ＵＮＩＸにおける教育用ウィンドウシステムの実現

雪田修一

要　　旨

ＵＮＩＸには優れたウィンドウシステムが既にいくつも商品化されている。それらは，やがては大衆的に普及して行くであろうが，現時点ではまだ高価なハードウェア（ＵＮＩＸが搭載可能で高度なグラフィック機能を持つワークステーション）を必要とする。

我々は現時点で教育用ウィンドウシステムを大衆的に利用可能なものとして実現したい。そのため文字端末，つまりＵＮＩＸマシン自体ではなく文字ベースの端末エミュレータを走らせることのできるＰＣおよびＤＯＳの利用を前提としたシステム開発のプランをもった。とくに，入力デヴァイスとしてはキーボードのみを仮定する。それは，通信機能をもつワープロ専用機でも教育用ウィンドウシステムを利用可能にしたいからである。ホストマシンに採用したＳＵＮワークステーションではＢＳＤ系とシステムＶ系の両方の機能が使えるのが特徴である。この論文ではｃｕｒｓｅｓライブラリ，疑似端末，ソケット，そして日本語環境を助けるストリーム機構を駆使したウィンドウマネジャの実現方法を紹介し，教育用システムへの応用を展望する。

目　　次

1. ウィンドウマネジャと入出力の多重化

ウィンドウシステムという言葉で次のような状況を想像して頂きたい。

端末画面上には問題を提示するウィンドウ，答えを入力するウィンドウ，シェルの走っている端末エミュレータウィンドウなどが重なりも許しつつ展開されている。これら多数のウィンドウに対応したホスト上のＵＮＩＸプロセスの入出力を画面上のウィンドウにどのように反映させていくか，特に入力フォーカスの割当て，ウィンドウの重なり合いの管理を担当するプロセスが必要になってくる，この管理プロセスを一般にウィンドウマネジャと呼ぶ。

入力デヴァイスとしてはキーボードのみを仮定する。通信機能をもつワープロ専用機でも教育用ウィンドウシステムを利用可能にしたいので，マウスの利用は考えない。そのためウィンドウマネジャがいくつかのコントロールキーを予約することになってしまい子プロセスとの通信に透過性が損なわれてしまった。例えば，ウィンドウ上の端末エミュレータでエディタなどを起動するとウ

ィンドウマネジャのコマンドとエディタのコマンドが衝突することも出てくる。これはエディタなどのカスタマイズで急場をしのぐことにする。

個々のウィンドウで走るプログラムはウィンドウマネジャのサブルーチンの場合もあれば”fork”された子プロセスの場合もある。前者の場合，ウィンドウマネジャとの通信は同期的（予期した時点で入出力がある）であるが，後者の場合は非同期的（勝手なタイミングで入出力が行なわれる）になる。”多重化”の問題はこの非同期の場合に生じる。

以下ではウィンドウマネジャとその複数の子プロセスの通信の多重化を論ずる。

(1) パイプによる子シェルとの通信

パイプはプロセス間通信の古典的なテクニックである。以下の例では２つの子シェルをｆｏｒｋして通信の多重化を行なう。実行時に子プロセスを生成，消滅させるのは簡単な練習問題であるから，興味のある方は試みられたい。

ウィンドウマネジャのソースコードの主要部分

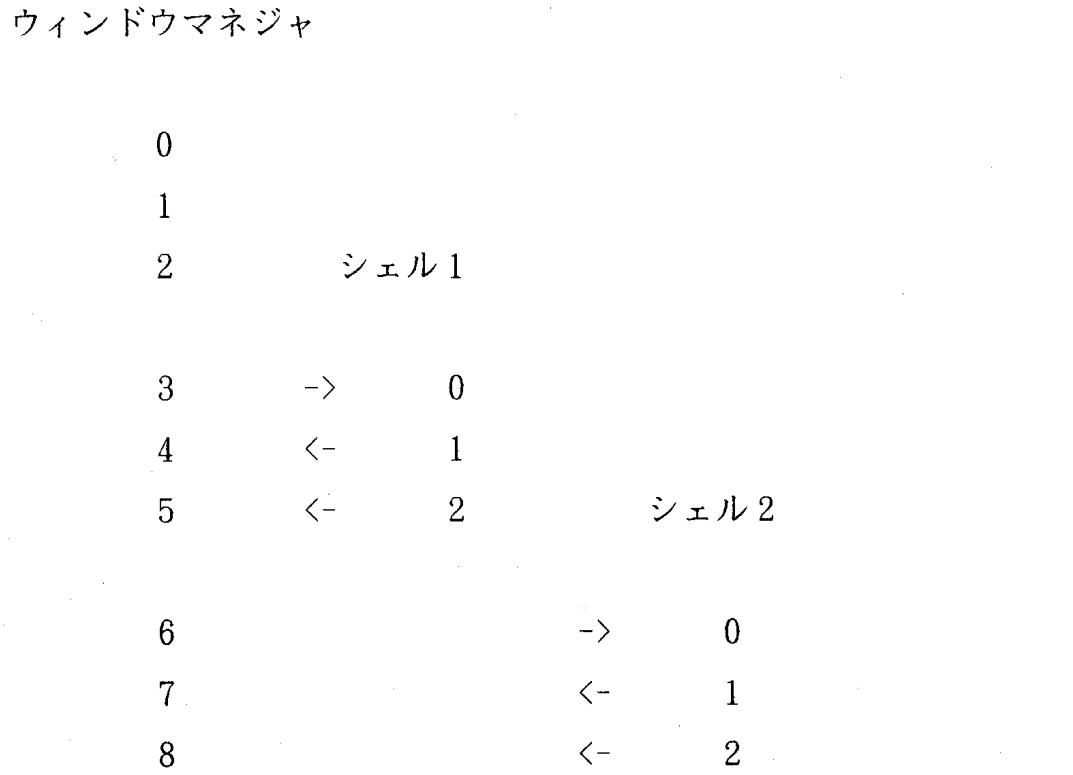
shell tool　構造体は子プロセス毎の管理情報を保持する。

```
struct  shell tool
        {
        int     id;
        WINDOW  *win,*frame;
        char    *input buf;
        int     len;
        }
        w 2 ;

main()
        {
```

```
        int     fd0 2 , fd1 2 , fd2 2 ;
        int     id;             /* プロセスｉｄ */
        int     c, len;
        int     sw;

/*  set up 1st shell */

        pipe(fd0);pipe(fd1);pipe(fd2);
        if((id = fork())== 0)
                {
                close(0);close(1);close(2);close(4);close(5);close(7);
                dup(3);dup(6);dup(8);
                close(3);close(6);close(8);
                execl("/usr/bin/csh","csh","-i",0);
                }
        else    /* winman's code */
                {
                w 0 . id = id;
                close(3);close(6);close(8);
                dup(4);close(4);dup(5);close(5);dup(7);close(7);
                }

/*  set up 2nd shell  */

        pipe(fd0);pipe(fd1);pipe(fd2);
        if((id = fork())== 0)
                {
                close(0);close(1);close(2);close(3);close(4);close(5);
```

```
        close(7);close(8);close(10);
        dup(6);dup(9);dup(11);close(6);close(9);close(11);
        execl("/usr/bin/csh","csh","-i",0);
        }
else    /* winman's code */
        {
        w1.id = id;
        close(6);close(9);close(11);dup(7);close(7);
        dup(8);close(8);dup(10);close(10);
        }
```

ここまでのコードで次のような通信路が作られる。数字は各プロセスのファイルディスクリプタで，矢印の起点はｗｒｉｔｅ，終点はｒｅａｄする側を表す。



子プロセスであるシェル１，シェル２のファイルディスクリプタ０，１，２はそれぞれの標準入力，標準出力，警告出力になっている。この方法だと合計４つの子プロセスの入出力を管理することができる。ウィンドウマネジャも１つのプロセスであるから２０個のファイルディスクリプタが許されている。

それならば５つまで可能ではないかという疑問が出てもよいが，実際は中間状態で２０の制限を突破してしまうのである。

次は処理のループに入る。

```
while(1)
        {
        readOk = mask;
        select(32, &readOk, NULL, NULL, NULL);
        if(readOk & 1)
                }
                c = wgetch(w sw .win);
                switch(c)
                        {
                        キーボードからの入力をウィンドウマネジ
                        ャ，子プロセスの間で分配する。
                        }
                  {

        if(readOk & (1 << 4) )
                シェル１の標準出力をウィンドウ０に書き出す。
        if(readOk & (1 << 5) )
                シェル１の警告出力をウィンドウ０に書き出す。
        if(readOk & (1 << 7) )
```

```
                シェル２の標準出力をウィンドウ１に書き出す。
        if(readOk & (1 << 8) )
                シェル２の警告出力をウィンドウ１に書き出す。
        }
```

以上が，ウィンドウマネジャの骨組みである。

この方法では端末ドライバをウィンドウマネジャのみが利用しているのでラインエディット機能はウィンドウマネジャの責任になる。詳しくは Appendix A を参照して欲しい。更に付け加えるとウィンドウ上で端末ドライバに依存するプログラムを起動することはできない。ここら辺がこの方法の弱点である。

ｅｄのようなエディタを使うことになる。

(2) 疑似端末の利用

単純にパイプを利用するだけでは前述したような問題点がある。

デヴァイスとファイル入出力をバイトストリームとして統一して捉えるＵＮＩＸの美点が崩れている部分が端末の取り扱いなのである。この節ではパイプのような単純で魅力的な方法を一旦捨てて端末ドライバと格闘することにしよう。

疑似端末は普通の端末と同様にファイルシステム内に

```
/dev/ttyp0
/dev/ptyp0
```

のようなデヴァイススペシャルファイルのペアとして存在している。

ｔｔｙはスレイヴデヴァイスと呼ばれ通常のｉｏｃｔｌ，ｒｅａｄ，ｗｒｉｔｅシステムコールでアクセスする。一方，ｐｔｙはコントローラデヴァイスと呼ばれ，ｔｔｙのｒｅａｄ，ｗｒｉｔｅと裏表のｗｒｉｔｅ，ｒｅａｄでｔｔｙをモニタすることになる。ｔｔｙとｐｔｙは異なるプロセスに所有され，ｔｔｙ側は普通に端末にアクセスする。ｐｔｙ側はデヴァイスをシミュレートしなければならない。

では，実例を挙げて解説しよう。ウィンドウ（子プロセス）は簡単のため1つで示す。

```
char    recv buf BLOCK+1 ;

main()
        {
        int     fd, id;
        int     mask, readOk;
        int     c;

        if((id = fork())== 0)
                }
                /* 子プロセスのコードの始まり */
                close(0);close(1);close(2);
                setpgrp();
                open("/dev/ttyrf",0 RDONLY);
                open("/dev/ttyrf",0 WRONLY);
                open("/dev/ttyrf",0 WRONLY);
                execl("/usr/bin/csh","csh","-i",0);
                }
        else
                {
                /* ウィンドウマネジャのコードの始まり */
                fd=open("/dev/ptyrf",0 RDWR);

                ここでウィンドウの初期化
                ......................
```

```
        mask =( 1<<(fd+1) ) - 1;
        while(1)
                {
                readOk = mask;
                select(32,&readOk,NULL,NULL,NULL);
                if(readOk & (1<<fd) )
                        ｐｔｙからデータを受取り
                        ウィンドウに書き出す。
                if(readOk & 1)
                        キーボードからデータを受取り
                        ｐｔｙに送り出す。
                }
        }
}
```

親プロセスであるウィンドウマネジャはｐｔｙをオープンし，子プロセスはｔｔｙを３回（標準入力，標準出力，警告出力）オープンしている。

ｐｔｙとｔｔｙの間のデータ交換は全二重になっており，親プロセス側は１つのファイルディスクリプタで１つの子プロセスを管理する。従って，パイプの場合と違ってこの方法は最大限１７個の子プロセスを１つのウィンドウマネジャが管理することになる。

子プロセスは親プロセスと異なるプロセスグループのリーダプロセスになることもこの方法の大きな特徴である。ｓｅｔｐｇｒｐシステムコールに注意されたい。ジョブ制御などもウィンドウ毎に可能になる。

上の例では子プロセスはｃｓｈをｅｘｅｃしているが Appendix B. に示すようにｌｏｇｉｎをｅｘｅｃして，ウィンドウ毎にログインユーザを割り当てることもできる。このようにすることの利点は複数ユーザ間の通信プログラムの

開発環境を実現できることである。また，この疑似端末のプログラム例ではｔｅｒｍｉｎｆｏレベルの機能の実現はしていないが近々実現する見通しなので次の報告に回す。

2. 日本語環境の実現

(1) パイプの場合

前述の通りウィンドウマネジャがラインディシプリンを実行するので，ウィンドウマネジャの支配端末ストリームに日本語変換モジュールがｐｕｓｈされていればよい。ｇｅｔｃｈのようなｃｕｒｓｅｓライブラリ関数は処理コードを扱うことができるのでキーボードからの入力イヴェントは複数バイト文字の形態で発生させることができる。

(2) 疑似端末の利用

日本語環境はカーネルレベルで対応しているので，子プロセスがｔｔｙストリームをオープンすると，自動的にＥＵＣラインディシプリンモジュールであるｌｄｔｅｒｍ，ストリーム以前との互換性をとるｔｔｃｏｍｐａｔモジュールがｐｕｓｈされる。端末の特性は多くのパラメタで指定する必要があり，プログラム例でも解かるとおりｉｏｃｔｌをうんざりするほど使うことになる。

3. 教育用システムへの応用

(1) 教卓プロセスと生徒プロセス

ウィンドウマネジャがあらゆるイヴェントを集中的に管理するので学習者が利用中のウィンドウとデータをやり取りするだけでなく，教卓プロセスとのデータ交換を同時に行なうことができる。教育用システムは一斉授業にも，分散した独習援助にもいずれにも対応できなくてはならない。教卓プロセスは無人

運転，有人運転の２つのモードを持つことになる。生徒プロセスは学習者の利用する　ウィンドウマネジャそのものである。教卓プロセスと生徒プロセスのデータ　交換にはサーバ，クライアントモデルがよくあてはまる。

(2) ソケットか共有メモリか？

学習者の端末エミュレータを単にモニタするにはソケットによるコネクションを教卓との間で確立するのが簡単である。しかし，スクリーンエディタなどを使用中の学習者にエディタ画面上で教師が介入するのはソケットのような手段で十分か疑わしい。教卓プロセスが共有メモリでウィンドウデータを直接アクセスすることも検討していく必要があろう。

Ａｐｐｅｎｄｉｘ　Ａ.　　パイプを利用したウィンドウマネジャ

```
#include <locale.h>
#include <curses.h>
#include <signal.h>
#define MAXFD   8
#define CR 10
#define SW  1
#define EOT 4
#define BS 127
#define BLOCK 128
/*  block size is critical */

char    send buf 2  60 ;
char    recv buf BLOCK+1 ;
int     command size, data size;

int     mask = (1 << (MAXFD + 1) ) - 1;
int     readOk;
int     die();

struct  shell tool
        {
        int     id;
        WINDOW  *win,*frame;
        char    *input buf;
        int     len;
```

```
        }
        w 2 ;

main()
        {
        int     fd0 2 , fd1 2 , fd2 2 ;
        int     id;
        int     c, len;
        int     sw;

        setlocale(LC ALL,"");

/*  set up the  1st shell */
        pipe(fd0); pipe(fd1); pipe(fd2);
        id = fork();
        if(id == -1)
                {
                fprintf(stderr,"fork failed for the 1st window.¥n");
                exit(1);
               }

        if(id == 0)
                {
                close(0);close(1);close(2);close(4);close(5);close(7);
                dup(3);dup(6);dup(8);close(3);close(6);close(8);
                if( execl("/usr/bin/csh","csh","-i",0)== -1)
                        {
                        fprintf(stderr,"The 1st shell cannot run.¥n");
```

```
                        exit(1);
                        }
                }
        else
                {
                w0.id = id;
                close(3);close(6);close(8);dup(4);close(4);
                dup(5);close(5);dup(7);close(7);
                }

/*  set up the  2nd shell  */

        pipe(fd0);
        pipe(fd1);
        pipe(fd2);
        id = fork();
        if(id == -1)
                {
                fprintf(stderr,"fork failed for the  2nd  window.¥n");
                exit(1);
                }
        if(id == 0)
                {
                close(0);close(1);close(2);close(3);close(4);
                close(5);close(7);close(8);close(10);
                dup(6);dup(9);dup(11);
                close(6);close(9);close(11);
                if( execl("/usr/bin/csh","csh","-i",0)== -1)
```

```
                {
                fprintf(stderr,"The 2nd shell cannot run.¥n");
                exit(1);
                }
        }
else
        {
        w 1 .id = id;
        close(6);close(9);close(11);dup(7);close(7);
        dup(8);close(8);dup(10);close(10);
        }

initscr();

w 1 .win=newwin(20, 60, 1, 1);
w 1 .frame=newwin(22, 62, 0, 0);
box(w 1 .frame,' ','-');
wrefresh(w 1 .frame);

w 0 .win=newwin(20, 60, 3, 5);
w 0 .frame=newwin(22, 62, 2, 4);
box(w 0 .frame,' ','-');
wrefresh(w 0 .frame);

signal(SIGINT, die); signal(SIGQUIT, die);
signal(SIGBUS, die); signal(SIGPIPE, die);
signal(SIGTERM, die); signal(SIGSEGV, die);
noecho(); crmode();
```

```
        scrollok(w 1 .win, TRUE);
        scrollok(w 0 .win, TRUE);

        w 1 .input buf=send buf 1 ;
        w 0 .input buf=send buf 0 ;
        w 1 .len=0; w 0 .len=0;
        sw=0;

        while(1)
                {
                readOk = mask;
                select(32, &readOk, NULL, NULL, NULL);
                if(readOk & 1)
                        {
                        c = wgetch(w sw .win);
                        switch(c)
                                {
                                case '¥n':
                                        w sw .input buf w sw .len ='¥n

                                        if(sw==1)
                                        write(3, w sw .input buf, w sw .
len+1);
                                        if(sw==0)
                                        write(6, w sw .input buf, w sw .
len+1);
                                       waddstr(w sw .win, "¥n");
                                        w sw .len=0;
```

```
            break;
        case BS:
            if(w sw . len) w sw . len--;
            wmoveprevch(w sw . win);
            wdelch(w sw . win);
            wrefresh(w sw . win);
            break;
        case SW:
            if(w (sw+1)%2 . id == -1)
                    break;
            sw=(sw+1) % 2;
            touchwin(w sw . frame);
            wrefresh(w sw . frame);
            touchwin(w sw . win);
            wrefresh(w sw . win);
            break;
        case EOT:
            werase(w sw . win);
            werase(w sw . frame);
            wrefresh(w sw . win);
            wrefresh(w sw . frame);
            delwin(w sw . win);
            delwin(w sw . frame);
            kill(w sw . id);
            w sw . id = -1;
            sw=(sw+1) % 2;
            if(w sw . id == -1) die();
            touchwin(w sw . frame);
```

```
                                wrefresh(w sw .frame);
                                touchwin(w sw .win);
                                wrefresh(w sw .win);
                                break;
                        default:
                                w sw .input buf w sw .len =(ch
ar)c;
                                waddch(w sw .win, c);
                                wrefresh(w sw .win);
                                w sw .len++;
                                break;
                        }
                }


        if(readOk & (1 << 4) )
                {
                data size=read(4, recv buf, BLOCK);
                recv buf data size ='¥0';
                waddstr(w 1 .win, recv buf);
                if(sw==1)
                        wrefresh(w sw .win);
                }
        if(readOk & (1 << 5) )
                {
                data size=read(5, recv buf, BLOCK);
                recv buf data size ='¥0';
                waddstr(w 1 .win, recv buf);
```

```
                    if(sw==1)
                        wrefresh(w sw .win);
                    }

                if(readOk & (1 << 7) )
                    {
                    data size=read(7,recv buf,BLOCK);
                    recv buf data size ='¥0';
                    waddstr(w 0 .win,recv buf);
                    if(sw==0)
                        wrefresh(w sw .win);
                    }
                if(readOk & (1 << 8) )
                    {
                    data size=read(8,recv buf,BLOCK);
                    recv buf data size ='¥0';
                    waddstr(w 0 .win,recv buf);
                    if(sw==0)
                        wrefresh(w sw .win);
                    }
                }
        }

int     die()
        {
        endwin();
        exit(1);
        }
```

Ａｐｐｅｎｄｉｘ　Ｂ．　疑似端末の使用例

```
/*************************************************************/
/*    日本語対応シェルウィンドウ      ＥＵＣ端末として実現   */
/*************************************************************/
#include <curses.h>
#include <locale.h>
#include <widec.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stream.h>
#include <sys/stropts.h>
#include <sys/termios.h>
#include <signal.h>
#define BLOCK 128    /* ブロックサイズを大きく取ると暴走する。   */

char     recv buf BLOCK+1 ;
/*
wchar t w buf BLOCK+1 ;
*/
int      die();
int      die2();

main()
        {
        int      fd, id;
        int      mask, readOk, datalen;
```

```
wchar t c;
char    send data 4 ;
char    *login name;
int     i, cnt, len;
WINDOW  *win, *frame;
struct  termios tty info;
struct  winsize dim;

setlocale(LC ALL,"");
id = fork();
if(id == 0)
        {
        close(0);close(1);close(2);
        setpgrp();
       open("/dev/ttyrf",0 RDONLY);
        open("/dev/ttyrf",0 WRONLY);
        open("/dev/ttyrf",0 WRONLY);


        ioctl(0, TCGETS, &tty info );
        tty info.c iflag &=  ISTRIP &  INPCK &  PARMRK &  IGNP
AR;
        tty info.c cflag  = CS8;
        tty info.c cflag &=  PARENB;
        tty info.c oflag &=  ONLCR;
        tty info.c lflag  = ECHOE;
        ioctl(0, TCSETS, &tty info );
```

```
                dim.ws_row = (unsigned short)20;
                dim.ws_col = (unsigned short)60;
                ioctl(0, TIOCSWINSZ, &dim);

                putenv("WIN=ON");
                putenv("WINNUM=1");

/*              login_name = (char *)getenv("LOGNAME");
*/
/*              execl("/bin/login","login","-p",login_name,(char *)0);
 */
                execl("/bin/csh","csh","-i",0);
                }
        else
                {
                fd=open("/dev/ptyrf",0_RDWR);
                initscr();
                win=newwin(20,60,1,10); frame=newwin(22,62,0,9);
                box(frame,' ','-'); wrefresh(frame);
                noecho(); crmode(); scrollok(win,TRUE);

                signal(SIGINT,die); signal(SIGSEGV,die);
                signal(SIGQUIT,die); signal(SIGCLD,die2);

                mask =( 1<<(fd+1) ) - 1;
                while(1)
                        {
                        readOk = mask;
```

```
                    select(32, &readOk, NULL, NULL, NULL);

                    if(readOk & (1<<fd) )
                            {
                            datalen = read(fd, recv buf, BLOCK);
                            recv buf datalen =' ¥0' ;
                            waddstr(win, recv buf);
                            wrefresh(win);
                            }

                    if(readOk & 1)
                            {
                            c = wgetch(win);
                            sprintf(send data, "%wc", c);
                            len=strlen(send data);
                            write(fd, send data, len);
                            }
                    }
            }
        die();
        }


int     die()
        {
        endwin();
        exit(1);
        }
```

```
int     die2()
        {
        endwin();
        fprintf(stderr,"Child died.¥n");
        exit(1);
        }
```

Refrences

1　Ｓｕｎワークステーションマニュアル

2　ＵＮＩＸ原典,　パーソナルメディア

3　The Design Of The UNIX OPERATIONG SYSTEM, M.J. Bach