実習：組込みリアルタイム・プログラミング
（ITRON 中級編）

プログラミング演習テキスト

トロンフォーラム 学術・教育 WG 委員

株式会社グレープシステム　宮下 光明

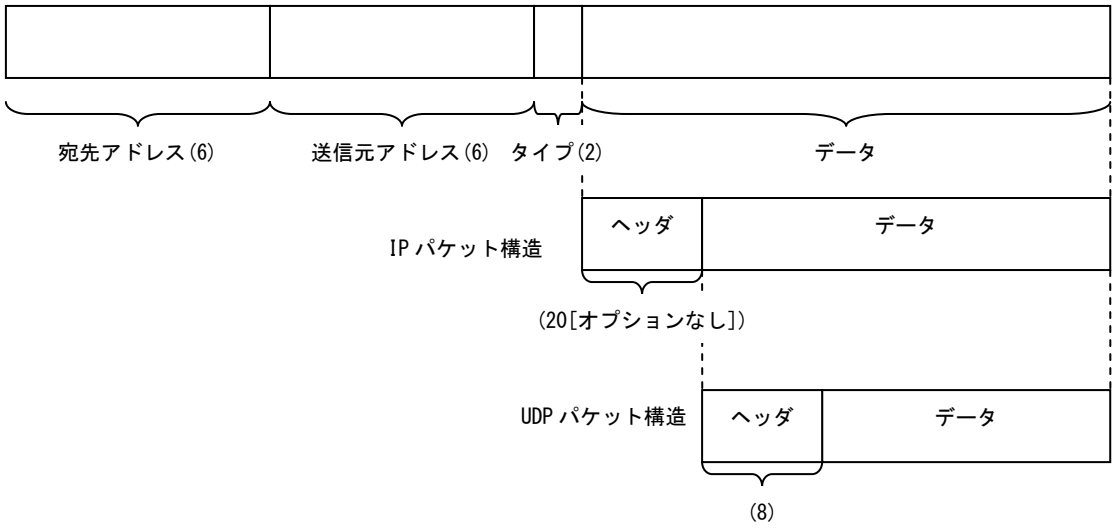
目次

1　課題：UDP/IP 通信処理

1.1　仕様

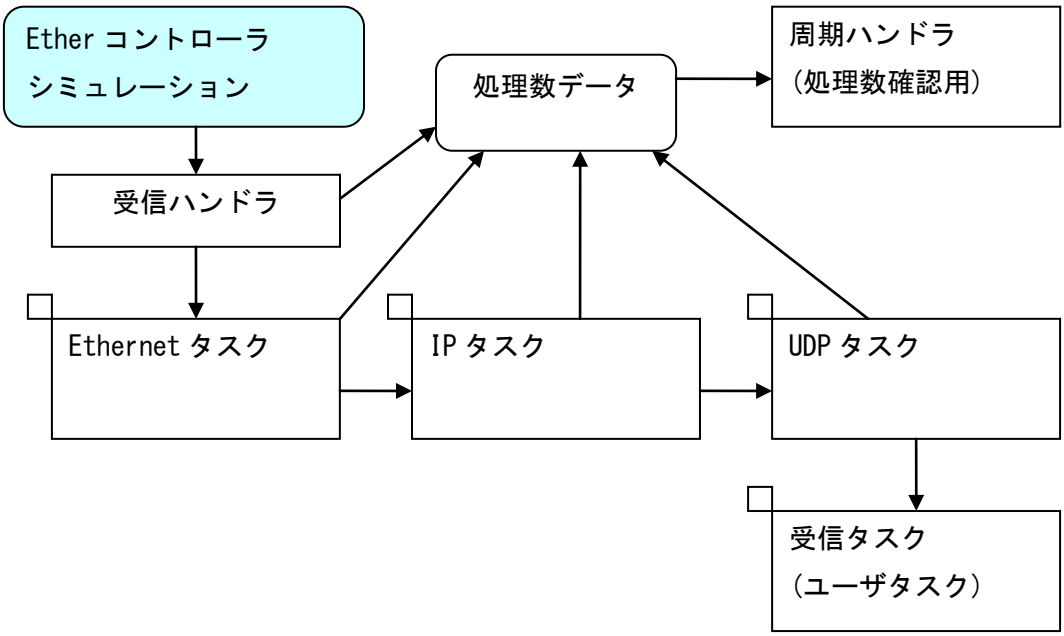
- Ethernet パケット(MAC パケット)を受信し、IP および UDP の確認を行い、受信データを受信タスクに渡す
- Ethernet パケットの構造および IP パケットと UDP パケットの概略構造を以下に示す

Ethernet パケット構造



- 動作構造のシミュレーションを行うため、エラーチェック、IP アドレスのチェック、IP フラグメントパケットの処理、ポート番号のチェックなどは省略している
  - IP およびポート番号のチェックを省略しているため、受信したすべての UDP データを、受信タスクが受け取る構造になっている
- 動作開始後、自動的にパケットを受信する形式になっている
- パケットが処理された数を、画面に表示している

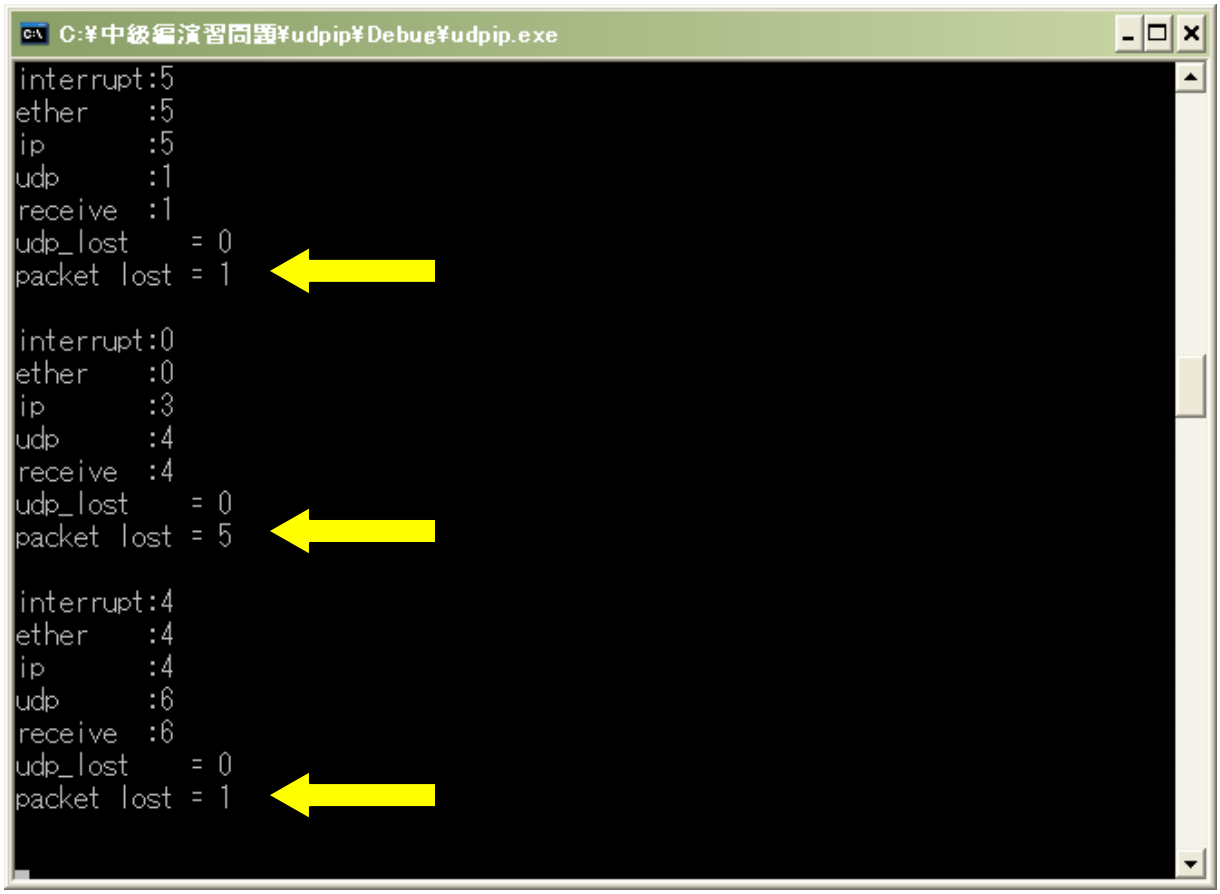
1.2　初期プログラムのタスク構成



1.3　初期プログラムの問題点と実習内容

初期プログラムでは、パケット処理構造に合わせてタスク分割を行った。処理を実行すると、パケットの処理が間に合わず、かなりの頻度でデータを取りこぼす。

課題）
通信処理関連のタスクを統合することでタスク切替えのオーバヘッドを少なくし、できる限りデータを取りこぼさないように処理を修正する。
なお、すでに初期プログラムに入っているエラーチェック（チェックサムチェック）は、処理を省略しないものとする。また受信タスクと通信を行うタスクとのインターフェースおよび各タスクの優先度は、変更しないものとする。


C:¥中級編演習問題¥udpip¥Debug¥udpip.exe
interrupt:5
ether :5
ip :5
udp :1
receive :1
udp_lost = 0
packet lost = 1
interrupt:0
ether :0
ip :3
udp :4
receive :4
udp_lost = 0
packet lost = 5
interrupt:4
ether :4
ip :4
udp :6
receive :6
udp_lost = 0
packet lost = 1

1.4　シミュレーション用サポート関数(ネットワーク関係)

ER eth_opn_por(INT portno, T_OINFO* info, VP_INT exinf);
機能：指定物理ポートをオープンする。本関数を実行すると、パケット受信割込みを受付ける状態となる。

ER eth_cls_por(INT portno);
機能：指定物理ポートをクローズする。

ER_UINT eth_rcv_frm(INT portno, VP* p_frame);
機能：指定物理ポートから、受信したパケットフレームのアドレスを取得する。本関数が呼出された後、解放されるまでに次のパケットを受信した場合には、パケットを取りこぼす。

ER_UINT eth_rel_rbf(INT portno, VP frame);
機能：受信したパケットフレームを解放する。

1.5　シミュレーション用サポート関数(その他)

unsigned short __checksum_ip(unsigned short *buf, int size, unsigned long sum);
機能：IP パケットのチェックサムを計算する。

unsigned short __checksum_udp(unsigned short *buf, int size, unsigned long sum);
機能：UDP パケットのチェックサムを計算する。なお本関数はダミーである。

int __lost_count(void);
機能：eth_rcv_frm が呼出されたあと、eth_rel_rbf が呼出される前に次のパケットを受信した数を取得する。本関数が呼出される毎に、内部で保持している数はクリアされる。

1.6　初期プログラムのリスト

1.6.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/

INCLUDE("udpip.h");

#define STACK_SIZE 2048

DEF_INH(INT_ETHER, {0, ether_handler});

CRE_TSK(ETHER_TSK, {TA_HLNG|TA_ACT, ETHER_TSK, ether_tsk, 20, STACK_SIZE, NULL});
CRE_TSK(IP_TSK, {TA_HLNG|TA_ACT, IP_TSK, ip_tsk, 20, STACK_SIZE, NULL});
CRE_TSK(UDP_TSK, {TA_HLNG|TA_ACT, UDP_TSK, udp_tsk, 20, STACK_SIZE, NULL});
CRE_TSK(RCV_TSK, {TA_HLNG|TA_ACT, RCV_TSK, rcv_tsk, 10, STACK_SIZE, NULL});


CRE_SEM(ETHER_SEM, {TA_TFIFO, 0, 100});
CRE_MPF(PACKET_MPF, {TA_TFIFO, 10, 1520, NULL});

CRE_DTQ(IP_DTQ, {TA_TFIFO, 5, NULL});
CRE_DTQ(UDP_DTQ, {TA_TFIFO, 5, NULL});
CRE_DTQ(RECEIVE_DTQ, {TA_TFIFO, 5, NULL});


CRE_CYC(CHECK_CYC, {TA_HLNG|TA_STA, CHECK_CYC, check_cyc, 1000, 1000});

/* User initialize function */
ATT_INI({TA_HLNG, 0, init_demo});
```

1.6.2 udpip.h

```
#define INT_ETHER   1

// ダミー定義
#define T_OINFO int

// シミュレーション関数
ER eth_opn_por(INT portno, T_OINFO* info, VP_INT exinf);
ER eth_cls_por(INT portno);
ER_UINT eth_rcv_frm(INT portno, VP* p_frame);
ER_UINT eth_rel_rbf(INT portno, VP frame);
ER eth_ena_cbr(INT portno, UINT flag);
ER eth_dis_cbr(INT portno, UINT flag);

unsigned short __checksum_ip(unsigned short *buf, int size, unsigned long sum);
unsigned short __checksum_udp(unsigned short *buf, int size, unsigned long sum);
int __lost_count(void);
```

```
// ユーザー関数
void init_demo(VP_INT exinf);
void ether_handler(void);

void ether_tsk(VP_INT exinf);
void ip_tsk(VP_INT exinf);
void udp_tsk(VP_INT exinf);

void rcv_tsk(VP_INT exinf);

void check_cyc(VP_INT exinf);



#define ULENDIAN_CHANGE(x)  ((x) = ((((x)&0x000000ffUL)<<24) + (((x)&0x0000ff00UL)<<8) + ¥
                                (((x)&0x00ff0000UL)>>8) + (((x)&0xff000000UL)>>24)))
#define USENDIAN_CHANGE(x)  ((x) = (unsigned short)((((x)&((unsigned short)0x00ff))<<8) + ¥
                                (((x)&((unsigned short)0xff00))>>8)))

#define MAC_HEADER_SIZE     14
#define MAC_TYPE_OFFSET     12

#define PROTOCOL_IP     0x0800

#define IPPROTOCOL_UDP      17

#define IPV4_HEADER_SIZE    20
#define UDP_HEADER_SIZE     8
```

1.6.3 udpip.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "udpip.h"


/*********************************************/
/* init_demo                                 */
/*********************************************/
void init_demo(VP_INT exinf)
{
    /* Initialization of demonstration environment. */
    init_window();
}

/* 処理状況をカウントアップするための変数 */
struct
{
    int interrupt;
    int ether;
    int ip;
```

```
    int udp;
    int receive;
    int udp_lost;
} packet_count =
{
    0, 0, 0, 0, 0, 0
};

/*********************************************/
/* ether_handler                             */
/*********************************************/
void ether_handler(void)
{
    /* ネットワークコントローラから受信割り込みが入った時のハンドラ */
    isig_sem(ETHER_SEM);
    packet_count.interrupt++;
}

/*********************************************/
/* ether_tsk                                 */
/*********************************************/
void ether_tsk(VP_INT exinf)
{
    int length;
    unsigned char* rcv_data;
    unsigned short protocol;
    ER ercd;
    unsigned char* ip_packet;

    eth_opn_por(1, NULL, NULL);

    while(1)
    {
        while(wai_sem(ETHER_SEM)==E_OK)
        {
            /* Get packet data from controler */
            loc_cpu();
            if(0<(length=(int)eth_rcv_frm(1, &rcv_data)))
            {
                unl_cpu();

                /* MACパケットからプロトコルタイプを取得します */
                protocol = rcv_data[MAC_TYPE_OFFSET];
                protocol <<= 8;
                protocol |= rcv_data[MAC_TYPE_OFFSET+1];
                switch(protocol)
                {
                case PROTOCOL_IP:
                    if((ercd=get_mpf(PACKET_MPF, (VP*)&ip_packet))==E_OK)
                    {
                        /* MACパケットからIPデータ部分をコピーします */
                        *((int*)ip_packet) = length-MAC_HEADER_SIZE;
                        memcpy(&ip_packet[sizeof(int)], &rcv_data[MAC_HEADER_SIZE],
                                                                    length-MAC_HEADER_SIZE);

                        /* IPデータを送ります */
```

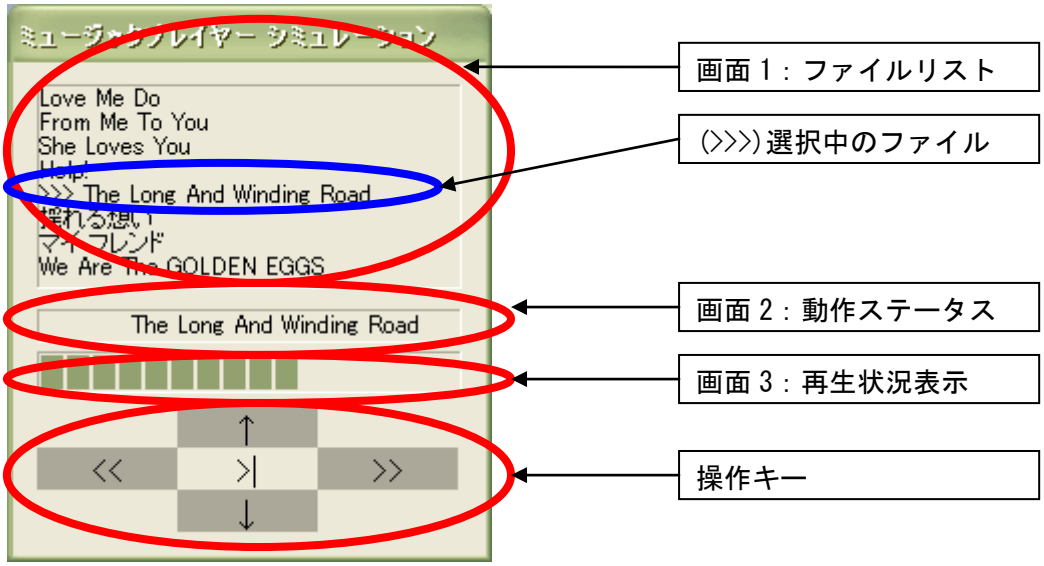
```
                        if((ercd=snd_dtq(IP_DTQ, (VP_INT)ip_packet))==E_OK)
                        {
                            packet_count.ether++;
                        }
                        else
                        {
                            rel_mpf(PACKET_MPF, ip_packet);
                        }
                    }
                    break;
                }

                if(rcv_data!=NULL)
                {
                    /* Release packet data to controler */
                    loc_cpu();
                    eth_rel_rbf(1, rcv_data);
                    unl_cpu();
                }
            }
            else
            {
                unl_cpu();
            }
        }
    }
}

/*********************************************/
/* ip_tsk                                    */
/*********************************************/
void ip_tsk(VP_INT exinf)
{
    ER ercd;
    unsigned char* ip_data;
    int length;
    unsigned char* ip_packet;
    unsigned char protocol;
    unsigned char* udp_packet;

    while(1)
    {
        if((ercd=rcv_dtq(IP_DTQ, (VP_INT*)&ip_data))==E_OK)
        {
            length = *(int*)ip_data;
            ip_packet = &ip_data[sizeof(int)];

            /* IPのチェックサムを計算します(削除不可) */
            if(__checksum_ip((unsigned short*)ip_packet, IPV4_HEADER_SIZE, 0)==0)
            {
                /* 本来ならここで自IP address宛て(ブロードキャスト含む)のチェック */
                /* やIPフラグメントパケット処理などを行う必要があります。 */
                /* 実習プログラムではコードを簡略化するため、処理を省略しています */

                protocol = ip_packet[9];
                switch(protocol)
```

```
                {
                case IPPROTOCOL_UDP:
                    if((ercd=get_mpf(PACKET_MPF, (VP*)&udp_packet))==E_OK)
                    {
                        /* IPデータからUDPデータ部分をコピーします */
                        *((int*)udp_packet) = length-IPV4_HEADER_SIZE;
                        memcpy(&udp_packet[sizeof(int)], &ip_packet[IPV4_HEADER_SIZE],
                                                                    length-IPV4_HEADER_SIZE);

                        /* UDPデータを送ります */
                        if((ercd=snd_dtq(UDP_DTQ, (VP_INT)udp_packet))==E_OK)
                        {
                            packet_count.ip++;
                        }
                        else
                        {
                            rel_mpf(PACKET_MPF, udp_packet);
                        }
                    }
                    break;
                }
            }

            /* IPデータを破棄します */
            rel_mpf(PACKET_MPF, ip_data);
        }
    }
}

/*********************************************/
/* udp_tsk                                   */
/*********************************************/
void udp_tsk(VP_INT exinf)
{
    ER ercd;
    unsigned char* udp_data;
    int length;
    unsigned char* udp_packet;
    unsigned char* receive_data;
    int receive_length;

    while(1)
    {
        if((ercd=rcv_dtq(UDP_DTQ, (VP_INT*)&udp_data))==E_OK)
        {
            length = *(int*)udp_data;
            udp_packet = &udp_data[sizeof(int)];

            /* UDPのチェックサムを計算します(削除不可) */
            if(__checksum_udp((unsigned short*)udp_packet, length, 0)==0)
            {
                /* 本来ならここでポート番号のチェックなどを行う必要があります。 */
                /* 実習プログラムではコードを簡略化するため、処理を省略しています */

                /* ユーザのデータ領域を取得します */
                /* (ユーザタスクとの通信方法は変更しないものとします) */
```

```
                if((ercd=prcv_dtq(RECEIVE_DTQ, (VP_INT*)&receive_data))==E_OK)
                {
                    /* UDPデータから実際のデータ部分をコピーします */
                    receive_length = *((int*)receive_data);
                    if((length-UDP_HEADER_SIZE)<receive_length)
                    {
                        receive_length = length-UDP_HEADER_SIZE;
                    }
                    *((int*)receive_data) = receive_length;

                    memcpy(&receive_data[sizeof(int)], &udp_packet[UDP_HEADER_SIZE],
                            receive_length);
                    packet_count.udp++;

                    /* 受信タスクを起床します */
                    wup_tsk(RCV_TSK);
                }
                else
                {
                    packet_count.udp_lost++;
                }
            }

            /* UDPデータを破棄します */
            rel_mpf(PACKET_MPF, udp_data);
        }
    }
}

/*********************************************/
/* rcv_tsk                                   */
/*********************************************/
void rcv_tsk(VP_INT exinf)
{
    ER ercd;
    unsigned char receive_data[1520];
    int length;

    /* このタスクはユーザ側タスクになります */

    while(1)
    {
        /* データ領域の先頭に、領域のサイズを設定します */
        *((int*)&receive_data[0]) = sizeof(receive_data);

        /* 受信タスクのデータ領域を送ります */
        if((ercd=snd_dtq(RECEIVE_DTQ, (VP_INT)receive_data))==E_OK)
        {
            if(slp_tsk()==E_OK)
            {
                /* データを受信しました */
                packet_count.receive++;
            }
        }
    }
}
```

```
/*********************************************/
/* check_cyc                                 */
/*********************************************/
void check_cyc(VP_INT exinf)
{
    /* 処理できたパケット数を表示します */
    iloc_cpu();
    printf("interrupt:%d¥n", packet_count.interrupt);
    printf("ether    :%d¥n", packet_count.ether);
    printf("ip       :%d¥n", packet_count.ip);
    printf("udp      :%d¥n", packet_count.udp);
    printf("receive  :%d¥n", packet_count.receive);
    printf("udp_lost    = %d¥n", packet_count.udp_lost);
    printf("packet lost = %d¥n", __lost_count());

    packet_count.interrupt = 0;
    packet_count.ether = 0;
    packet_count.ip = 0;
    packet_count.udp = 0;
    packet_count.receive = 0;
    packet_count.udp_lost = 0;
    printf("¥n");
    iunl_cpu();
}
```

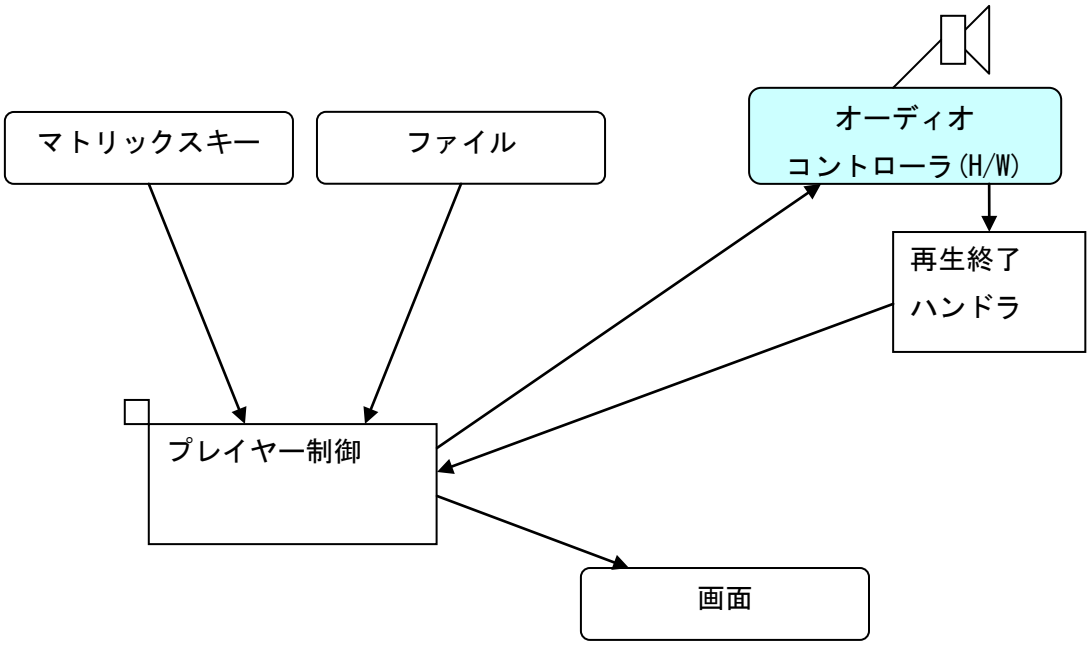
2　課題：携帯音楽プレイヤーをシミュレーション環境で試作



2.1　仕様

- ファイルリストの中から再生するタイトルを選択し、楽曲を再生する
- 操作キー入力は感圧式であり、ポーリングでボタン押下を確認する必要がある
- ファイルリストはシミュレーション上では固定とする
- シミュレーション上での操作は、選曲(↑ ↓)/再生(>|)[再生停止中に押下]/停止(>|)[再生中に押下]のみ行う
- 早送り(>>)/巻き戻し(<<)については、今回の実習では操作しない(押されていても無視する)
  - 課題が早くできた人は、操作と表示方法を工夫して機能を追加しても良い
- 動作ステータスは、Stop/Play 中の曲名を表示させる
- 動作ステータスのうち、Play 中の曲名を表示させるときは、横スクロールを行う
- 実際の音楽再生は行わないが、再生用ハードウェア(オーディオコントローラ)は仮想的に用意
  - オーディオコントローラは、再生データを送った後、しばらくしてから empty 割込みが発生する
  - オーディオコントローラの割込みは、割込み禁止関数の呼び出しによってのみ停止する

2.2　初期プログラムのタスク構成



2.3　初期プログラムの問題点と実習内容

初期プログラムでは、キー入力や表示がスムーズに操作できない。また音楽再生がスムーズにいかない状態(play data underrun)も発生している。
その理由を考察し、新たにタスク設計を行い、スムーズな操作と再生が行えるようにする。

課題 1)
初期プログラムのタスク構成(タスク数=1)で、スムーズな操作と再生がどの程度まで行えるのか、処理を調整してみる。

課題 2)
タスク分割案を基にタスク分割と優先度設定を行い、スムーズな操作と再生が行えるようにプログラムを修正する。

2.4　シミュレーション用サポート関数(キーおよび表示関係)

int io_key_data(void);
機能：操作キーの状態を取得する。返り値は押下位置を示し、以下の値になる。

[↑]	0x200
[↓]	0x002
[<<]	0x040
[>|]	0x020
[>>]	0x010

void io_status_char(unsigned short c);
機能：動作ステータス表示部に 1 文字追加する。
以前に表示された文字の右側に指定された文字が追加され、動作ステータス表示は左に移動する。

void io_status_clear(void);
機能：動作ステータス表示部にある文字をすべてクリアする。

int io_list_add(char* str);
機能：ファイルリスト表示部に文字列を追加する。文字列が追加できた場合は 0 以外を、追加できなかった場合には 0 を返す。なおファイルリスト表示部は、自動的にスクロールしない。

int io_list_delete(int index);
機能：ファイルリスト表示部の index 位置の文字列を削除する。削除できた場合には 0 以外を、削除できなかった場合には 0 を返す。

int io_list_insert(int index, char* str);
機能：ファイルリスト表示部の index 位置に文字列を追加する。追加できた場合には 0 以外を、追加できなかった場合には 0 を返す。

int io_list_reset(void);
機能：ファイルリスト表示部にある文字列をすべてクリアする。本関数は常に 0 以外を返す。

int io_list_select(int index);
機能：ファイルリスト表示部の index 位置の文字列を選択状態に設定する。設定できた場合には 0 以外を、設定できなかった場合には 0 を返す。

void io_progress(int pos);

機能：再生状況表示部の表示位置を設定する。設定値は 0 から 100 までが有効。

2.5　シミュレーション用サポート関数(ファイルおよびオーディオ関係)

int io_file_open(char* filename);
機能：指定されたファイルをオープンする。本関数は常に 0 以外を返す。

int io_file_close(void);
機能：オープンされているファイルをクローズする。本関数は常に 0 以外を返す。

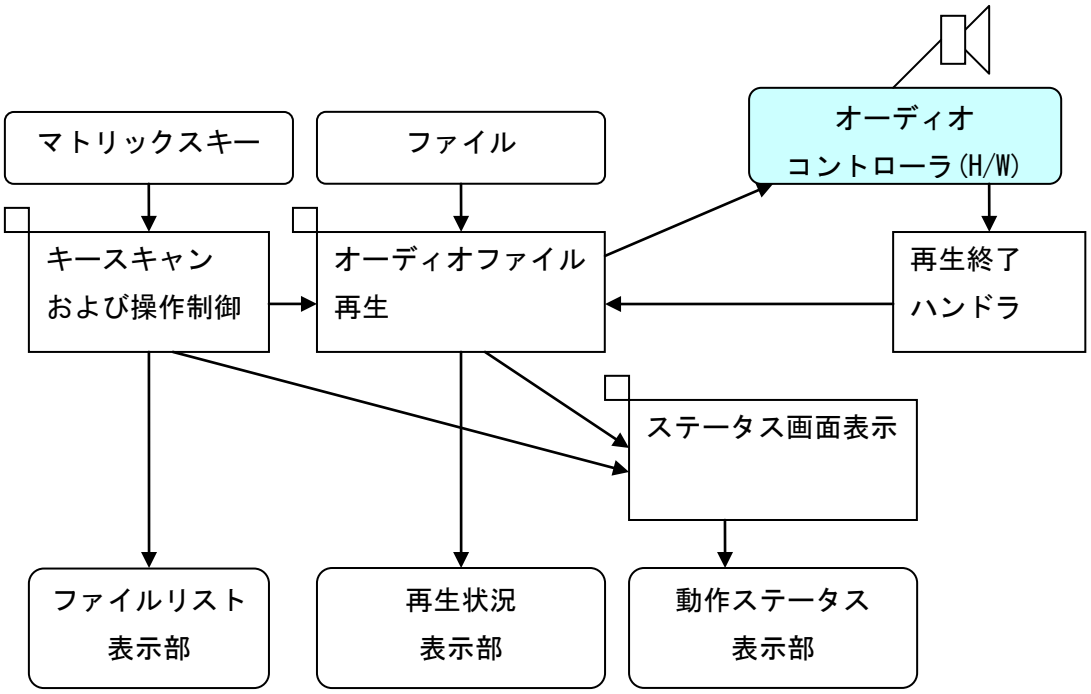
int io_file_read(char* buf, int readsize);
機能：オープンされているファイルから、データを読み出す。返り値は読み出したバイト数。

int io_sound_play(void* buf, int size);
機能：オーディオコントローラにデータを送り、再生を行う。割り込みが禁止されていた場合は、割り込みを許可する。本関数は常に 0 以外を返す。

int io_sound_stop(void);
機能：オーディオコントローラを割込み禁止に設定する。本関数は常に 0 以外を返す。

int io_sound_status(void);
機能：オーディオコントローラがデータを再生中かどうかを取得する。再生中の場合は 0 以外を、停止中の場合は 0 を返す。

2.6　タスク分割案



2.6.1 分割の理由

- dly_tsk の遅延とファイル読み込みの遅延が重なる場合など、全体の遅延時間が変わるため、キー操作やファイル読み込みとオーディオコントローラへデータを送る操作などのタイミングが一定でない
- まず出力側から考える
- 主な出力は画面表示と再生
- このうち、オーディオコントローラへデータを送る操作と再生状況表示については連動しているため、一連の処理(タスク)とする
- オーディオコントローラへデータを送る操作は、その前作業としてファイルからの読み込みが必要であり、ファイル操作も上記タスクに含めることにする
- 動作ステータス表示は、再生状況やキー操作状況に左右されないようにスクロールされるべきなので、別タスクとする
- 次に入力側を考える
- すべての操作はユーザのキー操作によって行われるため、主な制御はキー操作のタスクで処理する
- ファイルリスト表示はキー操作と連動しているため、キー操作のタスクで処理する

2.6.2優先度の設定

- キー操作のタスクはポーリングによりキースキャンを行うが、ユーザのキー操作が行われない限り、状態は変化しない
- オーディオデータの読み込み処理はある程度時間がかかるため、バックグラウンド処理として、空いている時間に読み込みを行う
- 動作ステータス表示は、空いている時間で行った方が良いが、キー操作のタスクよりも優先度を高くすると、操作性が低下する可能性がある
- 動作ステータス表示の優先度をオーディオデータの読み込み処理よりも低くすると、ファイルの読み込み状況に、スクロール表示の動作が左右されることになる
- したがって優先度は、キー操作のタスク(高)＞動作ステータス表示のタスク＞ファイルおよび再生のタスク(低)と設定すべき

2.7　初期プログラムのリスト

2.7.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/

INCLUDE("music.h");

#define STACK_SIZE 2048

DEF_INH(INT_MUSIC, {0, music_handler});

CRE_TSK(MUSIC_TSK, {TA_HLNG|TA_ACT, 0, music_tsk, 20, STACK_SIZE, NULL});


/* User initialize function */
ATT_INI({TA_HLNG, 0, init_demo});
```

2.7.2 music.h

```
#ifndef _MUSIC_H_
#define _MUSIC_H_

#define INT_MUSIC   5

// シミュレーション関数
int io_key_data(void);
void io_status_char(unsigned short c);
void io_status_clear(void);

int io_list_add(char* str);
int io_list_delete(int index);
int io_list_insert(int index, char* str);
int io_list_reset(void);
int io_list_select(int index);

void io_progress(int pos);

int io_file_open(char* filename);
int io_file_close(void);
int io_file_read(char* buf, int readsize);

int io_sound_play(void* buf, int size);
int io_sound_stop(void);
int io_sound_status(void);



// ユーザー関数
void music_handler(void);
```

```
void init_demo(VP_INT exinf);

void music_tsk(VP_INT exinf);

#endif /* _MUSIC_H_ */
```

2.7.3 music.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "music.h"

#include <stdio.h>
#include <string.h>

void init_window(void);
void set_pos(int pos);

int play_buffer_empty = 0;

/*********************************************/
/* init_demo                                 */
/*********************************************/
void init_demo(VP_INT exinf)
{
    /* Initialization of demonstration environment. */
    init_window();
}

/*********************************************/
/* music_handler                             */
/*********************************************/
void music_handler(void)
{
    if(play_buffer_empty!=0)
    {
        /* 次の割込みが発生するまでに処理できなかったかどうかを検証する */
        printf("play data underrun!¥n");
    }

    play_buffer_empty++;
}

/* 再生する曲の一覧(実習環境では固定で用意) */
typedef struct
{
    char* name;
    int size;
} PLAY_FILE_LIST;

PLAY_FILE_LIST filelist[] = {
    "Love Me Do", 12345,
    "From Me To You", 12345,
    "She Loves You", 12345,
    "Help!", 12345,
    "The Long And Winding Road", 12345,
    "揺れる想い", 12345,
    "マイ フレンド", 12345,
    "We Are The GOLDEN EGGS", 12345,
    "全力少年", 12345,
```

```
    "What A Wonderful World", 12345,
    "First kiss", 12345,
    "地上の星", 12345,
    "旅人のうた", 12345,
    "空と君のあいだに", 12345,
    "時代", 12345,
    "ヘッドライト・テールライト", 12345
};

/* 配列の数を取得するマクロ */
#define LIST_COUNT  sizeof(filelist)/sizeof(filelist[0])


/*********************************************/
/* music_tsk                                 */
/*********************************************/
void music_tsk(VP_INT exinf)
{
    ER ercd;
    int key_data;

    int select;
    int select_max;
    int offset = 0;

    int key_data_old = 0;

    PLAY_FILE_LIST* play_file = NULL;
    int play_status_char = 0;
    int play_status_scroll = 0;

    int play_file_pos = 0;
    char play_file_buf[512];
    int play_file_bufsize = 0;
    int read_size;

    /* 曲の一覧を画面に設定します */
    for(select_max=0; select_max<LIST_COUNT; select_max++)
    {
        if(io_list_add(filelist[select_max].name)==0)
        {
            select_max--;
            break;
        }
    }

    /* 初期選択位置を設定します */
    select = 0;
    io_list_select(select);

    while(1)
    {
        /* キースキャンのために、少し時間間隔を空けます */
        ercd = dly_tsk(50);
        key_data = io_key_data();
```

```
switch(key_data)
{
case 0x200: /* upキーが押されました */
    printf("up¥n");
    if(io_list_select(select-offset-1))
    {
        select--;
    }
    else if(0<select && 0<offset)
    {
        /* 一番上まできたら、スクロールさせます */
        offset--;
        select--;
        io_list_delete(select_max);
        io_list_insert(0, filelist[select].name);
    }
    break;

case 0x040: /* leftキーが押されました */
    printf("left¥n");
    break;

case 0x020: /* centerキーが押されました */
    if(0<=select)
    {
        if(key_data_old!=key_data)
        {
            printf("center¥n");

            if(!io_sound_status())
            {
                /* 再生中の曲が無ければ、選択された曲を再生します */
                io_sound_stop();

                if(io_file_open(filelist[select].name)!=0)
                {
                    /* 再生用の設定を行います */
                    play_file = &filelist[select];

                    play_status_char = 0;
                    play_status_scroll = 0;

                    /* 最初の再生用データを読み込みます */
                    play_file_bufsize = io_file_read(play_file_buf, sizeof(play_file_buf));
                    play_file_pos = 0;

                    /* オーディオコントローラに、最初の再生データを送ります */
                    play_buffer_empty = 0;
                    io_sound_play(play_file_buf, play_file_bufsize);
                }
            }
            else
            {
                /* 再生中の曲があれば、その曲を停止します */
                io_sound_stop();
                io_status_clear();
```

```
                io_progress(0);
                io_file_close();

                play_file = NULL;
                printf("stop by user¥n");
            }
        }
    }
    break;

case 0x010: /* rightキーが押されました */
    printf("right¥n");
    break;

case 0x002: /* downキーが押されました */
    printf("down¥n");
    if(io_list_select(select-offset+1))
    {
        select++;
    }
    else if(select<(LIST_COUNT-1))
    {
        /* 一番下まできたら、スクロールさせます */
        select++;
        io_list_delete(0);
        io_list_add(filelist[select].name);

        offset++;
        io_list_select(select-offset);
    }
    break;
}

key_data_old = key_data;

if(play_file!=NULL)
{
    /* 再生中の場合 */

    if(play_status_scroll==0)
    {
        /* 再生中の曲名を一文字ずつ設定します */
        io_status_char(*(unsigned short*)&play_file->name[play_status_char]);
        play_status_char = (play_status_char+1)%strlen(play_file->name);

        if(play_status_char==0)
        {
            /* 全部の文字を表示したあとは、スクロール処理を行います */
            play_status_scroll = 1;
        }
    }
    else
    {
        /* スペース文字を送ることでスクロールさせます */
        io_status_char(*(unsigned short*)"  ");
```

```
            play_status_scroll = (play_status_scroll+1)%60;
            if(play_status_scroll==0)
            {
                /* 表示文字をすべて送り終わったあとは、再度最初から表示させます */
                play_status_char = 0;
                io_status_clear();
            }
        }

        if(play_buffer_empty!=0)
        {
            /* 再生データが空になっていた場合 */
            play_file_pos += play_file_bufsize;

            /* 再生済みの表示を更新します */
            io_progress((play_file_pos*100)/play_file->size);

            if(play_file->size<=play_file_pos)
            {
                /* すべてのデータを再生していたら、停止状態に戻します */
                io_sound_stop();
                io_status_clear();
                io_progress(0);
                io_file_close();

                play_file = NULL;
                printf("stop¥n");
            }
            else
            {
                /* 次のデータを読み込みます */
                printf("next read¥n");

                if(sizeof(play_file_buf)<(play_file->size-play_file_pos))
                {
                    read_size = sizeof(play_file_buf);
                }
                else
                {
                    read_size = (play_file->size-play_file_pos);
                }
                play_file_bufsize = io_file_read(play_file_buf, read_size);

                /* オーディオコントローラに、次の再生データを送ります */
                play_buffer_empty = 0;
                io_sound_play(play_file_buf, play_file_bufsize);
            }
        }
    }
}
```

トロンフォーラム
実習：組込みリアルタイム・プログラミング(ITRON 中級編)
プログラミング演習テキスト

2016 年 10 月 26 日発行

発行所
トロンフォーラム
(YRP ユビキタス・ネットワーキング研究所内)
〒141-0031 東京都品川区西五反田 2-12-3 第一誠実ビル 9F
URL: http://www.tron.org/ja/
TEL:03-5437-0572(代表)　FAX:03-5437-2399(代表)
