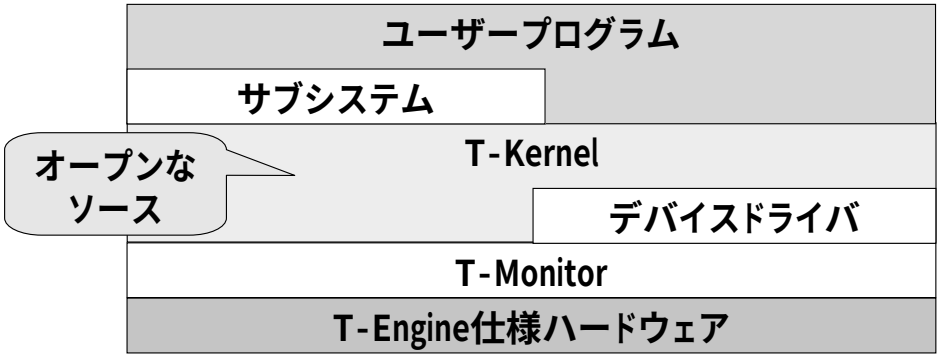
2005.09.13 version 1.01 64

T-Engineのソフトウェアアーキテクチャは図に示すように、
・様々なベンダからリリースされているT-Engine/μT-Engine仕様プラットフォーム
・T-Kernel及びデバイスドライバ、T-Monitor
・OSの機能を拡張するExtension
から構成されています。

T-Kernel/Extensionには
・T-Engineフォーラムが開発・リリースを行うNative Extension
(TE:Tiny Extension,SE:Standard Extension,EE:Enterprise Extension)
・ベンダが開発・リリースを行うPorted Extension
があります。

T-Kernelの意味と意義

- T-Kernelはオープンなソースコード
  - 様々な機器への適応改変ができる

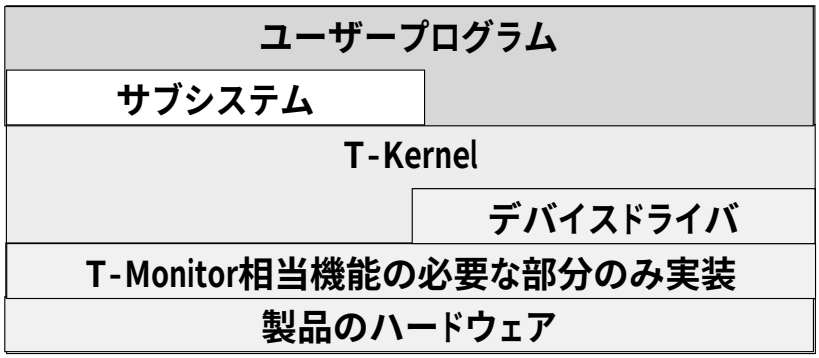


T-Kernelを使うことで様々な機器への適応化が可能となりました。

図はT-Engine開発キットでの構成を示しています。

オープンなソースであるT-Kernelを変更、設定することにより
T-Engine開発キット以外のハードウェアに適応化できます。

T-Kernelを製品に適応する



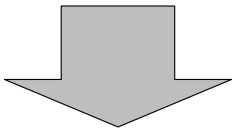
実製品にT-Kernelを適応するには、

①T-Engine仕様ハードウェア ⇒ 製品ハードウェア
②T-Monitor ⇒ T-Monitor相当機能の必要な部分のみ実装
③T-Engine用デバイスドライバ ⇒ 製品ハードウェアに対応したデバイスドライバ

の作業が必要になります。

適応化の範囲

- 製品ハードウェアへの適応化
- 性能、機能改善のチューニング
- ミドルウェア、開発環境への適応化



T-License

(T-Kernelソースコードのライセンス契約)

T-Kernelを実組み込み製品に適応するには、3つの項目が重要となります。

⑴ 製品ハードウェアへのソフトウェアの適応化
⑵ システム性能を満たすための性能、機能改善のチューニング
⑶ システム製品で使用するミドルウェアをどう使うか、またシステム開発環境をどうするか

この3つの項目を検討するためには、T-Kernelソースコードの使用条件であるT-Licenseについて理解することが必要となります。

T-Licenseとは (1)

◆ 組み込み開発に、抵抗無く使える契約

GPL → ソース公開の義務 →企業先端技術の漏洩
(GPLは対象が違う)

T-License → 改変しても、ソース公開の義務は無い

・無償で利用して良い。無償で最終製品に組み込んで良い。
【条件】- T-Kernelを利用した旨、表示すること

・改変して使って良い。

・改変したり追加したコードを公開しなくて良い。

※ 最終的には T-License原文 (フォーラムHP掲載) をご確認ください。

T-Licenseの重要なポイント(その1)

・T-LicenseはGPLと違い、ソース公開の義務はありません
・ロイヤリティーフリーである
・ソースコードを改変して使ってよいが、T-Kernelを使用した旨を表示することが必要
⇒T-Engine Applianceといい、ロゴが規定されています
・改変したソースコードは公開の義務はありません。

T-Licenseとは (2)

◆ T-Kernelとしてシングルソースを維持できる契約

ミドルウェア流通を実現することが大切

改変されたOS上でミドルウェア蓄積ができなくしたい

ソースコード配布に一定の条件を付与

・改変したソースコードを有償無償を問わず第三者にパッチで配布して良い。

〔条件〕- フォーラムA会員であること

- フォーラムに対し所定の手続きを経て改変版配布者として承認されていること
- 「改変されたソースコード」の名称と概要をフォーラムに通知し登録していること

・改変版配布者から配布された改変ソースコードを他に再配布できない

※ 最終的には T-License原文 (フォーラムHP掲載) をご確認ください。

T-Licenseの重要なポイント（その2）

T-Kernelはソフトウェア再生産性の向上も目指しているので、ミドルウェア流通を妨げないために シングルソースとなっています。

T-Kernelのビルド手順

1. GNUツールのセットアップ
   - T-Engine開発キットのGNUツールを使用
   - FSF等からソースコードをダウンロードしてビルドして使用
2. T-Kernelソースコードの準備
3. ライブラリの構築
4. カーネルの構築
5. 設定ファイルの構築

T-Kernelのビルドには、標準開発環境GNUが必要となります。
ビルドに使用するGNUツールにはT-Engine開発キット付属のGNUツールを使用するか、FSF等からソースコードをダウンロードし、ビルドして使用する方法もあります。

また、T-Kernelビルドについての詳細は、ソースコードを入手する際に同時に入手できる『T-Kernelソースコード説明書』に記載されております。

FSF･･･フリーソフトウェア財団 (Free Software Foundation)

T-Kernelの開発事例

- ルネサス　SH7145（SH-2）へのT-Kernel移植
  - SH7727(SH3-DSP)用T-Kernelをベース
    - コアアーキテクチャ違い、メモリマップ違い
  - ハードウェアはμT-Engine仕様
    - オリジナルは標準T-Engine仕様
  - ハードウェア依存部のみ移植した「単純移植」
    - カーネル基本部を変更する「改変」ではない

 71

T-Kernelの開発事例：ルネサス製SH7145(SH-2)での開発

・既公開T-KernelでSH7727等SHファミリCPUをサポートしていますので、
SH7727版を参考にしました。
但し、コアアーキテクチャ、メモリマップなどを考慮する必要がありました。

・開発ターゲットハードウェアはμT-Engine仕様であり、
標準T-Engine仕様のSH7727と差異を考慮する必要がありました。

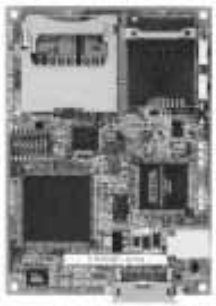
・移植にあたり短期間で開発を行うために、
カーネル基本部を改変せず、システム依存部のみを変更する「単純移植」で
開発を行いました。

移植したハードウェアの仕様

【 概略仕様 】

μT-Engineボード仕様に準拠

CPU	ルネサスSH7145F (動作周波数50MHz)
RAM	内蔵8kB、外付け=1MB
フラッシュメモリ	内蔵256kB、外付け=1MB
シリアルI/O	内蔵資源(SCI)
Compact Flash	1スロット
SDカードインタフェース	1スロット
eTRONチップインタフェース	SIMカードコネクタ
拡張バスインタフェース	
カレンダークロック	



ソフトウェア構成

- ターゲット用のT-Kernel (T-Kernel/OS, /SM, /DSを含む)
  - T-Kernel仕様書に準拠したリアルタイムカーネル
  - 2004/7月に公開されたT-Kernel v1.01.00を「単純移植」
- T-Monitor
  - T-Monitor仕様書に準拠し必要最小の機能リファレンスデザイン として設計
- デバイスドライバ
  - T-Kernel/SMのデバイス管理機能に基づいた各種デバイスドライバ
- ミドルウェア
  - T-Kernel仕様書に準拠した各種ミドルウェアを利用可能とする

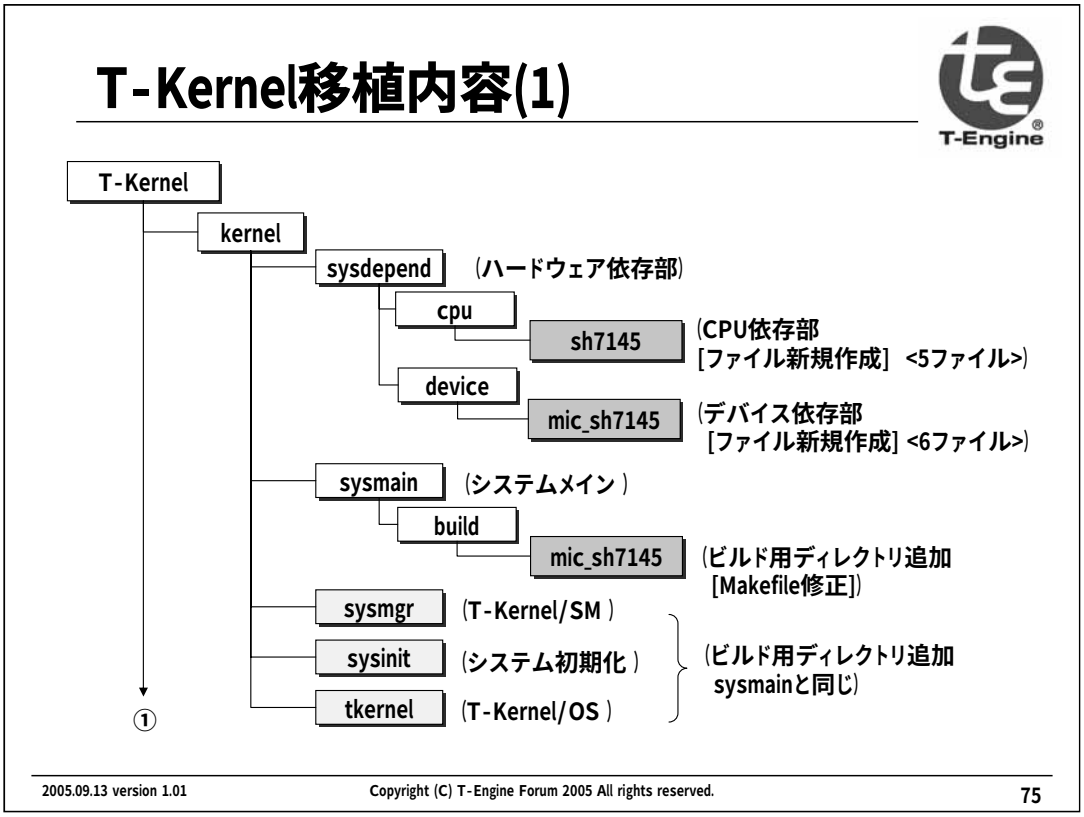
開発ターゲットでのソフトウェア構成です。

開発の方針

- T-Kernel
  - ハードウェア依存部のみの変更（単純移植）
  - 既公開SuperHファミリ用ソースコードで使えるものは流用
- T-Monitor（デバッグ環境）
  - 実応用では必須のJTAGエミュレータの使用を考慮
- クロスツール
  - 開発環境はSuperHファミリで実績あるKPIT GNUtoolへ
    - URL：http://www.kpitgnutools.com/
  - ルネサス製C/C++コンパイラとの親和性も考慮
- ホストOSもLinuxからWindowsへ
  - Linuxでもビルド可能

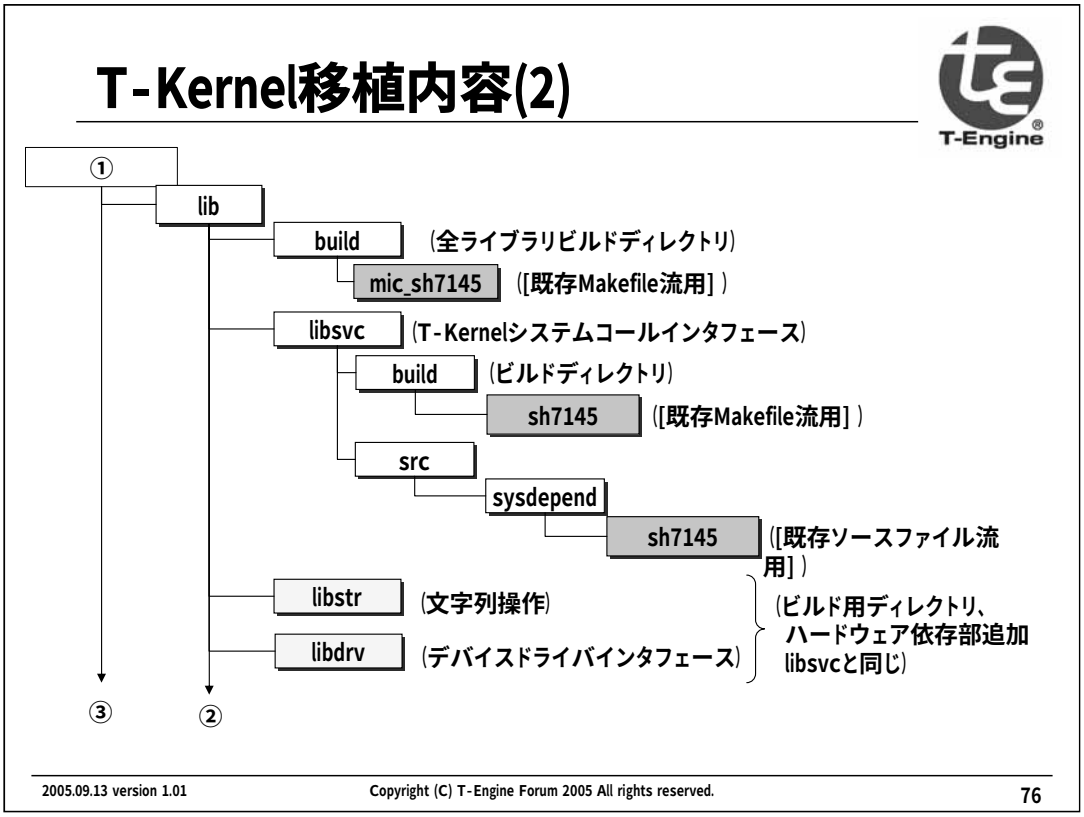
開発にあたって、以下のように方針を決めて開発を行いました。

(1)T-Kernelは単純移植で、SH7727用で使えるソースコードはそのまま流用
(2)T-Monitorのうち、デバッグ環境については組み込み製品への適応
　及び、半導体ベンダ、ツールベンダのJTAGエミュレータの使用を考慮
(3)クロスツールはSHファミリで実績あり、かつ無償サポートを受けられる
　KPIT社製GNUSHツールを使用することにしました。
　またコンパイルオプションもルネサス製コンパイラとの親和性を考慮しました
(4)ホストOSも標準ではLinuxとなっていますが、JTAGエミュレータ使用を考慮、
　Windows(CYGWIN)環境にしました。



T-Kernelソースコードは機能モジュールごとのブロック構成となっています。
ビルドするディレクトリはソースディレクトリとは別ディレクトリになっています。
(詳細はT-Kernelダウンロード時にダウンロードディレクトリにあるソースコード説明書に記載があります)

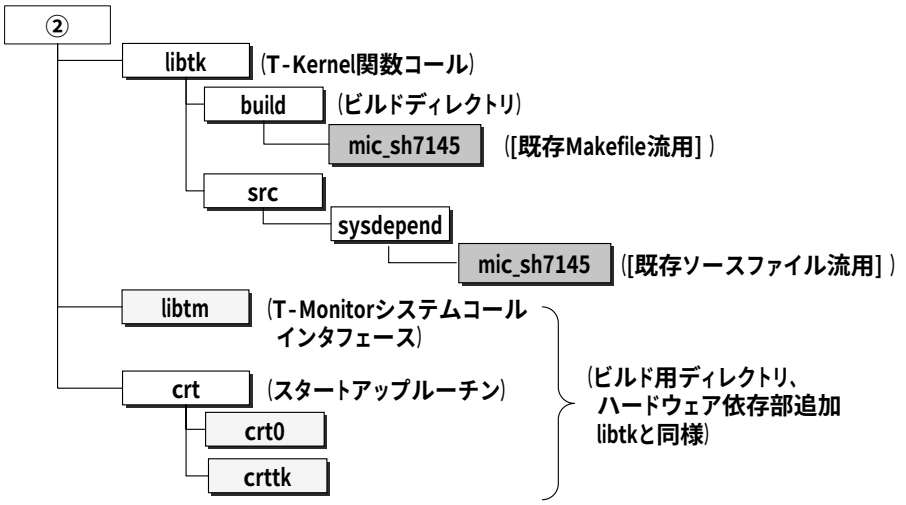
kernelディレクトリでは、
・sysdependディレクトリ
(タスクのディスパッチ、割り込みハンドリング、システム依存システムコール」などを行うハードウェア依存部)
⇒CPUディレクトリに”sh7145”作成し、必要なファイルを作成
⇒deviceディレクトリに”mic_sh7145”ディレクトリを作成し、必要なファイルを作成
・sysmain,sysmgr,sysinit,tkernelディレクトリ (カーネル基本部)
⇒ビルドディレクトリ (”mic_sh7145”) 作成し、Makefileをコピー・修正
の作業を行いました。



libディレクトリ変更内容(1)

build、libsvc、libstr、libdrvディレクトリ
⇒ビルドディレクトリとMakefileのみ追加しています。

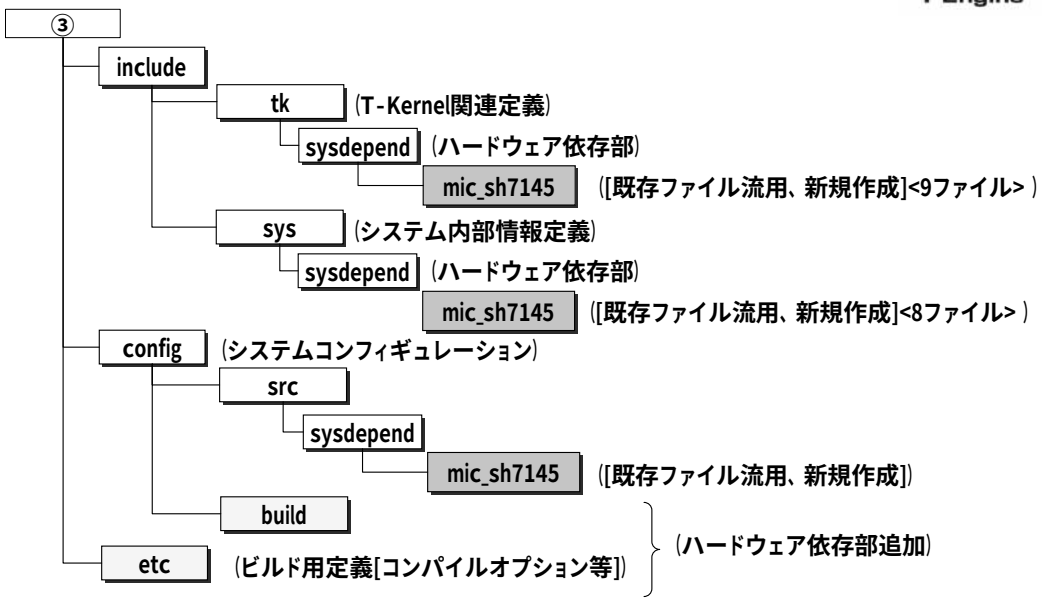
T-Kernel移植内容(3)



libディレクトリ変更内容(2)

libtk(T-Kernel関数コール)、libtm、crtディレクトリ
⇒ソースディレクトリに必要なファイルを追加しました
⇒ビルドディレクトリを追加し、必要なファイルを追加・変更しました

T-Kernel移植内容(4)



includeディレクトリ(tk、sysディレクトリ)
⇒システム依存部にファイル作成(SH7727版コピー後修正)しました

configディレクトリ(ROMINFO情報作成)
⇒システム依存部にファイル作成(SH7727版コピー後修正)しました

etcディレクトリ(コンパイルオプションなど定義)
⇒システム依存部にファイル作成しました

T-Monitor実装内容

機能概要	詳細	実装内容
システム機能	ハードウェア初期化、 システム起動処理	BSC設定、クロック設定、INTC初期化等
	例外、割り込み処理	割込み受付、ベクターテーブル設定など
プログラム サポート機能	モニタサービス関数	T-Kernel動作に必要な関数のみ実装
デバッグ機能	メモリ操作、レジスタ操作等	未実装(JTAGエミュレータ使用のため)

T-Monitor仕様については3つの機能があります。
T-Kernelの動作には全ての機能を実装する必要はありません。

T-Monitorサービス関数実装一覧

関数名	機能	実装内容
tm_monitor(void)	モニタへ入る	未実装
tm_getchar(INT)	コンソール1文字入力	実装
tm_putchar(INT)	コンソール1文字出力	実装
tm_getline(UB *)	コンソール1行入力	実装
tm_putstring(UB *)	コンソール1行出力	実装
tm_command(UB *)	モニタコマンド実行	未実装
tm_readdisk(UB *.INT,INT,VP)	ディスク読み込み	未実装
tm_writedisk(UB *.INT,INT,VP)	ディスク書き込み	未実装
tm_infodisk (UB *,INT,INT)	ディスク情報取り出し	未実装
tm_exit(INT)	システム終了	未実装
tm_extsvc(INT,INT,INT,INT)	拡張サービス機能	未実装

T-Monitorモニタサービス関数は、T-Kernelで使用する関数のみ実装を行います

T-Kernel移植のまとめ

分類	移植内容
ビルド関連	SH7145用ビルド用設定追加、ビルドディレクトリ作成 リンカスクリプト変更（メモリマップ変更等）
T-Kernel	割込み/例外・ディスパッチ処理部作成 マイコン依存周辺I/Oアドレス定義作成 マイコン依存アセンブラマクロ、型定義作成 システムコンフィギュレーション情報作成 システム情報（ROMINFO,SYSINFO）作成 マイコン依存インタフェースライブラリ作成 キャッシュ、アドレス変換部作成 システムタイマ処理作成
T-Monitor	システムスタートアップ処理作成 （マイコン周辺I/O初期化、レジスタ初期化等）

T-Kernel移植のまとめとして作業内容をまとめています。

本表は開発例での内容ですが、他のCPU(及びシステム）で開発を行う場合にも有効な内容です。

移植のポイント(工夫点)

- T-KernelとT-Monitorの分離
  - 様々なハードウェアに対応させ、可搬性をよくするため
- メモリマップ
  - マイコン内蔵RAM、フラッシュメモリの効率的な活用
    ⇒割込み応答時間を短くするため、ベクタテーブルは内蔵RAMへ
    (no waitアクセス)
- 既存ソースコードの積極的な活用
  - インタフェースライブラリなどそのまま使用できるものは流用(SH7727用等)
- 今後の拡張のために
  - T-Monitor機能はT-Kernel動作に必要なもののみを実装したが、追加機能も実装できるような仕組みを用意

移植のポイント(工夫点)

CPUやシステムに依存する部分はありますが、上記の項目を挙げることができます。

T-Kernelのデバッグ

- T-Kernelのデバッグ機能を使う
  - T-Kernel/DSシステムコールによるデバッグ
  - Syslogによるログ機能
- マイコンベンダの開発環境を使う
- ツールベンダの開発環境を使う

T-Kernelを利用した開発を行うにあたりどのようにT-Kernelをデバッグすればよいかも重要です。

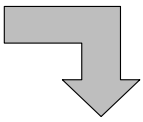
・T-Kernel/DSにはオブジェクト参照するシステムコール、フックルーチン処理などのシステムコールが用意されています。

・T-Kernel本体にもSYSLOG（システムログ）機能があります

また、マイコンベンダやツールベンダの開発環境を使用することも効果的です。

まとめ

- T-Kernelは組み込みシステム開発に有用
  - 既存サポートCPUも多く流用は容易
  - ソースコードの可読性の良さ
- 今後に向けて
  - 応用製品の適応化（への指針）
    - ⇒サブセット化
    - ⇒性能・サイズのチューニング



但し、ミドルウェア流通は維持できること

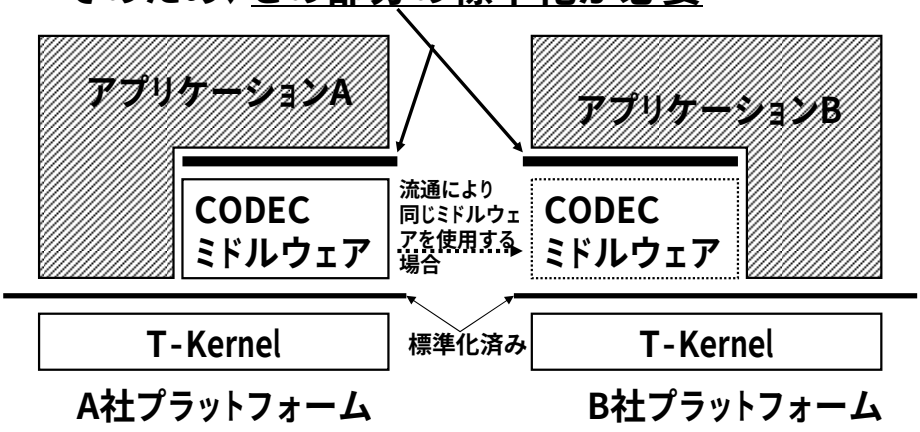
T-Format入門

本章の概要および前提知識

- 本章の概要
  - T-Formatの概要について
  - T-Formatの構成について
- 本章を読むために必要な前提知識
  - C言語に関する基本的な知識

ミドルウェア流通のために

- シンボル、ファイル名などの名前の重複を避けたい
- そのため、この部分の標準化が必要



T-Formatとは(1)

- 円滑なミドルウェア流通に必要な仕組みの一つ
  - 流通するソフトウェア間の不要な競合を避ける仕組み
    - バイナリコード形式
    - ソースコード形式
    - ドキュメント形式
  - コンパイルに必要なMakefile形式、ヘッダファイルやライブラリファイルを格納するディレクトリ構造も規定
- 流通するミドルウェアが増えるほど、重要性が増す

T-Formatとは(2)

- T-Engine, T-Kernel上で動作するミドルウェアやアプリケーションソフトウェアのコード形式を標準形式として規定する
- TCP/IP、FATファイルシステムのような、極めて代表的なミドルウェアは具体的な標準APIを定める。それ以外は標準的なコード形式としてT-Formatを規定する
- C言語 (T-Format/C)、C++言語 (T-Format/C++)、Java言語 (T-Format/Java)を基本言語とする

T-Formatの構成

- ソースコードスタイルガイドライン
  - ソースコードのスタイル形式
  - ライブラリ名、シンボル名など
- 標準バイナリコード形式
  - 実行コード形式、デバッガシンボル形式を規定
  - 標準リファレンス環境の出力を標準バイナリと規定
- 標準ドキュメント形式
  - 流通するソフトウェアに添付するドキュメントの種類と形式を規定
  - 現在策定中

ソースコードスタイルガイドライン

- 対象
  - ミドルウェア
  - グローバルシンボル
    - ライブラリファイル名、外部参照シンボル名、マクロ名、型名、クラス名など、ベンダ間の名前の重複を避ける
  - C言語、C++言語、Java言語
    - 現在はC言語から標準化を進めている

構成

- T-Engineベンダコード(以下 TVC)
- ミドルウェアファイル名
- exportされる関数名
- exportされる大域変数名
- 関数、大域変数で用いられるデータ型名
- ヘッダファイルに含まれる定数マクロ名
- 通称定義
- ミドルウェアパッケージ名
- バージョン表現

T-Engineベンダコード(TVC)

- T-Format/T-Engine仕様を満たすソフト/ハードを提供するベンダに一意なコードを割当てる
- TVCはベンダ（T-Engineフォーラム会員）の申し出に基づき、T-Engineフォーラムが割当てる
- ‘a’～‘z’, ‘0’～‘9’の3文字以上8文字以内
- 例
  - YRPユビキタス・ネットワーキング研究所
    - unl

ミドルウェアファイル名

- 流通させるバイナリとヘッダファイルが対象
- ライブラリファイル名
  - “lib”<ミドルウェア名>”.”<拡張子>
- その他ファイル名
  - <ミドルウェア名>“.”<拡張子>
- ミドルウェア名の定義
  - <ベンダ名>“_”<機能名>[“_”<詳細機能名>]
- 例
  - ベンダ<unl>, ミドルウェア名<mpeg2>
    - libunl_mpeg2.a
    - unl_mpeg2_basic.h

exportされる関数名

- exportされる関数名
  - <ミドルウェア名>“_”<関数識別子>
- 関数識別子
  - ‘A’～‘Z’, ‘a’～‘z’, ‘0’～‘9’ を使った2文字以上の文字列
- 例
  - mpeg2エンコード関数
    - void unl_mpeg2_encode();

exportされる大域変数名

- exportされる大域変数名
  - <ミドルウェア名>“_”<変数識別子>
- 変数識別子
  - ‘a’～‘z’, ‘0’～‘9’ を使った2文字以上の文字列
- 例
  - mpeg2ライブラリで使われる変数 “etronid” の場合
    - char unl_mpeg2_etronid[16];

データ型名

- 関数、大域変数で用いられるデータ型名が対象
- データ型名
  - <ミドルウェア名>“_”<データ型識別子>
- データ型識別子
  - ‘a’～‘z’, ‘0’～‘9’ を使った2文字以上の文字列
- 例
  - typedef unsigned long unl_mpeg2_ulong;

定数マクロ名

- ヘッダファイルに含まれる定数マクロ名が対象
- 定数マクロ名
  - <ミドルウェア名>“_”<マクロ識別子>
- 例
  - #define UNL_MPEG2_MAXID 255

通称定義

- 推奨仕様
  - 「名前が長い」、「一般的な名称と異なる」などの問題を解消する
  - 短く解りやすい通称を定義する
- 通称定義用ヘッダファイル
  - 通称定義用ヘッダファイルの命名規則がある
  - <ミドルウェア名>“_common.”<拡張子>
- 例
  - unl_mpeg_common.h
    - #include “unl_tcpip.h”
    - #define socket unl_tcpip_socket

ミドルウェアパッケージ名

- ミドルウェアパッケージ名を付けることができる
- 主にソフトウェア開発環境におけるパッケージの検索処理を想定している
- パッケージ名
  - <ミドルウェア名>“.”<拡張子>
- ミドルウェア名
  - <ソフトウェアベンダ名>“_”<機能名>
    [“_”<詳細機能名>]“_”<バージョン表現>
  - バージョン表現は次節で説明
- 例
  - unl_mpeg2_codec_10311

バージョン表現

- 5桁の数字と文字で表現する　xyyzz
- x (0～9)　メジャーバージョン番号
  - バックワードコンパチビリティがない時にインクリメント
- yy(a0～z0, 00～99)　マイナーバージョン番号
  - APIに変化はあるがバックワードコンパチビリティが確保されている場合にインクリメント
  - a0～z0はプレリリース、00～99は正式リリースに使用
- zz(00～99)
  - 外部から見てコンパチビリティが保たれている場合にインクリメント

デバイスドライバの実装方法

本章の概要および前提知識

- 本章の概要
  - T-Kernelにおけるデバイスドライバの考え方とドライバ開発作業の流れに関する説明
  - sdi,gdi という枠組みを用いたドライバ実装方法の説明
- 本章を読むために必要な前提知識
  - リアルタイムOSについての一般的な知識

・本章の前半では、T-Kernelにおけるデバイスドライバの考え方と、ドライバ開発作業の流れに関して説明します。

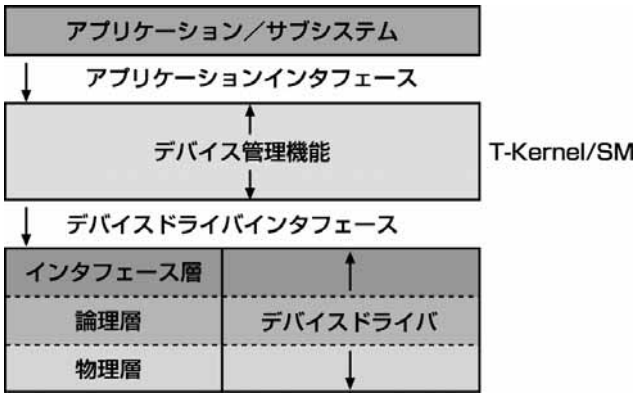
・本章の後半では、sdi および gdi といった枠組みを用いたドライバの実装方法に関して、具体的に説明します。

本章でご紹介する内容

- ドライバの枠組み
- ドライバ開発の流れ
- ドライバインタフェース層
- ドライバ内の各関数の役割

・なし

ドライバの枠組み



- アプリケーション開発者側のメリット
  - どんなドライバでも統一された手法(アプリケーションインタフェース)で呼び出せる
- ドライバ開発者側のメリット
  - 作成したドライバをミドルウェアとして流通できる
  - 他機種への対応も容易(基本的には物理層のみ入れ替えればよい)

・組込みの世界では、以前からデバイスドライバのインタフェースが各社まちまちだったため、ドライバやそれを使ったミドルウェアの流通が困難な状況でした。この問題を解決するため、T-Engineプロジェクトでは、T-Kernelの仕様の中でデバイスドライバとのI/F仕様も標準化しました。

・T-Kernelのアプリケーション側からデバイスドライバを呼ぶには tk_opn_dev(オープンデバイス)、tk_rea_dev(リードデバイス)などのAPIを利用します。デバイスの種類によらず、このような統一されたアプリケーションインタフェースでデバイスをアクセスできます。

・インタフェース層では、複数のアプリケーションから同時に発行されたデバイスアクセスの要求に対して、排他制御を行ない、その要求をドライバの論理層へ渡します。

・論理層では、受け付けたデバイスアクセス要求に対する処理を行います。その際、必要に応じて物理層を呼び出します。論理層では、原則として機種やハードウェアに依存しない処理を行います。

・物理層では、実際のデバイスのハードウェアにアクセスします。

ドライバ開発の流れ

1. ドライバの仕様書の作成
   固有データ、属性データの仕様を決める
2. インタフェース層の選択:sdiかgdiか
3. 論理層/物理層のプログラミング
4. 動作テスト
5. 流通はバイナリのみで可(ソース不要)

・デバイスドライバ開発時の作業の流れを説明します。

・まず、ドライバの仕様を決めます。ドライバ名、固有データ（何バイト単位でアクセスするか、ランダムアクセス可能かなど）、属性データ（デバイスの設定の取得と設定変更）などに関して、具体的な仕様を定めます。

・具体的なデバイスドライバの仕様例として、T-Engineフォーラムのミドルウェア WGで策定した標準デバイスドライバの仕様書が公開されています。必要であれば参考にしてください。

・インタフェース層では、単純(シンプル)ドライバインタフェース層(sdi)か汎用(ジェネラル)ドライバインタフェース層(gdi)を選択します。
両者の選択の基準については、次のページで説明します。

・論理層と物理層のプログラミングを行います。
この時、機種非依存部である論理層の処理プログラムと機種依存部である物理層の処理プログラムをソース上でうまく整理しておくことができれば、後で他機種へ移植するのが容易になります。

・動作テストの後は、デバイスドライバのプログラムをオブジェクト(バイナリ)で流通させることができます。ソースの公開は必要ありません。

ドライバインタフェース層

- sdi (simple device driver interface)
  - リード/ライトに待ちがほとんど発生しないデバイス向け
    - 例: 液晶画面、リアルタイムクロック
- gdi (general device driver interface)
  - リード/ライトに待ちが発生するデバイス向け
    - 例: ディスク, LANなどの通信系

→ドライバ開発者が、デバイスの性質に応じて、どちらかを選択

・T-Engineのデバイスドライバには、sdi と gdi の2種類の実装方法があります。

・sdi を使うと、ドライバの構造が単純になります。

・ただし、待ちが発生するデバイスに対して sdi を使うと、その分だけシステム全体のリアルタイム性が低下します。たとえば、CF カードやハードディスクのように、ATAコマンドをコントローラに投げて、そのレスポンスを待つようなコマンドレスポンス型デバイスの場合、不定期の待ちが発生しますので、sdi は不適当です。

・一般論として、途中で待ちに入る必要がないデバイスの場合は構造の単純なsdiが適当であり、途中で待ちに入る通信系やストレージ系のデバイスの場合には、割込みやマルチタスク機能を活用できるgdiの方が適当です。

sdi と gdi の違い

- sdi

  リード/ライトは、ドライバを呼び出した
  アプリケーションのコンテキストでそのまま行う

→プログラミングは単純

- gdi

  リード/ライトは、アプリケーションとは別に、
  ドライバ内の要求処理タスクで行う

→要求をキューから取り出して独自タスク内で処理するので、
プログラミングはそれなりに複雑になる

・sdiの場合、実際のデバイスアクセス処理を行うコンテキスト(タスク)は、要求を出したアプリケーションと同じです。

・一方、gdiの場合、ドライバ内の独自のコンテキスト(タスク)でデバイスアクセス処理を行います。すなわち、実際のデバイスアクセス処理を行うコンテキスト(タスク)は、要求を出したアプリケーションとは異なります。

ドライバ内の各関数の役割(1)

論理層は次の関数に分かれる:

- main()
  ドライバの登録、要求処理タスクの生成(gdiのみ)
  などのセットアップ

- open(), close()
  ドライバをオープンする際に必要なデバイスの初期化
  ドライバをクローズする際に必要なデバイスの後始末

- event()
  サスペンド、リジューム、CF/USB の挿抜時などに呼ばれる

・main 関数では、ドライバ登録など、ドライバを利用するための準備を行います。

・オープン関数、クローズ関数では、デバイスの初期化や後始末を行います。
例えば、電源管理をしているデバイスの場合、オープンの要求があった回数を保持しておき、最初のオープン時にデバイスへの電源供給を開始し、最後のクローズ時に電源を落とすといった処理を行います。

・イベント処理関数は、アプリケーション側からのデバイスアクセス要求とは無関係に(非同期に)、システム側の状態変化があった場合に呼び出されます。
例えば、USB スロットに USB デバイスを挿入した場合に、この関数が呼ばれます。ドライバ側では、挿入された USB デバイスの種類を調べて、対応可能なデバイスであれば、そのデバイスの接続処理を行います。

ドライバ内の各関数の役割(2)

- read(), write() (sdi の場合)
  デバイスのリード/ライト処理を行う
  (ドライバを呼び出されたアプリケーション側と同じコンテキスト)

- 要求処理タスク (gdi の場合)
  要求をキューから取出し、それに応じて
  デバイスのリード/ライト処理を行う
  (独自コンテキスト)

- abort() (gdi のみ)
  要求処理タスクでリード/ライト実行中に、それを途中で
  すみやかに中断させるための処理を行う

・デバイスのアクセス処理(リード/ライト)を行う部分は、 sdi ではリード/ライト関数になっていますが、 gdi では独立したタスク内でこの処理を行います。

・gdi では、途中で処理を中断させたい時にアボート関数が呼ばれます。
アボート関数が呼ばれた場合、要求処理タスク内でのリード/ライト処理をすみやかに中断させる必要があります。

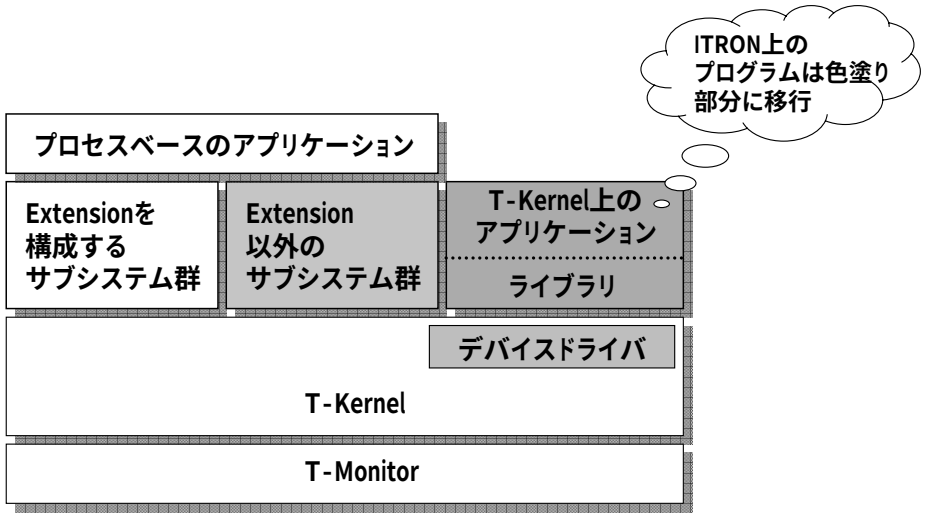
ITRONからT-Kernelへの
ソフトウェア移行方法

本章の概要および前提知識

- 本章の概要
  - ITRONからT-Kernelへの移行とは
  - TCP/IPミドルウェアの移行事例
  - まとめ

- 本章を読むために必要な前提知識
  - リアルタイムOSを使えること
  - C言語のプログラムの読み書きができること
  - 組み込みソフトウェア開発ができること
    （プログラミング⇒ビルド⇒デバッグ）

ITRONからT-Kernelの移行とは



ITRON上のプログラムをT-Kernel上に移行する場合

- デバイスドライバのT-Kernel対応
- アプリケーション（ライブラリ）のT-Kernel対応
- ミドルウェアのサブシステム化

という作業が必要となります。

移行レベル

- レベル1
  - システムコールAPIをT-KernelのAPIへ変更
- レベル2
  - レベル1＋T-Kernel仕様に基づくサブシステム化やデバイスドライバ化、T-Format対応
- レベル3
  - レベル2＋MMU利用環境でも動作するように変更

ITRON上のソフトウェアをT-Kernelへ移行する方法として3つのレベルがあります。

システムコールAPIの変更

- オブジェクトIDの割り当て
  - T-Kernelではオブジェクトは自動割付
- 静的API
  - T-Kernelでは静的API未サポート
  - コンフィギュレーション方法も未サポート
- 待ちになるシステムコール
  - T-Kernelは「タイムアウト時間指定」のAPIのみ
- T-Kernelでサポートされていないの機能
  - データキューとオーバーランハンドラは未サポート
    ⇒データキューはメールボックス機能で一部代替可能
    ⇒オーバーランハンドラはタスクのタイムスライス時間設定で対応可能

```
ER ercd = tk_chg_slt(ID tskid, RELTIM slicetime);
          tskid:タスクID
          slicetime:スライスタイム（ミリ秒）
```

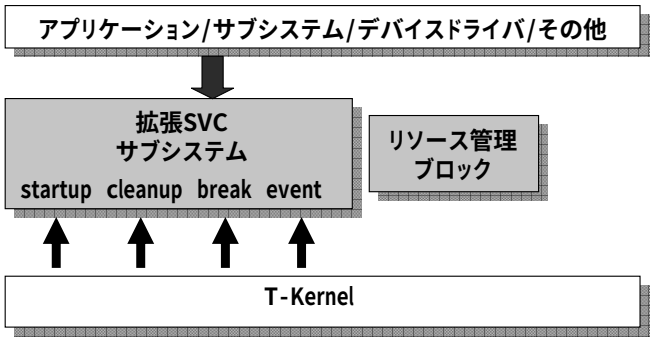

システムコールAPIの変更について

・基本的にはITRONもT-Kernelも同様のシステムコールAPIをサポートしているので機械的な変更でよいが、一部留意が必要なところもあります

・タスク生成時のパラメータ、特に論理空間IDやリソースIDなどについては注意が必要です

サブシステム化

- サブシステムとは
  - 拡張SVC、startup/cleanup/break/event処理関数、及びリソース管理ブロックで構成されるT-Kernel標準形式のミドルウェア



  - T-Kernel/OSのサブシステム管理機能を用いて動的な登録/削除が可能

```
ER ercd = tk_def_ssy(ID ssid,T_DSSY *pk_dssy);
          ssid:     サブシステムID
          pk_dssy:サブシステム定義情報
```


サブシステムとは、

・拡張SVC、ブレーク関数/スタートアップ関数/クリーンアップ関数/イベント処理関数及びリソース管理ブロックから構成されています。

・T-kernelでは拡張SVCハンドラは、単独ではなく、サブシステムの一部として定義されます。
（サブシステムは単なる拡張SVCハンドラではありません）

デバイスドライバ化

- T-Kernel/SM仕様でデバイス管理機能を規定

```
ER ercd = tk_def_dev(UB *devnm,T_DDEV *ddev, T_IDEV *idev);
          devnm:   物理デバイス名
          ddev:    デバイス登録情報
          idev:    デバイス初期情報(リターンパラメータ)
```

- インタフェース関数の記述

```
オープン関数:ER openfn(ID devid, UINT omode,VP exinf)
クローズ関数:ER closefn(ID devid, UINT option, VP exinf)
処理開始関数:ER execfn(T_DEVREQ *devreq,TMO tmout,VP exinf)
完了待ち関数:ER waitfn(T_DEVREQ *devreq, INT nreq,TMO tmout, VP exinf)
中止処理関数:ER abortfn(ID tskid, T_DEVREQ *devreq,INT nreq, VP exinf)
イベント関数:INT eventfn(INT evttype,VP evtinf, VP exinf)
```

- デバイスドライバI/F層(リファレンス)
  - 単純デバイスドライバI/F層(Simple Driver Interface)
  - 汎用デバイスドライバI/F層(General Driver Interface)
- 標準デバイスドライバ仕様の公開

T-Kernelのデバイスドライバは

・T-Kernel/SMでデバイス管理機能を規定しています。
⇒デバイス登録・デバイスオープン・デバイスクローズ・読み込み・書き込みのシステムコール
⇒インタフェース関数も仕様が規定されています

・デバイスドライバI/F層としてSDIとGDIの2つのリファレンスがあります。

・標準デバイスドライバについては仕様が会員向けに公開されています。

T-Format対応

- ミドルウェア流通を円滑にするための規定
- ベンダコード体系
  - 3文字以上8文字以内の文字列でフォーラム管理
- C言語グローバルシンボル名
  - ファイル名、関数名、グローバル変数名など規定
- バイナリコード形式
  - ELFフォーマット

T-Format仕様書はT-EngineフォーラムWebサイトにて公開

T-Formatは複数のベンダが提供するミドルウェアを組み合わせシステム構築するための規定です。

・ファイル名やグローバル変数名、関数名などの命名規約を定めています。

・命名にはフォーラム管理のベンダコードを一意に割付、機能コードなどと組み合わせます。

⇒ファイル名　　unl_mpeg2_decode.c, libunl_mpeg2.a
⇒グローバル関数名　void unl_mpeg2_decode(…);
⇒グローバル変数名　char unl_mpeg2_id[4];
⇒マクロ名　　#define UNL_MPEG2_CONST　2

T-Kernel上で動作するソフトウェアはT-Formatに従っている必要があります。

MMU利用システムへの対応

- T-Kernel/OSタスク管理機能
  - タスク固有空間の設定･参照

```
タスク固有空間参照：ER tk_get_tsp(ID tskid, T_TSKSPC *pk_tskspc)
タスク固有空間設定：ER tk_set_tsp(ID tskid, T_TSKSPC *pk_tskspc)
```

- T-Kernel/SMアドレス空間管理機能
  - ライブラリ関数

```
アドレス空間設定：ER SetTaskSpace(ID tskid)
アドレス空間チェック（読み込み）：ER ChkSpaceR(VP addr,INT len)
アドレス空間チェック（読み書き）：ER ChkSpaceRW(VP addr, INT len)
アドレス空間チェック（読み実行）：ER ChkSpaceRE(VP addr, INT len)
アドレス空間ロック（常駐化）：ER LockSpace( VP addr, INT len)
アドレス空間アンロック（常駐解除）：ER UnlockSpace(VP addr, INT len)
物理アドレス取得：INT CnvPhysicalAddr(VP vaddr,INT len,VP *paddr )
論理アドレスへのマッピング：ER MapMemory(VP paddr, INT len, UINT attr, VP *laddr)
論理空間の開放：ER UnmapMemory(VP laddr)
```

 119

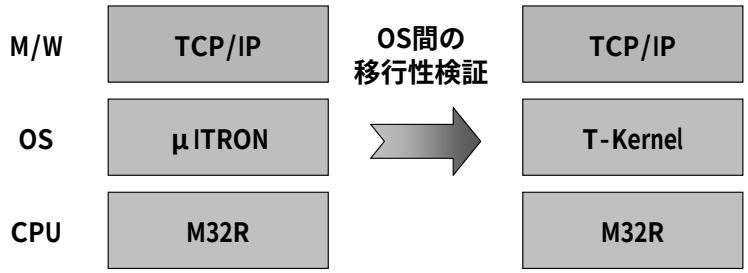
MMUを使ったシステムでも対応できるように、

･プロセス管理
･仮想記憶システム

に対応する必要があり、T-Kernelのタスク管理機能システムコールやT-Kernel/SMのアドレス空間管理機能のライブラリ関数を使用します。

移行事例

■ ITRON用TCP/IP機能をT-Kernelへ移行



本事例では、ITRON上のライブラリ形式のTCP/IP機能を
T-Kernel上のサブシステムとして実装　(移行レベル2に対応)

ITRONからT-Kernelの移行事例について、
本事例では、M32R用TCP/IPをITRONからT-Kernelへ移行する方法について説明します。

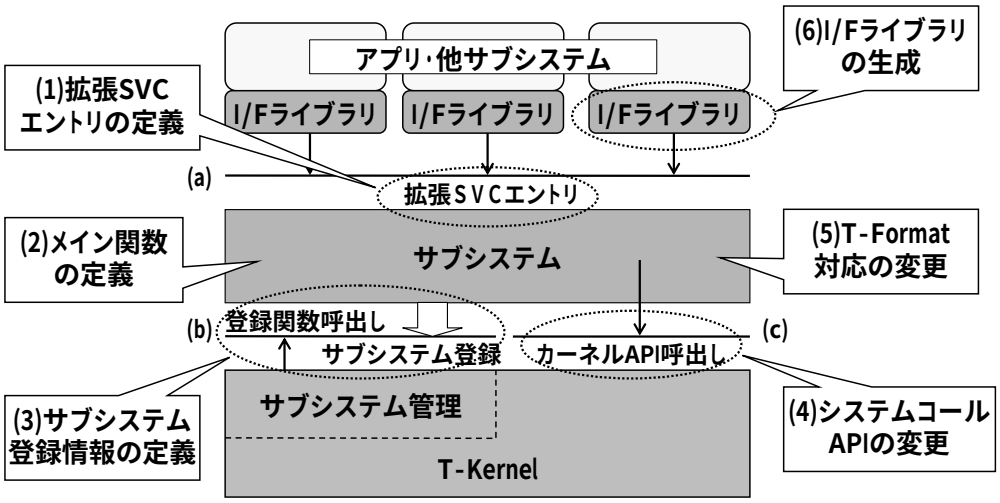
本事例では、レベル2のサブシステム化、T-Format化を行っています。

開発ターゲット

	移植元	移植先
TCP/IP機能	下記ITRON用通信ミドルウェア「M3S-TCP32R」 （BSDソケットAPIサポート）	
LANドライバ	上記TCP/IP機能に付属のLAN91C111用サンプルドライバ	
OS	ルネサス製 M32R用μITRON3.0仕様 準拠OS「M3T-MR32R」	T-Kernel
H/W	M32R版μT-Engineボード ＋SMSC製LAN91C111搭載拡張LANボード （パーソナルメディア製μT-Engine/M32104開発キット）	

開発ターゲットはM32104搭載μT-Engineにて行っています。

サブシステム化への移行項目



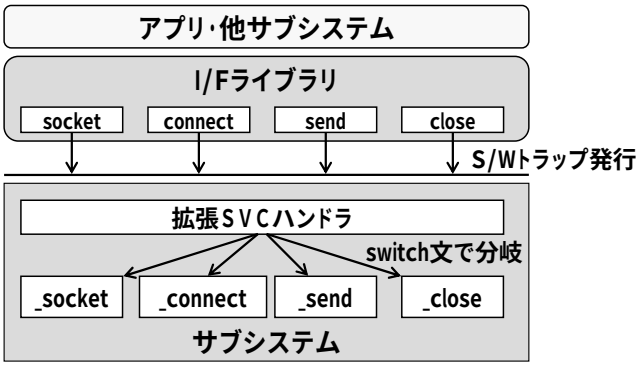
サブシステム化への移行項目を示しています。

⑴ 拡張SVCエントリの定義
⑵ メイン関数の定義
⑶ サブシステム登録情報定義
⑷ システムコールAPIの変更
⑸ T-Format対応の変更
⑹ I/Fライブラリの生成

の6項目です。

拡張SVCエントリの定義

- ソケットAPIの処理関数（旧API関数）を呼出すための拡張SVCハンドラを生成



拡張SVCエントリの定義では

アプリケーションがBSDソケットAPIを呼び出したとき、TCP/IPサブシステム側では単一の拡張SVCハンドラが起動し、APIごとに定義されている機能コードで場合分けし、各APIに対する処理関数を実行します。

メイン関数の定義

- 初期化処理とサブシステム登録処理を記述
  ⇒メイン関数は、サブシステムの起動/終了時に呼出される
  エントリポイント関数

```
EXPORT  ER  main( INT ac,  UB *av[ ] )
{
        if ( ac >= 0 ) {
                init_tcpip( );                  /* TCP/IP機能の初期化 */
                init_landrv( );                 /* LANドライバの初期化 */
                tk_def_ssy( id, &t_dssy);       /* サブシステムの登録 */
        } else {
                tk_def_ssy( id, NULL );         /* サブシステムの終了 */
        }
}
```


サブシステムのメイン関数では、初期化関数の呼出しと、サブシステム登録処理を記述します。

サブシステム登録情報の定義

- サブシステム登録時の各種情報を定義
  ⇒登録情報は以下の通り

```
t_dssy.ssyatr = TA_NULL;              /* サブシステム属性 */
t_dssy.ssypri = TCPIP_PRI;            /* サブシステム優先度 */
t_dssy.svchdr = (FP) svc_handler;     /* 拡張SVCハンドラ */

/* カーネル向け4つのサービスルーチンの登録 */
t_dssy.startupfn = NULL;              /* 使用資源の獲得処理 */
t_dssy.cleanupfn = NULL;              /* 使用資源の解放処理 */
t_dssy.eventfn = NULL;                /* イベント毎の固有処理 */
t_dssy.breakfn = (FP) break_fn;       /* タスク例外時の処理中断 */

t_dssy.resblksz = 0;                  /* リソース管理ブロック */

tk_def_ssy (TCPIP_ID, &t_dssy);       /* サブシステムの登録 */
```


T-Kernel/OSのシステムコールtk_def_ssyを用いたサブシステム登録処理では、サブシステムの属性と優先度、拡張SVCハンドラ、及び4つのサービスルーチンの登録などを定義します。

カーネル向け4つのサービスルーチンのうち、スタートアップ関数、クリーンアップ関数、及びイベント処理関数は、それぞれの関数呼出し要求のシステムコールが発行された時に実行されるものであり、サブシステム内の使用資源の獲得･解放処理やイベント毎の固有処理を記述しておきます。

また、ブレーク関数は、拡張SVCハンドラの実行中のタスクに、タスク例外が発生した場合に呼出されるものであり、拡張SVCハンドラの処理を中止するための処理を記述しておきます。

これら4つのサービスルーチンは、実行すべき処理がない場合、登録なしでもよいです。本TCP/IPサブシステムでは、コネクション接続待ちやデータ受信待ちを解除する処理を記述したブレーク関数だけを登録しています。

システムコールAPIの変更

- 本事例では、15種類のカーネルAPI呼出し部分に関して、合計36箇所の修正を実施

使用カーネルAPIの変更内容	該当するカーネルAPI数
API名のみ変更	6
API名と引数の変更	7
API引数のみ変更	1
使用APIの変更（対応するAPIの統合による）	1
合計数	15

µITRON3.0仕様からT-Kernel仕様へのAPIの変更内容一覧を表に示しています。

本事例での変更内容は、API名や引数の変更といった機械的な修正で対応可能なAPIが大半を占めています。

T-Format対応の変更

■ C言語グローバルシンボル名及び各種ファイル名に、固有のプレフィックスを追加

ミドルウェア名	＝	登録ベンダコード	＋	サブシステム機能名
renesas_tcpip		(renesas)		(tcpip)

サブシステムファイル名	librenesas_tcpip.a，renesas_tcpip.h
Exportされる関数名	renesas_tcpip_socket()　etc
M/Wパッケージ名	renesas_tcpip_10000

T-Format対応の変更では

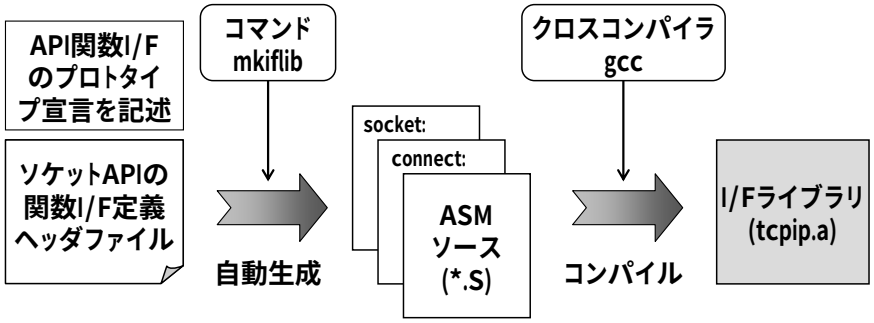
・ソースコードのスタイル形式に関するC言語グローバルシンボル名及び各種ファイル名の命名規則に従い、

それぞれにミドルウェア名「renesas_tcpip」（ルネサスのベンダコード”renesas”と本サブシステム機能名”tcpip”）を付与しています。

T-Formatに対応することで、他のミドルウェアやデバイスドライバとのシンボル衝突を回避できると共に、個々のミドルウェアやデバイスドライバを特定するのが容易になります。

I/Fライブラリの生成

- ソケットAPIの関数I/F定義の記述、及びI/Fライブラリの生成
  ⇒I/Fライブラリ生成までの流れは以下の通り



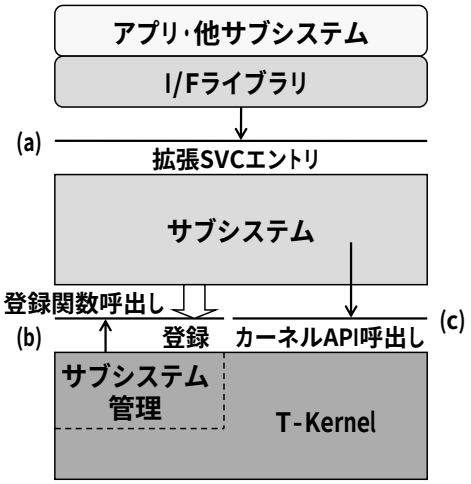
T-Engine開発キットを用いた場合、I/Fライブラリの生成は以下のステップで行います。

まず始めに、ヘッダファイルにAPI関数I/Fのプロトタイプ宣言を記述します。

次に、記述したヘッダファイルに対して、T-Engine開発キットに含まれるmkiflibコマンド（スクリプト）を実行することにより、I/Fライブラリのアセンブラコードが自動生成されます。

最後に、コンパイルを実行し、I/Fライブラリのバイナリモジュールを生成します。

移行性の評価



	モジュール間I/F毎の移行性評価
(a)	拡張SVCエントリの関数I/F定義を記述するだけでASMコードが自動生成されるツールが利用可能なため作業は容易
(b)	M/W流通性確保の枠組みとして、カーネル向け4つのサービスルーチン登録が規定されているが、その処理内容はM/Wに応じて自由に定義可能であり（未定義でもよい）、対応は容易
(c)	使用カーネルAPIの変更は呼出し箇所が明確なら、機械的に置き変え可能であり（親和性は高い）、移行は容易

ここでは、μITRONからT-Kernelへのソフトウェア移行性の評価について示します。

具体的には、サブシステムの外部インタフェースの作りやすさについて、今回の移行作業を通じて得られた知見を簡単にまとめています。

今回の移行作業では、ミドルウェア内部の構成やアルゴリズムは一切変更せずに、外部インタフェース部分を作り込むだけでサブシステム化を実現しています。

まとめ

- ITRON⇒T-Kernelへの移行には3レベルあり、どのレベルまで対応するかは選択可能
- サブシステム化やデバイスドライバ化することで、ミドルウェアの流通性/再利用性の向上が可能
- サブシステム化する場合は6つの移行項目に対応
- ITRONとT-Kernelは親和性が高く、移行は比較的容易（慣れれば、さらに期間短縮が可能）

T-Dist入門

本章の概要および前提知識

- 本章の概要
  - T-Distとは
  - T-Distの仕組み
  - T-Distの利用例
- 本章を読むために必要な前提知識
  - T-Engine/T-Kernelに関する基本的な知識

ミドルウェア流通に向けた施策

- カーネルのシングルソースコード化
  - 強い標準化により、T-Engineが目的とする「ミドルウェア流通のためのプラットフォーム」を実現する
- T-Format
  - T-Engine、T-Kernel上で動作するミドルウェアやアプリケーションソフトウェアのコード形式を規定する
- 標準デバイスドライバ仕様
  - 標準的なデバイスドライバ仕様の規定
- T-Dist
  - ミドルウェア流通プラットフォーム

ミドルウェア流通に向けて、T-Engine Forumでは様々な施策が実施されています。

・カーネルのシングルソースコード化

・この強い標準化により、T-Engineが目的とする「ミドルウェア流通のためのプラットフォーム」を実現することが可能となります。

・ T-Formatの導入

・T-Engine,T-Kernel上で動作するミドルウェアやアプリケーションソフトウェアのコード形式を規定しています。

・標準デバイスドライバ仕様の策定

・標準的なデバイスドライバ仕様を策定することにより、上位アプリケーションへのインタフェースを標準化することが可能となります。

T-Distとは

- T-Engineプロジェクトの大きな目的のひとつがソフトウェア・ミドルウェアの流通
  - T-Engine/T-Kernel上のソフトウェア流通を強力にサポートするプラットフォーム
    - 組み込みシステムにおける開発成果の再利用性を高め、開発コストの低減と開発期間の短縮を目指す
  - 現在、T-Engine Forum ミドルウェア流通WGにてソフトウェアの流通実験が実施中

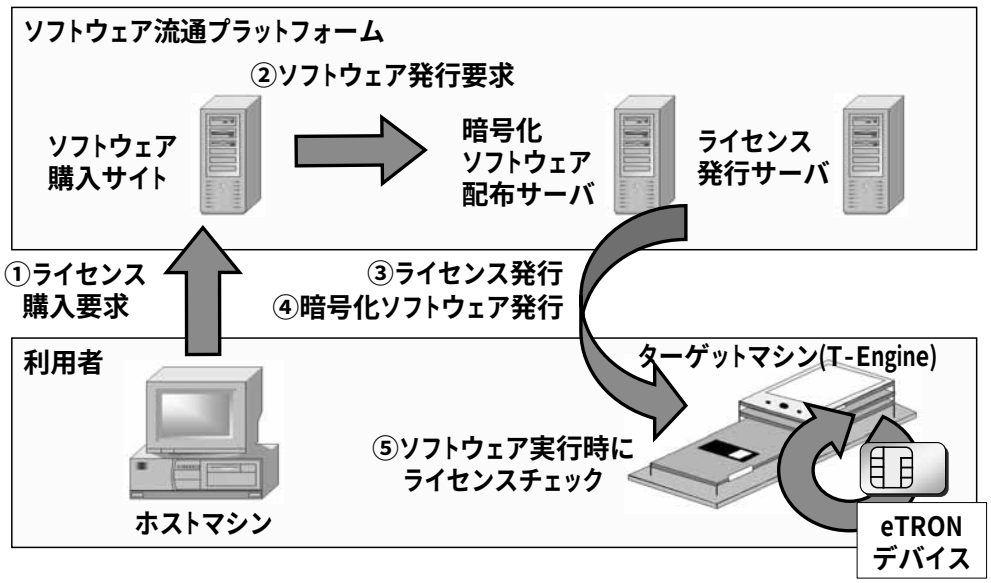
・T-Distとは

T-Engine上のソフトウェア流通を強力にサポートするプラットフォームです。

組み込みシステムにおける開発成果の再利用性を高め、開発コストの低減と期間短縮を行い、T-Engineおよび、T-Engineソフトウェアを普及させていくことがT-Distの目的となっております。

現在、T-Distは、T-Engineフォーラムのミドルウェア流通ワーキンググループにて流通実験が実施されております。

T-Distの仕組み



①ライセンス購入要求

購入要求の流れは既存の販売方式と基本的に変わらないため、新たなスキームを構築する必要はありません。

②ソフトウェア発行要求

通常であれば、ここでソフトウェアを納品して終わりですが、T-Distの場合は、購入要求を受け付けた販売サイトは、実行形式のソフトウェア、付与するライセンス情報、利用者の情報等をセットでT-Dist配布機構に渡します。

③ライセンス発行

利用者からのライセンスダウンロード要求があると、サーバは利用者のeTRONチップ内にサーバから直接セキュアなプロトコルでライセンスを書き込みます。

このプロトコルは先ほどご説明いたしましたeTPと呼ばれるもので、価値情報をセキュアに通信することが可能なプロトコルです。

④ソフトウェア発行

サーバにて生成された暗号化ソフトウェアについても利用者に対して配布します。

⑤ソフトウェア実行時にライセンスチェック

利用者がソフトウェアをロードしようとすると、そのソフトウェアローダ(OS)が、eTRONチップと通信を行ってライセンスを確認します。

T-Dist実現のための基盤技術

- T-Engine / T-Kernel
  - ハードウェアプラットフォームの違いを吸収し、ソフトウェアの有効利用が可能な共通プラットフォーム
- eTRONデバイス
  - 「暗号化されたデータを復号するとともに課金する」という処理を確実に内部で行なうチップデバイス
- eTP (entity Transfer Protocol)
  - セキュアなデータをeTRONデバイス内に安全に価値情報を配送するプロトコル

eTRONデバイス

- 特徴
  - 16bit マイクロプロセッサ
  - 32.5 KB の EEPROM
  - 96 KB の ROM.
  - 4KB の RAM
  - コプロセッサ内蔵
    - 1024 ビットの 剰余演算の補助を行う
    - 512 バイトRAM を内蔵
    - RSA 暗号などの計算を補助する機能を提供
  - セキュリティ
    - 内部データの拡散保持、消費電力解析攻撃対策などのセキュリティ機能
  - T-Engine Forum 認定チップ

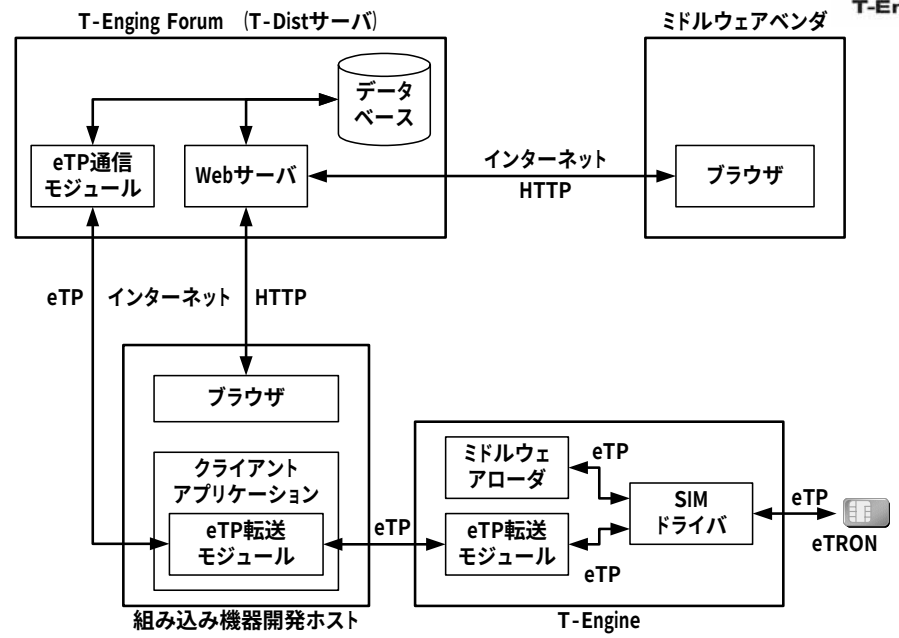
・eTRONデバイス

eTRON（entity TRON）は、コンピュータ化された社会生活において中核となる「情報」を安全に、格納またはデジタル情報基盤上で流通させることを目的とした、分散広域システムアーキテクチャです。

また、eTRONデバイスは、T-Engineといった各種コンピュータノードに組み込んで利用することを想定して設計され、価値情報を扱う多様なアプリケーションを支援するための高機能命令を備えています。

実装の一つとして、T-Distで利用されているSIM形状のものがあります。

システム構成



T-Dist提供機能

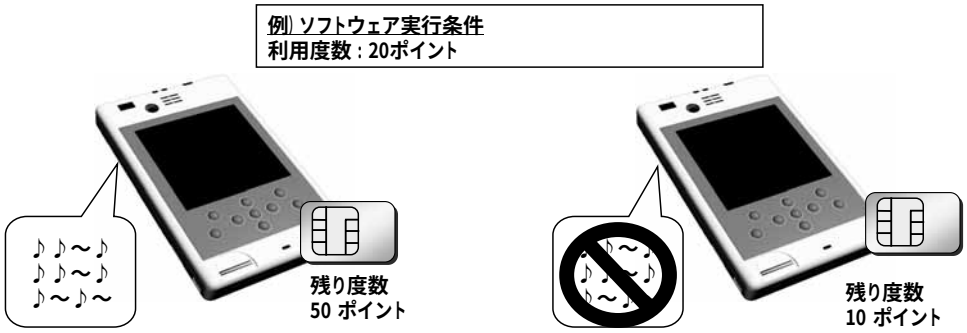
- ローダ提供機能
  - 署名検証機能
  - eTRON IDチェック機能
  - 課金（ライセンス確認）機能
  - ソフトウェア復号機能
  - ソフトウェア実行機能
- サーバ提供機能
  - ライセンス（鍵）発行機能
  - 暗号化ソフトウェア生成機能
  - ライセンス発行機能
  - ソフトウェア登録・削除・検索機能

T-Dist利用のメリット

- ミドルウェアベンダのメリット
  - T-Distを販売チャネルのひとつとして活用できる
    - T-Dist上でソフトウェア試用版を公開し、試用回数を超えて継続使用する場合には、ベンダから購入
  - ソフトに不正利用対策はT-Distにより提供
    - ソフトウェアは、eTRONを利用したライセンスチェックをパスしないかぎり実行できない
    - ベンダはソフトウェアに手を加えて、実行制御処理を入れる必要が無い
- 利用者のメリット
  - T-Engine上で動作するソフトウェアを簡単に入手･実行できる
    - ソフトウェア利用の敷居が低くなる
    - ソフトウェアの比較検討が容易になる

T-Dist利用例

- プリペイド方式
  - 利用度数を発行し、一度の利用につき度数を減算する方式
  - ソフトウェアの利用代金を前払いし、利用度数をeTRONチップに蓄積しておく
  - 度数が足りなくなると、ソフトウェアは利用できなくなる



・T-Distによるソフトウェア実行制御

プリペイド方式による実行制御の例です。

あらかじめ、利用者は、ソフトウェアの利用代金を前払いし、利用度数をeTRONチップに蓄積しておきます。

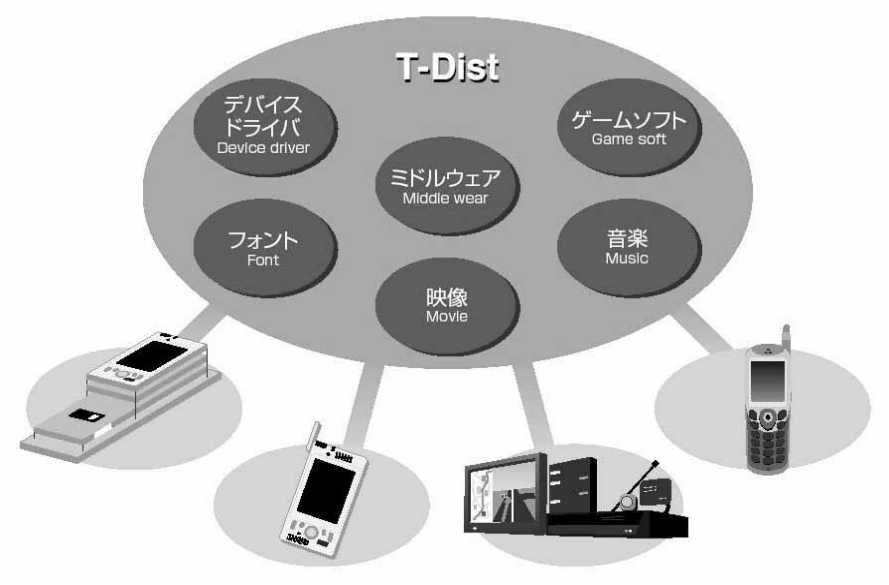
ソフトウェア一度の利用につき度数を減算する方式となっております。eTRONチップ内の度数が足りなくなると、ソフトウェアは利用できなくなります。

この方式を応用いたしますと、試用回数を超えて継続使用する場合には、ベンダから購入するような、いわゆるTRY AND BUY方式が可能となります。

T-Distの将来

- ミドルウェアの流通を促進するプラットフォームとして、ソフトウェアの情報を広く公開/配布
- ソフトウェアの流通を強力にサポートし、T-Engineを利用した組み込みシステムの開発効率の大幅向上を目指す
- コンシューマ向けソフトウェアや音楽、映画、映像といったコンテンツについても配信し、コンテンツ流通ビジネスの新たなプラットフォームとして利用されるよう進めていく

T-Distの将来



付録

T-Kernel 利用によるソフトウェア開発方法
- ユビネットパスAD-L開発事例 -

本章の概要および前提知識

- 本章の概要
  - T-Kernelを利用したソフトウェア開発の事例紹介
  - 大日本印刷株式会社（以下、DNP）が開発した液晶ディスプレイ付きICカードR/W「ユビネットパスAD-L」について
- 本章を読むために必要な前提知識
  - ICカード関連知識
    - ◆接触/非接触ICカード
    - ◆ICカード リーダライタ
  - T-Kernel基礎知識

用語説明

- UIM (User Identity Module)
  - 携帯電話会社が発行する、契約者情報を記録した小型のICカード。携帯電話機などに差し込んで、利用者の識別に用いる。
  - 幅25㎜、高さ15㎜で同サイズのICカード全般をさすこともある。
- ISO/IEC 7816
  - ICカードの接触通信について規定している国際規格
- ISO/IEC 14443
  - ICカードの非接触通信について規定している国際規格

ユビネットパスシリーズ

■ UIM（小型のICカード）用USBスティックタイプR/W

接触型（ISO/IEC 7816）

ハイブリッド型
（ISO/IEC 7816/14443）

用途：ネットワークセキュリティ

指紋センサ付き

用途：バイオメトリクス本人認証

・ユビネットパスは小型のICカードであるUIM用のUSBスティックタイプリーダライタです。UIMを挿入し、ユビネットパスごとパソコンなどの端末のUSBポートに挿して使っていただけます。

・接触型はパソコンのログイン管理などネットワークセキュリティ用途に適しています。

・非接触型はアンテナを内蔵しておりドアゲートの開閉などに便利です。

・指紋センサ付きは暗証番号のかわりに指紋照合により本人認証を行うことができます。

ユビネットパスAD-L

- 特徴
  - モノクロ液晶ディスプレイ
  - ISO/IEC 14443 アンテナ（TYPE-A,B,eTRON）
  - USBコネクタ（mini-B）
  - 2次電池内蔵USBバスパワー充電
  - T-Kernelを搭載、T-Engine Appliance第一号
- 利用方法
  - 電子マネーの残高や電子チケットの内容を確認
  - 接触（ISO/IEC 7816）、非接触（ISO/IEC 14443）のデュアルI/Fに対応、場面に応じて使い分け

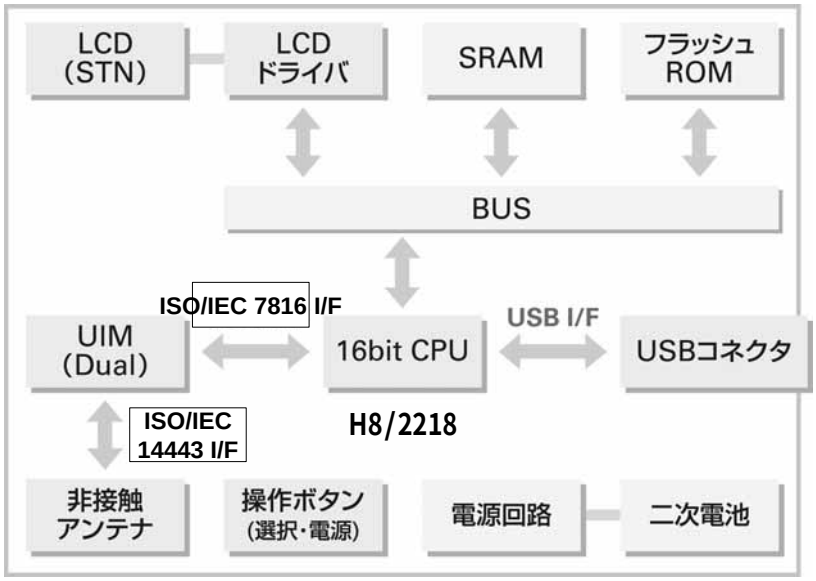
・「ユビネットパスAD-L」は液晶を搭載したモデルです。

・特徴は上記の通りです。

・液晶ディスプレイとバッテリを搭載しているため、ボタン操作で簡単に電子マネーの残高や電子チケットの内容を表示して確認することができます。

・オフィスや自宅のPCと接続するときはUSB mini-Bで、店舗のレジや交通機関の改札では内蔵のアンテナで非接触で使用可能です。

ユビネットパスAD-L　ブロック図

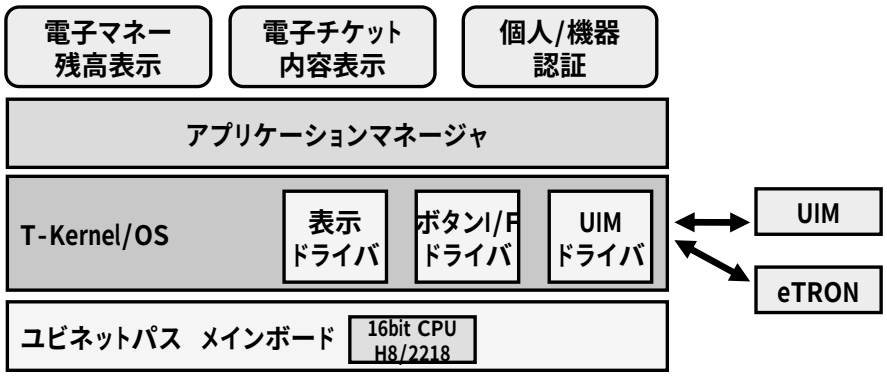


ユビネットパスAD-Lのブロック図です。

USB I/Fで受信したICカードへのコマンドはCPUでISO7816というプロトコルに変換され、接触I/Fを通してICカードに渡されます。

非接触通信の場合はアンテナが直接UIMに接続されており、CPUは関与しません。従ってUIMはeTRON/16Dualのようなデュアルl/F対応のものが必要です。

ユビネットパス　ソフトウェア構成図



ソフトウェアの構成図です。

T-Kernelの上にアプリケーションマネージャと呼ぶ管理層を設け、12種類までのアプリケーションを搭載、管理できるようになっています。起動直後のメインメニュー表示なども行います。

T-Kernel搭載の経緯

- OS搭載の理由
  - 従来はUSB⇔ICカードプロトコル変換ロジックのみ搭載。
  - 今後、高機能CPU搭載やI/F追加等シリーズ展開を想定。その際デバイスドライバの追加で対応するなど、トータルでのソフトウェア開発工数低減を狙いOSを搭載。
- T-Kernelの選択理由
  - 仕様がオープン。ソースコードも公開。
  - T-Licenseにより最終製品搭載時の最適化が可能。

ユビネットパスは市場にあわせI/Fやデバイスを追加したり異なるCPUを選択するなどカスタム化やシリーズ展開を想定しています。そうした部品が追加された際にデバイスドライバの追加で対応するなど、今後の開発を含めトータルでのソフトウェア開発工数低減を狙いOSを搭載しました。

T-Kernelは仕様がオープンであり、ソースコードレベルで公開されています。またT-Licenseにより最終製品搭載時の最適化が可能なため、本製品には適していました。

T-Kernel実装範囲

- T-Kernel(32bit)をH8/2218(16bit)に移植。
  システム依存部をH8用にC言語、アセンブラで作成。
- 実装範囲
  - 最終製品に必要なT-Kernel/OSとT-Monitorの一部。
    - T-Kernel/OS ：タスク管理、同期･通信、メモリ管理、
      例外割込み制御、時間管理、サブシステム管理
    - T-Kernel/SM：システムメモリ管理、アドレス空間管理、
      割込管理、省電力管理、システム構成情報管理、
      I/Oポートアクセスサポート
    - T-Kernel/DS：カーネル内部状態参照、実行トレース
    - T-Kernel/
      Monitor ：システム機能、デバッグ機能、
      プログラムサポート

T-Kernelのソースコードを移植しましたが、OSの外販は行う予定がなかったため、製品に必要なT-Kernelのコア部分のみを実装しています。

未実装部の例

- MMU関連
  - T-Kernel/OSのMMU関連。CPUに機能がない為。
- デバッグ関連
  - T-Monitor：コンソールI/O（製品化時に不要な為）
  - T-Kernel/DS：OSの外販はせず、製品に不要な為。
- その他
  - 例：省電力機能（⇒1分後に電源OFFの仕様の為）

⇒最終製品のスペックに合わせてサブセット化した。

このように、最終製品のスペックに合わせたサブセット化を行ないました。

開発環境

- OS移植作業、アプリケーション開発ともに
  ターゲットチップの純正ツール（Renesas製）を使用、
  エミュレータ、検証機にて開発・デバッグ。
  - OS：Windows2000（SP4）
  - コンパイラ、アセンブラ：HEW3 ver3.0.02(Release5)
  - 書込みツール：1)FLASH Development Toolkit ver 3.0
    2)自社製ダウンロードツール
  - エミュレータ：HITACHI MY-ICE　H8S/2218用

いわゆるGNU開発環境ではなくターゲットチップのベンダの純正ツールとエミュレータにて開発しました。

アプリケーションのダウンロードは新規にツールを開発しています。

修正内容

- システム依存部分
  - レジスタ数の不足（半減）による調整
  - トラップ命令の制限（4つまで）による調整
- 基本部分（システム非依存部）
  - CPUの違いからくる修正（32bit⇒16bit）
    - int型をlong型に（typedef.h 修正）
    - long long型をdouble型に（システム時間に制限）
  - コンパイラの違いによる修正（標準オプションの場合）
    - ターゲットチップのコンパイラとGNUコンパイラの違い
      例：ctask.task = log_task;　⇒ctask.task=(VP) log_task;
  - その他、ifdef部分、デバッグ関数サポート削除

ターゲットCPUが32bitから16bitになったこともありリソースが減少し、レジスタを使いまわすなどの調整を行いました。

基本部分はほぼ再利用できましたが、16bit CPU向けにint型などを調整しました。

またコンパイラの違いによりキャストのチェックが厳しい部分があり、修正が必要でした（GNUでもオプションの指定により異なると思います）。

まとめ

- GNU環境以外で移植･開発を実施
- 16bit CPUへの移植も可能
- 単なるリコンパイルではないCPU依存部の移植やサブセット化にはある程度RTOSへの知見が必要
  - 本事例ではμITRON経験者が担当
- 今後はT-Kernelの適応化が予定されており、16bit CPUを使った製品にも搭載しやすくなる

参考URL・参考文献

参考URL

- T-Engineフォーラム
  - http://www.t-engine.org/
- トロンプロジェクト
  - http://www.tron.org/
- 組み込みネット
  - http://www.kumikomi.net/

参考文献

- 組み込みシステム開発のためのエンベデッド技術
  - 電波新聞社
- T-Kernel標準ガイドブック
  - パーソナルメディア

参考文献

- **ISO/IEC 7816-3:1997**
  Information technology — Identification cards — Integrated circuit(s) cards with contacts —
  Part 3: Electronic signals and transmission protocols
  (情報技術—識別カード—外部端子付ICカード—第3部：電気信号及び転送プロトコル)

- **ISO/IEC 7816-3:1997/Amd 1:2002**
  Electrical characteristics and class indication for integrated circuit(s) cards operating at 5 V, 3 V and 1,8 V
  (5V、3V及び1.8Vで動作する識別カードの電気的特性及びクラス分類)

- **ISO/IEC 7816-4:2005**
  Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange
  (識別カード—外部端子付ICカード—第4部：構成、セキュリティ及び交換のためのコマンド)

- **ISO/IEC 14443-1:2000**
  Identification cards — Contactless integrated circuit(s) cards — Proximity cards — Part 1: Physical characteristics
  (識別カード—外部端子なしICカード -プロキシミティカード- 第1部：物理特性)

- **ISO/IEC 14443-2:2001**
  Identification cards — Contactless integrated circuit(s) cards — Proximity cards —
  Part 2: Radio frequency power and signal interface
  (識別カード—外部端子なしICカード -プロキシミティカード- 第2部：高周波出力及び信号インタフェース)

- **ISO/IEC 14443-3:2001**
  Identification cards — Contactless integrated circuit(s) cards — Proximity cards —
  Part 3: Initialization and anticollision
  (識別カード—外部端子なしICカード -プロキシミティカード- 第3部：初期設定及び衝突防止)

用語集

用語	欧文表記	意味
CLI	Command Line Interpreter	パーソナルメディア株式会社より発売されている「T-Engine開発キット」に付属する開発用ツール。T-Engine上のアプリケーションのひとつであり、下記機能を有する。 ・コマンドによるファイルを中心とした各種操作 ・システムプログラム（サブシステム）のロード／アンロード ・アプリケーションプログラムの実行 ・コマンドファイルの実行
Cygwin	Cygwin	GNUの開発ツールを含む、UNIXの様々なフリーソフトウェアをWindowsに移植したもの。GPLに従ったオープンソースのフリーソフトウェア。
eTP	entity Transfer Protocol	eTRONを利用した価値情報を暗号認証通信により安全に通信するプロトコル。
eTRON	entity and economy TRON	汎用的な電子的な「実体：entity」を「経済：economy」のために実現する、「どこでもコンピュータ」社会のセキュリティ基盤のための、基本パーツからインフラまでのトータルアーキテクチャ。
FPGA	Field Programmable Logic Device	ユーザが自由にプログラミングすることができるLSI。
GNU	GNU is Not Unix.	FSF(Free Software Foundation)が進めているUNIX互換ソフトウェア群の開発プロジェクトの総称。 http://www.gnu.org/
GPL	General Public License	フリーのソフトウェアライセンスであり、コピーレフトを主張するライセンス。 http://www.gnu.org/licenses/
gterm	gterm	T-Engine開発環境に付属するターミナルソフト。
ICE	In Circuit Emulator	マイコンを開発する際に使うデバッガ。
MMU	Memory Management Unit	モジュール間のメモリ保護が可能になるメモリ管理ユニット。
RS-232C	Recommended Standard 232 version C	米国電子工業会により標準化されたシリアル通信規格。開発ホストのPCとターゲットであるT-Engine間をシリアル通信で接続する際に利用する。
T-Dist	T-Distribution	YRPユビキタス・ネットワーキング研究所が研究を行っている、T-Engine/T-Kernel上のミドルウェア流通プラットフォーム。
T-Engine ボード	T-Engine Board	T-Engineフォーラムにより標準化されているオープンな組み込み機器開発プラットフォームハードウェア。
T-Engineフォーラム	T-Engine Forum	組み込み機器向けのリアルタイムOS「μITRON」をベースにした開発プラットフォーム「T-Engine」に関連する技術の研究開発、標準化、普及に取り組む業界団体。

用語	欧文表記	意味
T-Engineフォーラム ミドルウェア流通WG	T-Engine Forum Middleware WG	T-Kernel上で動作するミドルウェア流通をテーマとしたワーキンググループ。 ソフトウェア関連企業が中心。
T-Format	T-Format	アプリケーションを複数のライブラリとリンクする際に外部シンボルの二重定義などを回避するための決まり。
T-Kernel	T-Kernel	T-Engineの標準リアルタイムカーネル。μITRONと似た機能やAPIを持つが、ミドルウェアの流通性を高める機能が追加されている。ソースが公開されており、利用条件(T-License)に同意すれば、誰でも無償で利用できる。
T-Kernel Extension	T-Kernel Extension	T-Kernelで共通に利用される、基本的・汎用的なミドルウェア。プロセス管理や仮想記憶、ハイパーテキスト型のファイル管理などを提供するStandard Extensionのほか、Tiny Extension、Enterprise Extensionなどがある。
T-Kernel/SM	T-Kernel System Manager	システムメモリ管理機能、アドレス空間管理機能、デバイス管理機能、割り込み管理機能、I/Oポートアクセスサポート機能、省電力機能、システム構成情報管理機能といった機能を有する。
T-Monitor	T-Monitor	T-KernelなどのOSが動作しない環境において対話的にメモリ参照やI/Oアクセスを行ったり、ハードウェアの初期化や自己診断、OSが起動するまでのブート処理などを行うプログラム。PCのBIOSに相当。
TRON	The Realtime Operating system Nucleus	理想的なコンピュータアーキテクチャの構築を目指して、1984年に東京大学の坂村氏が始めたプロジェクト。
T-Shell	T-Shell	パーソナルメディア株式会社より発売されている多漢字GUIシステムを実現するT-Engine用ミドルウェア。
UC	Ubiquitous Communicator	YRPユビキタス・ネットワーキング研究所が開発した、PDA型のユビキタスコンピューティング環境とのコミュニケーションツール。
カーネル	Kernel	OSの基本機能を実装したソフトウェア。
クロス開発環境	Cross-Development Environment	プログラムのコンパイルを行うマシン（ホスト）とコンパイルされたプログラムを実行するマシン（ターゲット）が異なる開発環境。
サブシステム	subsystem	T-Kernel Standard Extensionをはじめとする、システムの拡張機能を提供するプログラム。ミドルウェアをサブシステムとして実現することにより、システムにダイナミックにロード・実行したり、削除したりすることが可能となる。

用語	欧文表記	意味
デバイスドライバ	device driver	周辺機器を動作させるためのソフトウェア。OSがデバイスを制御するための橋渡しを行う。
ミドルウェア	Middleware	OS上で動作し、アプリケーションソフトウェアに対してOSよりも高度で具体的な機能を提供するソフトウェア。OSとアプリケーションソフトウェアの間の中間的な性格を持っている。ソフトウェアとほぼ同義。

ミドルウェア開発者のための T-Engine 入門　執筆協力者（順不同・敬称略）

松為　彰	パーソナルメディア株式会社
中村　大真	パーソナルメディア株式会社
青山　典生	東芝情報システム株式会社
齋藤　芳弘	株式会社ルネサスソリューションズ
福井　昭也	株式会社ルネサスソリューションズ
遠藤　幸典	三菱電機株式会社
花田　一郎	大日本印刷株式会社
萩庭　崇	大日本印刷株式会社
斎藤　博夫	大日本印刷株式会社
水本　米喜	東芝ソリューション株式会社
越塚　登	YRP ユビキタス・ネットワーキング研究所
高木　悟	YRP ユビキタス・ネットワーキング研究所
田口　剛生	YRP ユビキタス・ネットワーキング研究所
田丸　厚司	YRP ユビキタス・ネットワーキング研究所
河合　泰浩	沖電気工業株式会社
田中　卓弥	株式会社 KDDI 研究所
村松　茂樹	株式会社 KDDI 研究所

T-Engine フォーラム ミドルウェア流通 WG

ミドルウェア開発者のための T-Engine 入門

2005 年 9 月 13 日 発行

発行所

T-Engine フォーラム

〒141-0031 東京都品川区西五反田 2-20-1 第 28 興和ビル

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

TEL : 03-5437-0572 (窓口) FAX : 03-5437-2399 (窓口)