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

解答例

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　解答例(UDP/IP 通信処理)

1.1　system.cfg

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

INCLUDE("udpip.h");

#define STACK_SIZE 2048

DEF_INH(INT_ETHER, {0, ether_handler});

#if 0
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});
#else
CRE_TSK(ETHER_TSK, {TA_HLNG|TA_ACT, ETHER_TSK, ether_tsk, 20, STACK_SIZE, NULL});
#endif
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});
```

1.2　udpip.c

```
#include  "kernel.h"

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

#if 1
void ip_proc(int length, unsigned char* ip_packet);
void udp_proc(int length, unsigned char* udp_packet);
#endif

/************************************************/
/* 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)
                    {
#if 0
                        /* 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);
                        }
#else
                        /* MACパケットからIPデータ部分をコピーします */
                        memcpy(ip_packet, &rcv_data[MAC_HEADER_SIZE],
                                                                length-MAC_HEADER_SIZE);

                        /* Release packet data to controler */
                        loc_cpu();
                        eth_rel_rbf(1, rcv_data);
                        unl_cpu();
                        rcv_data = NULL;

                        ip_proc(length-MAC_HEADER_SIZE, ip_packet);
                        rel_mpf(PACKET_MPF, ip_packet);

                        packet_count.ether++;
#endif
                    }
                    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                                    */
/*********************************************/
#if 0
void ip_tsk(VP_INT exinf)
#else
void ip_proc(int length, unsigned char* ip_packet)
#endif
{
    ER ercd;
#if 0
    unsigned char* ip_data;
    int length;
    unsigned char* ip_packet;
#endif
    unsigned char protocol;
    unsigned char* udp_packet;

#if 0
    while(1)
    {
        if((ercd=rcv_dtq(IP_DTQ, (VP_INT*)&ip_data))==E_OK)
        {
            length = *(int*)ip_data;
            ip_packet = &ip_data[sizeof(int)];
#endif

            /* 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 0
                    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);
```

```
                        }
                    }
#else
                    udp_proc(length-IPV4_HEADER_SIZE, &ip_packet[IPV4_HEADER_SIZE]);
                    packet_count.ip++;
#endif
                    break;
                }
            }

#if 0
            /* IPデータを破棄します */
            rel_mpf(PACKET_MPF, ip_data);
        }
    }
#endif
}

/*********************************************/
/* udp_tsk                                   */
/*********************************************/
#if 0
void udp_tsk(VP_INT exinf)
#else
void udp_proc(int length, unsigned char* udp_packet)
#endif
{
    ER ercd;
#if 0
    unsigned char* udp_data;
    int length;
    unsigned char* udp_packet;
#endif
    unsigned char* receive_data;
    int receive_length;

#if 0
    while(1)
    {
        if((ercd=rcv_dtq(UDP_DTQ, (VP_INT*)&udp_data))==E_OK)
        {
            length = *(int*)udp_data;
            udp_packet = &udp_data[sizeof(int)];
#endif

            /* 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++;
                }
            }

#if 0
            /* UDPデータを破棄します */
            rel_mpf(PACKET_MPF, udp_data);
        }
    }
#endif
}

/*********************************************/
/* 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　解答例(携帯音楽プレイヤー)

2.1　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, 10, STACK_SIZE, NULL});
#if 1
    CRE_TSK(PLAY_TSK, {TA_HLNG|TA_ACT, 0, play_tsk, 20, STACK_SIZE, NULL});
    CRE_TSK(SCROLL_TSK, {TA_HLNG|TA_ACT, 0, scroll_tsk, 15, STACK_SIZE, NULL});

    CRE_SEM(PLAY_SEM, {TA_TFIFO, 0, 10});
    CRE_DTQ(PLAY_DTQ, {TA_TFIFO, 10, NULL});

    CRE_DTQ(SCROLL_DTQ, {TA_TFIFO, 10, NULL});
#endif

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

2.2　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);
#if 1
void play_tsk(VP_INT exinf);
void scroll_tsk(VP_INT exinf);
#endif

#endif /* _MUSIC_H_ */
```

2.3 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)
{
    /* Initialize demo dialog. */
    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++;
#if 1
    isig_sem(PLAY_SEM);
#endif
}

/* 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;

#if 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;
#endif

    /* 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 */
#if 0
                        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
                        snd_dtq(PLAY_DTQ, (VP_INT)&filelist[select]);
                        snd_dtq(SCROLL_DTQ, (VP_INT)&filelist[select]);
#endif
                    }
                    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();

#if 0
                        play_file = NULL;
#else
                        snd_dtq(PLAY_DTQ, (VP_INT)NULL);
                        snd_dtq(SCROLL_DTQ, (VP_INT)NULL);
#endif
                        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 0
        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);
                }
            }
        }
#endif
    }
}

#if 1
/*********************************************/
/* play_tsk                                  */
/*********************************************/
void play_tsk(VP_INT exinf)
{
    ER ercd;
    PLAY_FILE_LIST* play_file = NULL;

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

    while(1)
    {
        if(rcv_dtq(PLAY_DTQ, (VP_INT*)&play_file)==E_OK)
        {
            while(play_file!=NULL)
```

```
{
    /* 再生を停止します */
    io_sound_stop();
    io_progress(0);
    io_file_close();

    if(io_file_open(play_file->name)!=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;

        do
        {
            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 */
                play_file = NULL;
                break;
            }
            else
            {
                /* オーディオコントローラに、再生データを送ります */
                play_buffer_empty = 0;
                io_sound_play(play_file_buf, play_file_bufsize);

                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);

                /* 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);
            }

            if(prcv_dtq(PLAY_DTQ, (VP_INT*)&play_file)==E_OK)
            {
                break;
            }
        } while((ercd=wai_sem(PLAY_SEM))==E_OK);
    }
    else
    {
        play_file = NULL;
    }
}
```

```
        }

        /* 再生を停止します */
        io_sound_stop();
        io_progress(0);
        io_file_close();

        snd_dtq(SCROLL_DTQ, (VP_INT)NULL);

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

/*********************************************/
/* scroll_tsk                                */
/*********************************************/
void scroll_tsk(VP_INT exinf)
{
    PLAY_FILE_LIST* play_file = NULL;
    int play_status_char = 0;
    int play_status_scroll = 0;

    while(1)
    {
        io_status_clear();

        if(rcv_dtq(SCROLL_DTQ, (VP_INT*)&play_file)==E_OK)
        {
            play_status_char = 0;
            play_status_scroll = 0;

            while(1)
            {
                if(play_file!=NULL)
                {
                    /* 再生中 */
                    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();
                        }
                    }
                }
                else
                {
                    break;
                }

                if(trcv_dtq(SCROLL_DTQ, (VP_INT*)&play_file, 80)==E_OK)
                {
                    if(play_file!=NULL)
                    {
                        /* 最初から表示しなおす */
                        play_status_char = 0;
                        play_status_scroll = 0;
                    }
                }
            }
        }
    }
}
#endif
```

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 解答例

2017 年 11 月 1 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 2-20-1 第 28 興和ビル
URL: http://www.tron.org/ja/
TEL:03-5437-0572(代表) FAX:03-5437-2399(代表)
