Special

RTLinuxによるFAシステム構築

芳之内弘

株式会社ヨシノブレインとオリタシステム株式会社は、およそ10年前に段ボール製造工場のコルゲータライン制御のために、LANを構成した制御システムを開発し、現在までこれを販売しておりました。この制御システムは、自前の通信カードと自前の通信プロトコルと、5台のNEC PC-9801で構成されておりました。

7月の終わりごろ、新たにこのシステムを受注しましたので、今回はこのシステムのハードウェアをPC互換機とEthernetカードに切り替え、OSにはLinux(RTLinux)を使用したLANを構成し、納めることにしました。

私はLinuxでのプログラミングの経験が全くないのですが、RTLinuxの解説記事を読むと、DOSからの移植は容易だと書かれていたので、RTLinuxとGTK+、それにgccを使えば、DOSで組んだプログラムの移植は納期の3カ月で十分だろう、と軽く考えてスタートしました。ところがこの考えの甘さに気が付いたのはその後まもなくでした。

古いシステムのマルチタスクはこれも独自の方式で、Linuxの本格的なマルチタスクとは異なっています。割り込みの制御方法も異なります。また、GTK+を使用する場合には、すべてのグラフィックス表示をイベントドリブン方式に書き換える必要があります。従って、DOSのプログラムをLinuxシステムへ移植するために、その半分以上を作り変えることになってしまいました。友人の応援も得て5カ所のPCのプログラムが完成したのは、納品の2日前でした。

それでは今回の実施例について具体的に説明します。

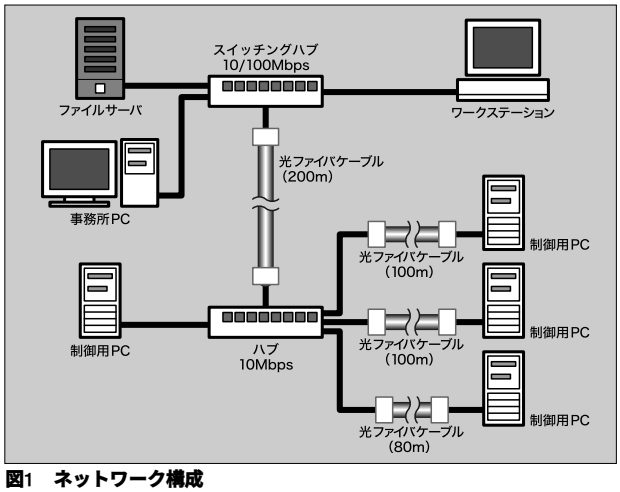

図1　ネットワーク構成

構成

●LANの構成

ネットワークの簡単な構成を図1に示しますので参考にしてください。段ボールを製造するためのコルゲータラインには制御用PCを4台配置します。また、事務所には制御データの送受信と編集を受け持つPCを1台配置します。これら5台のPCはEthernetでLANを構成しており、制御データは事務所PCからTCPパケットにより転送されます。生産中のデータもTCPパケットで各PCに伝達されます。この事務所のPCは、さらにワークステーションで構成されたネットワークにファイルサーバを経由してリンクされています。ファイルサーバは、ワークステーション側からNFSにより制御データをもらい、これを事務所PCに渡します。事務所のPCはこのデータを制御

側のネットワークに転送します。さらに、製品の生産データは事務所のPCとサーバを通してワークステーション側のネットワークに転送されます。

LANケーブルは耐ノイズ性と距離を考慮して、光ファイバーケーブルを採用し、光変換ユニットのコストを考えて、10Mbpsのものを使っています。

●ハードウェアの構成

PCはサーバを含めてすべてデスクトップ型のPC互換機です。CPUはCeleron 500MHzです。制御用PCのディスプレイは、発熱を抑えるためと小型化のために、14インチのLCDを採用しました。制御用のインターフェイスボードはすべて株式会社コンテックのPCIボードを使用しました。その種類は次の通りです。

- PIO-32/32L(PCI)……64ビット出力(絶縁型)
- PIO-64L(PCI)……32ビット入力32ビット主力(絶縁型)
- CNT24-4(PCI)……4チャンネル(AB相)カウンタ

コンテックのボードは、PCIバスタイプは今回が初めてでしたが、Cバスタイプ、ISAバスタイプでの使用実績がかなりあり、安心して採用しました。ところが、PIO-32/32Lボードにはバグがあるのではないかと思われます。購入した7枚のうち、2枚に動作不安定なものがありました。出力にフラフラと反転するビットがあって、当初はソフトウェアのバグか、配線の間違いを疑いましたが、どうもボードが疑わしいということで交換したのですが直らず、とうとうPCまで交換しました。それでも直らないので、さらに別のボードに交換してみると直りました。結局2枚とも不良品だったわけです。それも購入直後の新品のボードでしたので、このボードには注意が必要です。

●ソフトウェアの構成

特にマルチタスクのプログラムを作成するときは、その構成を決定してから取り掛かる必要があります。私の場合は最初は全く何も分からず、単にDOSプログラムをそのままgccで書き直すぐらいにしか考えていなかったので、コーディングを進めてから構成を考え直すという手順を踏んでしまいました。そういうわけで、今回の例は最適の構成にはなっていないと思います。

・モジュールおよびプロセス間のデータの共有

描画プロセスとメインプロセスの間はFIFOでデータを転送します。メインプロセスから描画プロセスへの転送はシグナルSIG_RD1でイベントを発生させて知らせます。SIG_RD1は独自に定義したシグナルです。逆方向の転送ではメインプロセスがselect関数でFIFOをスキャンして読み込みます。

パケットの受信プロセスとメインプロセス間はパイプ(名前なしFIFO)でデータを転送します。そしてお互いはシグナルでハンドシェイクしています。

メインプロセスとリアルタイムプロセスの間はrtl FIFOでデータを転送します。メインプロセスのrtl FIFOの書き込みでは、リアルタイムモジュールに割り込みが発生するので、シグナルは不要です。リアルタイムモジュールからメインプロセスへの書き込みは、メインプロセス側のselect関数でFIFOをスキャンして読み込みます。

・I/O制御

I/Oの制御はリアルタイムモジュールで一括処理します。モジュール別にこれを実行するとデータの整合が複雑になりますので、1つのモジュールで統一したほうが確実です。

PCIボードのデバイスドライバは入手できないことと、作る余裕がないので、PCIボードのベースアドレスを基板ごとに取得して、I/Oアドレスを直接アクセスして制御します。

・リアルタイムモジュール

周期タイマ割り込みによるスレッドのタイマ周期は0.5ms、1.0ms、2ms、4ms、200msを使用しています。また、リアルタイムFIFOを合計6個使用していますので、ハンドラも6つ作成します。

具体的な構成を図2に示しますので参考にしてください。実際の構成はもっと複雑ですが、説明のため簡略化しています。

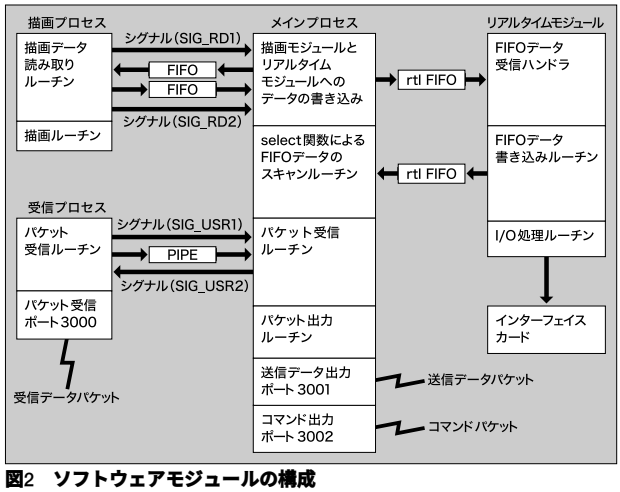

図2　ソフトウェアモジュールの構成

プログラミングの上での注意点

それでは私がプログラミングをする上で苦労した点を取り上げてみます。

●DOSからの移植の注意

私もそうでしたが、制御用のDOSプログラムをLinuxに移植する要求がかなりありそうなので、その場合の注意点を簡単に説明します。

いうまでもなくLinuxはマルチタスクOSなので、これを活用するならば、必然的に複数個のモジュールやプロセスに分けてプログラミングする必要があります。このモジュールの間ではデータエリアは全く独立しています。従って、モジュール間でデータを共有したい場合には次の方法があります。

- 共有メモリを使用する。
- FIFOでデータをやりとりする。
- パイプ(名前なしFIFO)でデータをやりとりする。

共有メモリの場合はモジュール間での共有メモリのアクセス制御が複雑になります。私の場合はFIFOとパイプを用いてデータをやりとりすることにしました。

もちろん、単一モジュールでのプログラミングならば、データ領域も単一ですので共有の必要は発生しません。

グラフィックスの表示にはGTK+を用いると比較的容易にプログラミングできます。ただし、GTK+は完全にイベントドリブンのモジュールになります。FIFOでデータを受け渡ししたり、表示を変える場合にはシグナルなどでイベントを発生させます。

●RTLinuxを使う上での注意

私が用いたのは、カーネル2.2.14にRTLinux-2.2のパッチを当てたものですが、このバージョンでは `stdlib.h`が使えません。Ver. 3.0では解消しているそうですが、開発版なので私は使いませんでした。私がこのバグで困ったのは、`atol()`や`atoi()`関数が使えないことです。仕方がないので、自前の関数を作って対応しました。

また、Ver. 2.xのRTLinuxではシステムコールは一切使えません。従って、システムコールはすべてユーザーレベルのモジュールで実行します。

●GTK+/GDKを使う上での注意

GTK+で描画を行っているときに割り込みが発生し、その割り込みイベント処理でさらに描画を行うと、GTK+のエラーが発生します。このエラーが発生すると描画の画面が固まり、以後の描画は実行しなくなります。そこで、割り込みイベント処理ルーチンでの描画を禁止するか、あるいは描画の開始前にフラグを立てて、描画が重なって実行されないように処理を変更します。そうすれば、この問題は回避できます。

キー割り込みのイベント処理の中では、プログラムのコードの順番に描画を実行するとは限りません。これはほかのイベントでも起こることかもしれません。例えば、最初の文字を描画して、次のシステムコールを実行します。その後でさらに描画を実行する場合には、最初の描画は次のシステムコールの実行が終了するまで待機されます。私の場合は、モニタの設定(`XF86Config`)で1024ドット×768ドット、16bitのカラーモードを8bitに変更すると、この問題は解決しました。この対策ができない場合は、前者の問題点と絡んで、とんでもない不具合が発生することがありますので注意が必要です。

ネットワーク機器の注意点

ネットワーク機器の設置経験が豊富な人はもちろん承知のことでしょうが、現地にインストールする前に、実際に使用する条件で実際の機器をテストすることが肝要です。メーカーや代理店が言う相性の保証をうのみにしないことです。特にハブの100Mbps／10Mbpsの自動認識については注意する必要があります。私の場合は、現地でスイッチングハブが10Mbpsの光変換モジュールを自動認識できず、ネットワークが全く動かなくて困りました。納入前に、同じ組み合わせでテストすることを忘れていたのが失敗でした。仕方なく10Mbpsのリピータハブを現地で購入してしのぎました。代理店では保証済みの組み合わせでしたが、全く信用できませんでした。代理店は型番の異なる機器でテストをして保証していたのです。接続テストの手抜きは後でひどい目に遭います。

また、友人の使用しているEthernetカードでは、100Mbps／10M自動認識で次のようなトラブルがありました。このカードと10Mbpsのハブとを組み合わせてTCPパケットを転送する場合、コネクションの開始に数秒も必要とするために、全く実用になりませんでした。

ネットワーク全体を10Mbpsか100Mbpsで統一すればこのような問題もないのでしょうが、コストや購入の容易さなどから、どうしてもスピードは混在しがちなので、機器の組み合わせには注意が必要です。特にスイッチングハブは10Mbpsのものが手に入らないので、スピードの混在が生じがちです。光変換ユニットが安くなれば、自動認識タイプはやめて、

100Mbpsに統一するのがベストな方法だと思います。

実施例の具体的な説明

具体的にソフトウェアを組んでいきますと、意外なところでつまずきます。例えば、「LinuxではPCIボードはどうやってアクセスすれば良いのか？」また「シリアルポートのアクセスはどうすれば良いのか？」など、何ごとも初体験だと疑問だらけで取り掛かりに時間を取られてしまい、肝心な制御の部分に時間を割くことができなくなります。そこでこのような疑問を持っている人のお役に立てるように、実施例を具体的に説明します。

●Linuxシステムの環境

telnetとftpを使ったリモートメンテナンスや、現地でのソフトウェアの変更とコンパイルなどを考慮して、Kondara MNU/Linux 1.1の環境がIDEのハードディスクにそのままインストールしてあります。そしてカーネルはReiserFSとRTLinux-2.2のパッチを当てたVer. 2.2.14をインストールしています。

●プログラムの自動立ち上げ

PCを制御に使用する場合、オペレータはシステム全体の電源スイッチを入れたり、切ったりするだけです。従って、Linuxへのログインとアプリケーションの起動も自動的に実行しなくてはいけません。この自動ログインと自動起動の方法はいろいろあると思いますが、私は次の方法を実施しました。この方法が最適とは思いませんが、とりあえずこれで起動しています。

・/etc/inittabの編集

inittabをリスト1のように編集して自動ログインにします。username(ユーザー名)はrootに変更するとroot権限でログインできます。-fを指定するとパスワードは必要ありません。また、リスト2を追加してXサーバ、ウィンドウマネージャ、作成した制御プログラムが自動起動するようにします。

リスト1　inittab

```
2:2345:respawn:/bin/login -f username <dev/tty1>/dev/tty1 2>/dev/tty1
```

リスト2　inittabでのプログラムの自動実行

```
x6:2345:respawn:sh -c ‘cd /home;PATH=$PATH:/usr/X11R6/bin;/usr/local/bin/start.bin‘
```

```
#!/bin/bash
cd /home
source ~/.bashrc

# RAMドライブを作成してマウントします。このRAMドライブには
# FIFOを作ります。
mke2fs /dev/ram
mount /ram

# RTLinux付属のモジュールをインストールします。
insmod /usr/src/rtlinux-2.2/modules/rtl_time.o
insmod /usr/src/rtlinux-2.2/modules/rtl_sched.o
insmod /usr/src/rtlinux-2.2/modules/rtl_posixio.o
insmod /usr/src/rtlinux-2.2/modules/rtl_fifo.o

# 制御用に作成したリアルタイムモジュールをインストールします。
insmod /home/rt/rtl.o

# XサーバとWindow Managerをバックグラウンドで起動します。
/usr/X11R6/bin/startx  &

# 最後に制御プログラムを起動します。
/home/program
```

**リスト3
起動スクリプト**

・起動スクリプトの作成

/usr/local/binに起動スクリプト「start.bin」を作成します。リスト3に私が作ったスクリプトを掲載します。

システムのシャットダウンは、オペレータがメインキーをオフにしたときに、PCのプログラムが「shutdown -h now」を自動的に実行します。そして一定時間経過後、電源のスイッチがオフにされます。フラッシュROMにシステムを組めば、電源のオン、オフの制御も楽になりますが、X Window Systemを使っていることなどで、現在はIDEのハードディスクにすべてのソフトウェアを入れています。

リスト4　I/O基板の情報を取得するプログラム

```
/*************************************************************
   PCI Board Base Address and Interrupt level Get Routine
*************************************************************/
#define     VENDOR      0x1221              // ContecのベンダID
#define     DEVICE_PIO   0x8122             // PIO-32/32L(PCI)のデバイスID

void  get_pci_base_address(void ) {
  ushort  i, idata1,idata2;
  long        laddr,ldata;
  int     iboard_no=0;

  ldata = idata1 = idata2 = 0;
  basePIO1 = basePIO2 = -1;
  irq_no_PIO1 = irq_no_PIO2 = 0xff;

  for(i=0;i<32;i++){                        //最大32枚のボードをチェックします。
      ldata = (i << 11) | 0x80000000;
      outl_p(ldata,0xcf8);                  //情報の読み込みコマンド
      idata1=inw_p(0xcfc);                  //ベンダ情報を読み込む
      if(idata1==VENDOR){
       idata2=inw(0xcfe);                   //デバイス情報を読み込む
       if(idata2==DEVICE_PIO){
           iboard_no++;
           ldata = (i << 11) | 0x80000010;  //Address Base Reg.No1
           outl_p(ldata,0xcf8);             //ベースアドレス読み込みコマンド
           laddr = inw_p(0xcfc);            //ベースアドレスを読む
           if(iboard_no==1)
               basePIO1 = (uint)(laddr & 0x0000fffc);
           else if(iboard_no==2)
               basePIO2 = (uint)(laddr & 0x0000fffc);

           ldata = (i << 11) | 0x8000003c;  //割り込みレベル読み込みコマンド
           outl_p(ldata,0xcf8);             //コマンド出力
           if(iboard_no==1)
               irq_no_PIO1 = inb_p(0xcfc);  //最初の基板をNO1とする
           else if(iboard_no==2)
               irq_no_PIO2 = inb_p(0xcfc);  //次の基板をNO2とする

           if(i==31)                        // 不要？
              break;
       }
      }
  }
  ldata = 0;
  outb_p(ldata, 0xcf8);

  if(basePIO1 == -1)
    printk("PIO1 Board Base Address Error \n");
  if(basePIO2 == -1)
    printk("PIO2 Board Base Address Error \n");
}
/*************************************************************************/
```

●PCIボードの制御

コンテックのボードではLinux用のデバイスドライバが入手できませんので、I/Oアドレスを直接アクセスして制御します。PCIボードの制御には、そのコンフィギュレーションレジスタの内容を取得する必要があります。

PCIボードのベースアドレスや割り込みレベルはPCメインボード上のPCIブリッジが管理しています。このPCIブリッジより情報を得る方法を説明します。リスト4のプログラムは、I/Oボード(PIO-32/32L)の情報を取得するものです。実装されているコードにはさらにほかのボードの検出も含まれるのでもうすこし複雑になりますが、原理は同じです。PCIバスの詳しい説明は、記事末のRESOURCE「PCIバスに関するもの」を参照してください。

リスト4のルーチンで取得したベースアドレスでI/Oをアクセスするには、以下のようにします。

```
outb(outdata,basePIO1+6);
//データoutdataを7番目のポートに出力

indata = inb(basePIO2+1);
//2番目のポートから入力
```

●シリアルポートの制御

PC互換機標準のCOM1、COM2の制御は、メインモジュールに次の方法で実装しました。リスト5はCOM2をアクセスするためのルーチンです。シリアルポートの設定をより詳しく知りたい人は、「man ioctl」を実行してマニュアルを読んでください。なお、リスト5のインクルードファイルには不要なものも含まれています。このルーチンを使用してポートに

リスト5　シリアルポートの制御

```
/*********************************************************
  RS232C Program     by H.Yoshinouchi
*********************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/termios.h>
#include <sys/ioctl.h>

#include <unistd.h>
#include <sys/types.h>

int com_fd2;
/*************************************************
  Open RS232C port (COM 2 ポートのオープン)
*************************************************/
int   open_com2(const char *device) {
  struct termios fd_attr;              //8bit, parity no,stop 1

  com_fd2 = open(device,O_RDWR | O_NOCTTY | O_NONBLOCK);
                                       //read/writeモードでオープン
  if (com_fd2 == -1) {
      fprintf(stderr,"failed to open %s: %m",device);
      return(com_fd2);
  }
  fd_attr.c_iflag = IGNBRK | IGNPAR;
  fd_attr.c_oflag = 0;
  fd_attr.c_cflag = CS8                // 8bit設定
      | CLOCAL                         //モデム制御線無視
        | CREAD;                       //受信有効
  fd_attr.c_lflag = 0;
  cfsetospeed(&fd_attr,B9600);         // 送信ボーレート 9600
  cfsetispeed(&fd_attr,B9600);         // 受信ボーレート 9600
  if (tcsetattr(com_fd2,TCSANOW,&fd_attr) == -1) {
      fprintf(stderr,"failed to tcsetattr on %s: %m",device);
      close(com_fd2);
      com_fd2=-1;
```

リスト5　続き

```
        return(com_fd2);
    }
    return(com_fd2);
}
/
*****************************************************
  Send Data to Com2 (Com2 ポート 送信ルーチン)
*****************************************************/
int  com_send2(char *buf, int isize){

      com_fd2 = open_com2("/dev/ttyS1");
      if (com_fd2==-1) {
            printf("Can't Open Port!\n");
            exit(0);
      }
         write(com_fd2, buf, isize);
      close(com_fd2);
      return 0;
}

/
*****************************************************
  Read Data from Com2 (Com2 ポート受信ルーチン)
*****************************************************/
int  com_read2(char *buf, int isize){
  int isize;
      com_fd2 = open_com2("/dev/ttyS1");
      if (com_fd2==-1) {
            printf("Can't Open Port!\n");
            exit(0);
      }
         isize = read(com_fd2, buf, isize);
      close(com_fd2);
      return isize;
}
/
*****************************************************/
```

メッセージを送信するには次のようにします。

```
char buf_tx[] = {"This is the test
communication of Com2\r\a"};
com_send2(buf_tx,sizeof(buf_tx));
```

最後に

組み込みPCにLinuxをインストールした応用例はかなりありますが、デスクトップPCでの応用はあまり聞きません。そこで、今回の私の経験をまとめ、Linuxの有効な応用に貢献したいと思いました。今回の方法は最適なものではありません。時間のない中で、プログラミングの参考書と格闘しながら完成したものですので、改良の余地は多々あると思います。この記事を読まれ、さらに良い方法をお気付きの方がいらっしゃいましたら、ご指摘していただけますと幸いです。私が今回参考にした文献を記事末のRESOURCEに挙げましたので、Linuxでのプログラミングのノウハウ、RTLinuxの詳細、そしてPCIバスに関することなどはそれらを読んで理解してください。

以上簡単に、LinuxによるFAネットワークの実施例を説明してきました。紙面の都合により今回はこれで終了しますが、機会がありましたらさらに詳しい説明、とくにRTLinuxの使い方についての説明ができたらと思います。

R E S O U R C E

●Linuxでのプログラミング一般に関するもの
（特にTCPパケット通信、FIFO、プロセスの起動、システムコールなどで参考にしたもの）

- 「Linuxプログラミング」
  Neil Maatthew、Richard Stones共著／葛西重夫訳／ソフトバンクパブリッシング発行／ISBN 4-7973-0819-2／4200円
- 「プログラミングLinux」
  Michael K. johnson、Eric W.Troan共著／株式会社クイック訳／アスキー発行／ISBN 4-7561-2044-X／5800円
- 「Linuxシステムコール」
  塚越一雄著／技術評論社発行／ISBN 4-7741-1031-0／2480円

●GTK+/GDKに関するもの

- 「GTK+/GDKによるLinuxアプリケーション開発」
  Eric Harlow著／アンク訳／アンク監修／翔泳社発行／ISBN4-88135-775-1／3400円
- 「GTK+とGladeで作るLinuxプログラミング超入門」
  鈴木哲也 著／すばる舎発行
- 「GTK+入門　基礎から始めるXプログラミング」
  田中ひろゆき著／ソフトバンクパブリッシング発行／ISBN 4-7973-0871-0／2600円

●RTLinuxに関するもの

- 「Linuxリアルタイム計測／制御開発ガイドブック」
  船木陸議、羅正華共著／秀和システム発行／ISBN 4-87966-849-4／4200円
- 「リアルタイムOSとしてのLinux活用法」
  インターフェース1999年11月号／CQ出版発行
- 「開発者のためのUNIX/Linux入門」
  インターフェース2000年4月号／CQ出版発行

●PCIバスに関するもの

- 「PCI/CompactPCIバスの概要と応用」
  インターフェース1997年3月号／CQ出版発行
- 「PCIデバイス設計入門」
  TECH I Vol.3／CQ出版発行／ISBN 4-7898-3314-3／2200円
- 「PCIバスの詳細と応用へのステップ」
  OpenDesign No.7／CQ出版発行／ISBN 4-7898-3530-8／1835円