プロジェクト演習２

Linux プログラミング演習

最終報告書

Cy16179 渡辺爽太

1. はじめに

今回の授業ではLinuxプログラミングについて学んだ。今まで自分が使ってきたOSはwindowsであったが、今回用いたOSはUNIXである。UNIXとは特定のOSのことではなく、さまざまな種類があり、LinuxもUNIXの1つである。

本レポートでは組み込み開発環境の概要、プログラムの動作の概要、システムの詳細、プログラムのテスト、本演習の際に身につけた技術を記載する。

2. 組込み開発の概要

組込み機器では開発対象と開発環境が別なので、クロス開発を行う必要がある。これはセルフ開発のようにコンパイルしても、OSなどが異なっていることから正常に動作しないためである。クロスコンパイルをすることで開発環境と実行環境が別の場合でも正常に動作するようにする。具体的にはARM用クロスコンパイラを用いてコンパイルをする。ホスト側でクロスコンパイルをした後、scpコマンドを用いてファイルを転送し、ターゲット側でプログラムを実行させた。ここでは、Linuxを用いてプログラムを書いたPCがホストであり、またarmadillo-440がターゲットである。

またターゲットにはキーボードがないので、ターゲットにキーボードからコマンドを入力するのは、シリアル(本当はRS-232C)インターフェースとminicomというソフトを使い、ターゲットに送る。しかし、ターゲットはscpサーバになれないため、ターゲットからscpコマンドを使ってファイルを転送する必要がある。

3. 動作の概要

3.1 動作の詳細

今回作成したプログラムはサンプルドライバに以下の機能を追加した物である。作成手順としてはドライバコースの4をサンプルドライバの機能として追加した。その後、ドライバコースの1の機能を確認した後、ドライバコース2,3の機能を追加した。

ドライバコース1： ioctlによる情報取得

サンプルドライバでは関数tactsw_read()の中にwait_event_interrupt()というマクロ関数が存在し、そのためキー入力まで「待ち」の状態になってしまい、他の動作ができない。そこでioctlを使い、キーの状態を瞬時に知るという機能を追加した。

ドライバコース2： キーの組合せによる文字入力

キーの組合せにより出力されるものが変わるという機能である。今回ではスイッチを順に'1','2','4'とし、スイッチに対応した数字が出力されるようにした。同時に複数押された場合は対応した数字が足された数字を出力される。

ドライバコース 3 : シグナル化

指定したタクトスイッチが押されたときにシグナルを送る機能をつけた。具体的にはスイッチを全て押したとき、つまり'7'が出力されたときに、終了を意味するシグナルをアプリケーション側に送り、プログラムを終了させた。

ドライバコース 4 : オートリピートドライバ

スイッチを押し続けると連続して文字が入力される機能を追加した。具体的にはスイッチを押し、0.7秒後に同じスイッチを押し続けていた場合、0.1秒ごとに同じ数字を出力し続ける。

3.2 アプリケーションについて

アプリケーションは第三回で作成したinput.cを基にし、関数tactsw_ioctl()などの関数を追加した物である。関数open()により/dev/tactswを開き、その後永遠に関数read()を実行し続ける。ただし終了を意味するシグナルを受信した場合、プログラムを終了する。

ドライバコース1の「ioctlによる情報取得」の場合、関数read()などをコメント化し、関数ioctl()とsleep(1)だけを永遠にループさせ、ioctlが正常に動作しているか確かめた。付録では関数ioctl()、sleep(1)などはコメント化している。

3.3 ドライバについて

サンプルドライバにカーネルタイマ、関数my_timer_handler() (タイムアウト関数)、関数get_num()、関数ioctl()などを付け加えた物である。アプリケーション側の関数に対応した関数が実行される。スイッチからの入力をアプリケーション側に送るのが主な目的である。

3.4 プログラムの連携

アプリケーションの関数open()が実行されたときドライバ側の関数tactsw_open()が実行をされる。

アプリケーションの関数read()が実行されたときドライバ側の関数tactsw_read()が実行される。

アプリケーションの関数close()が実行されたときドライバ側の関数tactsw_release()が実行される。

アプリケーションの関数ioctl()が実行されたときドライバ側の関数tactsw_ioctl()が実行される。

4. システムの詳細

4.1 各関数の説明

・カーネルタイマについて

一般のアプリケーションなら sleep を使って指定した時間だけ寝ればいいだけだが割り込みハンドラの中では寝ることは許されない。そこでカーネルタイマを用いる。カーネルタイマとは、ドライバがハードウェアに対して DMA 指示を行ったあと、DMA 完了のタイムアウトを監視するために、一定時間後に特定の関数(タイムアウト関数)を呼び出すため仕組みである。

・my_timer_handler(unsigned long arg)について

これはカーネルタイマにより一定時間後に呼び出される関数であり、タイムアウト関数と呼ばれる。これは引数に unsigned long 型をとる。また今回は使わないが複数の引数を渡す場合は構造体のポインタを渡すようにする。

・関数 mod_timer()について

カーネルタイマは一度タイムアウトすると二度と呼ばれないワンショットタイマである。一定時間おきに関数を呼び出し続けたい場合、すなわちインターバルタイマを実現するにはタイムアウト関数の最後に関数 mod_timer()を宣言する必要がある。

・関数 init_timer()について

timer_list 構造体の初期化をするカーネル関数である。

・関数 add_timer()について

タイマの監視を開始する関数である。

・関数 del_timer()について

このままプログラムを実行すると関数 add_timer()が 2 回実行されるために kernal panic が起きる。その対処として関数 del_timer()を用いる。

・関数 get_num()について

対応した数字を返す関数である。対応しているスイッチが押されているか確認し、押されている場合は、変数 num に加える。return する際、数字を ASCII コード表より変換している。

4.2 テスト

ドライバコース 1,2 の場合

何もキーを入力していない場合、1 秒ごとに 0 が出力された。また何かキーを入力したとき、対応している数字が出力された。

ドライバコース 2,3,4 の場合

スイッチをそれぞれ A、B、C とする。そしてテストを行う。

テスト 1
A を押す→A を離す→B を押す→B を離す→C を押す→C を離す
結果
1→0→2→0→4→0
テスト 2
A を押す→B を押す→A を離す→B を離す→C を押す→B を押す→A を押す
結果
1→3→2→0→4→6→7→0 プログラム終了
テスト 3
A を押し続ける→A を離す→B を押す→A を押す→A と B を押し続ける→加えて C を押す
1→一定時間経過→1→1→1→1→1→1→1→0→2→3→一定時間経過→3→3→3→3→3→7→0 プログラム終了
テスト 4
高速で A を押す、離す、を繰り返す
1→0→1→0→1→0→1→0→1→0→1→0→1→0→1→0→1→0

以上のことから正常に動作していることが確認できた。

5. 反省

またドライバ側のプログラムだが、参考文献を読んでみると「タイムアウト関数が呼び出される前に、ドライバをアンロードする」場合について書いてあった。このテストを行わなかったことが今回の反省である。おそらく今のままではkernel panicが起きるので関数tactsw_exit()の中に関数del_timer()を入れれば解決すると考えられる。時間がなく、テストが出来なかったので次の機会にはこのテストも行おうと思う。

6. 終わりに

今回のプロジェクト演習2では初めてドライバの解析をした。人のプログラムを解析するという経験は初めてだったのでとても難しかったが、理解できたときは達成感が感じられた。今回は時間がなかったので次の機会にはもっと多くの拡張をしてみたい。

7. 参考文献

平田豊(2008), 「Linux デバイスドライバプログラミング」, SBクリエイティブ.
プロジェクト演習2　Linuxプログラミング　山崎　2018年度　第3版　(配布資料)

付録A ドライバ側のプログラム
サンプルドライバに変更を加えた部分だけ載せる

```
//cy16179 渡辺爽太　auto_repeat
/**
 * Sample driver for tact switch
 * File name: auto repeat
 * Target board: Armadillo 440

 * For Project-enshu 6 @ Shibaura Institute of Technology
 */
#include <linux/timer.h>
#include <linux/init.h>

#define INTERVAL (70)
#define INTERVAL_SHORT (10)
#define SW_A 1
#define SW_B 2
#define SW_C 4
int ch = '0',pre_ch = '0';//グローバル変数

struct timer_list my_timer;//カーネルタイマー宣言


static void my_timer_handler(unsigned long arg){//タイムアウト関数
   int mlen;
   unsigned long irqflags;
 if(pre_ch != ch){
    mod_timer(&my_timer,jiffies+INTERVAL);//mod_timer()にはerrorが存在しない
 }else{
```

```
    mod_timer(&my_timer,jiffies+INTERVAL_SHORT);//mod_timer()にはerrorが存在しない
    if(ch != '0'){
       spin_lock_irqsave(&(tactsw_info.slock), irqflags);
       mlen = tactsw_info.mlen;
       if (mlen < MSGLEN) {
          tactsw_info.msg[mlen] = ch;
          tactsw_info.mlen = mlen+1;
          wake_up_interruptible(&(tactsw_info.wq));
       }
       spin_unlock_irqrestore(&(tactsw_info.slock), irqflags);
    }
 }
 pre_ch = ch;
}

static int tactsw_open(struct inode *inode, struct file *filp)
{
   init_timer(&my_timer);//カーネルタイマー初期化!
   my_timer.expires = jiffies + INTERVAL;//ミリ秒後に起きる
   my_timer.function = my_timer_handler;//タイムアウト関数指定
   add_timer(&my_timer);
   int i;
   unsigned long irqflags;
   int retval = -EBUSY;
   spin_lock_irqsave(&(tactsw_info.slock), irqflags);
   if (tactsw_info.used == 0) {
      tactsw_info.used = 1;
      retval = 0;
   }
   spin_unlock_irqrestore(&(tactsw_info.slock), irqflags);
   return retval;
}

   spin_unlock_irqrestore(&(tactsw_info.slock), irqflags);
   return read_size;//←いつも1
}
```

```
int get_num(){
   int num = 0;
   if(!gpio_get_value(tactsw_info.buttons[1]))num += SW_A;
   if(!gpio_get_value(tactsw_info.buttons[2]))num += SW_B;
   if(!gpio_get_value(tactsw_info.buttons[3]))num += SW_C;
   return num + 48;//ASCIIコード表により変換
}

int tactsw_ioctl(struct inode *inode, struct file *filp,
                        unsigned int cmd, unsigned long arg)
{
   static struct pid *my_pid;
   int retval=0,i,errno,error;
   switch(cmd){
   case 1://fetch current status
      for (i = 0; i < tactsw_info.nbuttons; i++) {
         int gpio = tactsw_info.buttons[i],num;
         if(gpio_get_value(gpio)){
                        num = get_num();
                        if(!num)return;//num が0のときは何も返さない
                        return num - 48;//ASCIIコード表により変換
         }
      }
      break;
   case 2:// set signal
      my_pid = get_pid(task_pid(current));
      break;
   case 3:// clear signal
      put_pid(my_pid);
```

```
        break;
    case 4:// send signal
        error    = kill_pid(my_pid,SIGUSR1,1);
        if(!error){
            printk("%d¥n",errno);
            retval = -EFAULT;// other code may be better
        }
        break;
    default:
        retval = -EFAULT;         // other code may be better
    }
    return retval;
}

static int tactsw_release(struct inode *inode, struct file *filp)
{
    int error;
    del_timer(&my_timer);
    tactsw_info.used = 0;
    return 0;
}
static irqreturn_t tactsw_intr(int irq, void *dev_id)
{
    int i;
    for (i = 0; i < tactsw_info.nbuttons; i++) {
        int gpio = tactsw_info.buttons[i];
        if (irq == gpio_to_irq(gpio)) {
            int mlen;
            unsigned long irqflags;
            ch = get_num();//関数get_num()よりchを指定する
            spin_lock_irqsave(&(tactsw_info.slock), irqflags);
            mlen = tactsw_info.mlen;
            if (mlen < MSGLEN) {
                tactsw_info.msg[mlen] = ch;
                tactsw_info.mlen = mlen + 1;
                wake_up_interruptible(&(tactsw_info.wq));
```

```
            }
            spin_unlock_irqrestore(&(tactsw_info.slock), irqflags);
            return IRQ_HANDLED;
        }
    }
    return IRQ_NONE;
}
```

付録B　アプリケーション側のプログラム

```
#include<fcntl.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/ioctl.h>
#include<signal.h>
int end = 0;
void sig_handler(int signum){
    end = 1;
}
error_check(char *error){
    if(error < 0){
        perror(error);
        printf("¥n");
    }
    exit(1);
}
int main(){
    signal(SIGUSR1,sig_handler);
    char buff[1];
    int error,i,fd,cmd,ret,errno;
    fd = open("/dev/tactsw", O_RDONLY);
    if(fd<0) error_check("open_error");
    ret = ioctl(fd,2);
    if(ret < 0)error_check("ioctl_error (fd,2)");
    for(;;){
        /*
        sleep(1);//ioctlによる情報取得
```

```
        cmd = 1; //ioctlによる情報取得
        ret = ioctl(fd,cmd);//ioctlによる情報取得
        if(ret > 0)printf("%d",ret);//ioctlによる情報取得
        if(ret < 0) error_check("ioctl_error (fd,1)");//ioctlによる情報取得
       */
        error = read(fd,buff,1);
        if(error < 0)error_check("read_error");
        error = write(1,buff,1);
        if(error < 0)error_check("write_error");

        ret = ioctl(fd,1);//ioctlによる情報取得
        if(ret < 0)error_check("ioctl_error (fd,1)");
        if(ret == 7){
           error = ioctl(fd,2);//set signal
           if(error < 0)error_check("ioctl_error (fd,2)");
           error = ioctl(fd,4);//send signal
           if(error < 0)error_check("ioctl_error (fd,4)");
        }

        if(ret == 6){
           error = ioctl(fd,3);//clear signal
           printf("aaa¥n");
           if(error < 0)error_check("ioctl_error (fd,3)");
        }

        if(end){
           printf("end the program¥n");
           close(fd);
           exit(1);
        }
     }
}
```