

ファイル入出力と プロセス間通信 (2)

2004年12月16日

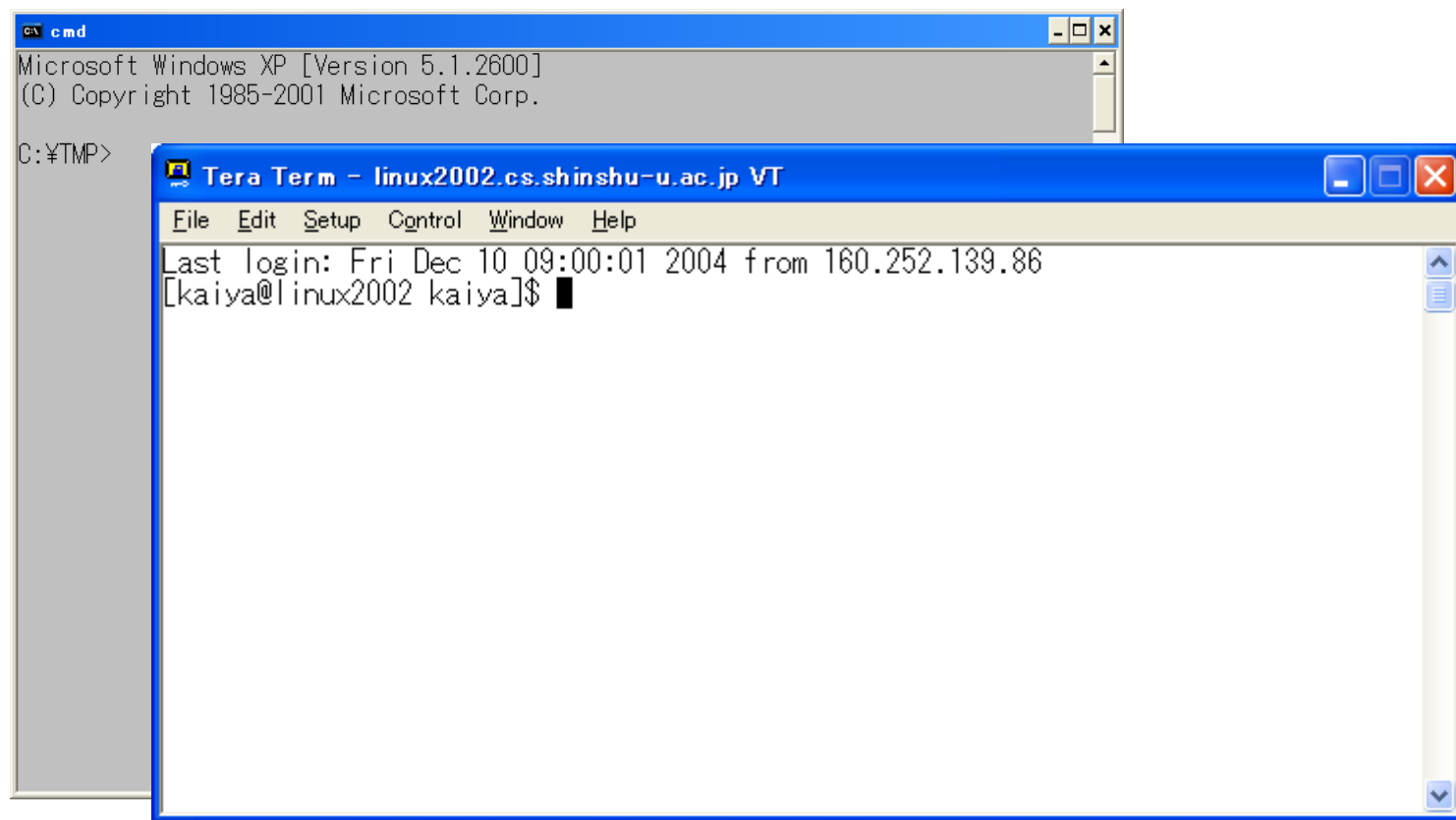
海谷 治彦

目次

- ターミナルとコンソール (tty)
 - デバイスドライバへの伏線
- リダイレクションの実装
- パイプ
- パイプによるプロセス間通信
- 簡易なプロセス間通信

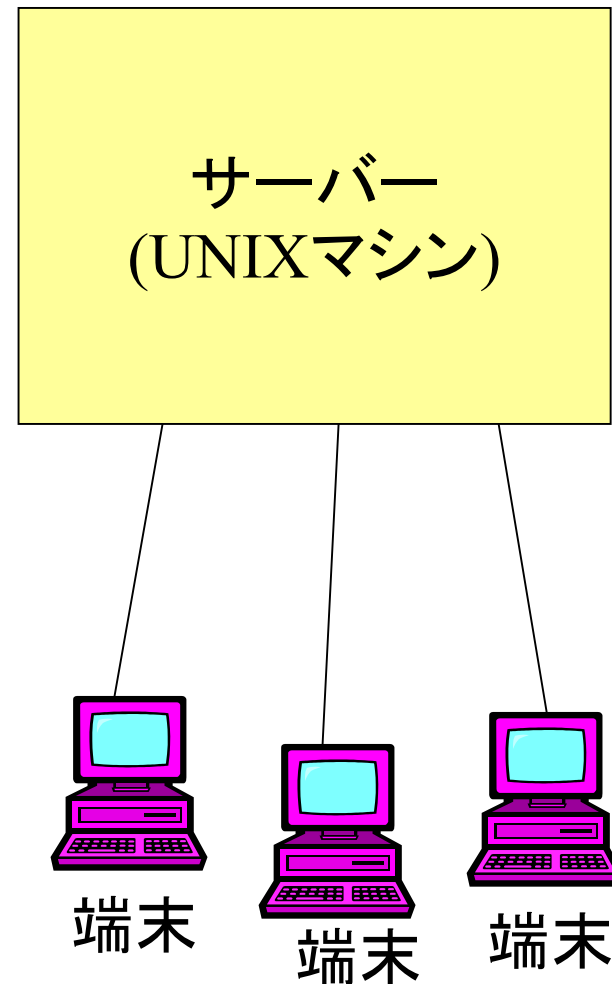
ターミナル, コンソール

- 入門的なCプログラム等で必ず利用されるユーザーインタフェース.
- ちょっと使うのが退屈.



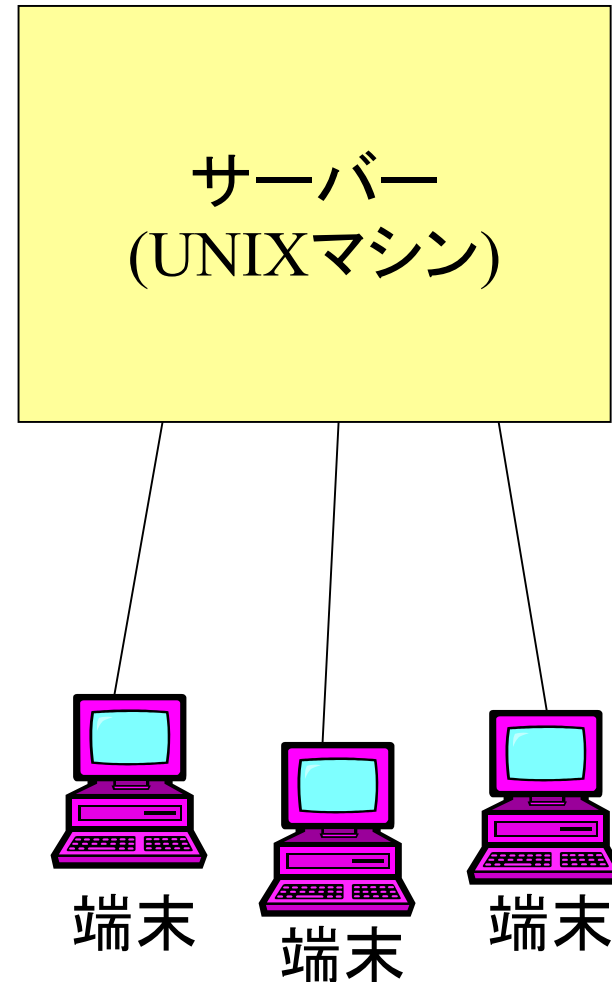
初期(～1990年)のUNIX環境(1)

- UNIXマシンは個人で占有せず,
- 端末装置(ターミナル)を使って, 共有利用した.
 - ソレ自体は頭の良くないコンピュータ, 通信機能くらいしかない.



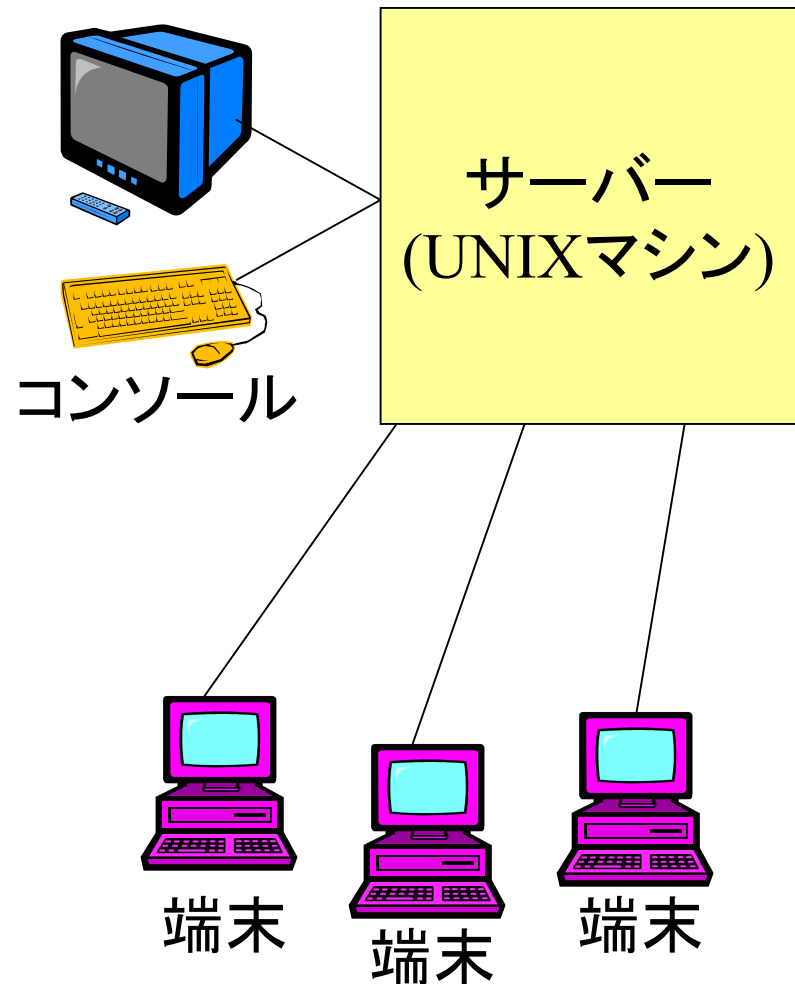
初期(～1990年)のUNIX環境(2)

- UNIXマシンと端末は、かなり遅い通信回線で結ばれていた。
 - RS232C等を用いたシリアル回線.
 - 9600bps程度.
 - 今のイーサが100Mbps (100,000,000bps)
- テキストを入力して、テキストで結果が返ってくる単純な入出力処理のみを行った。
 - グラフィック等は扱えない.



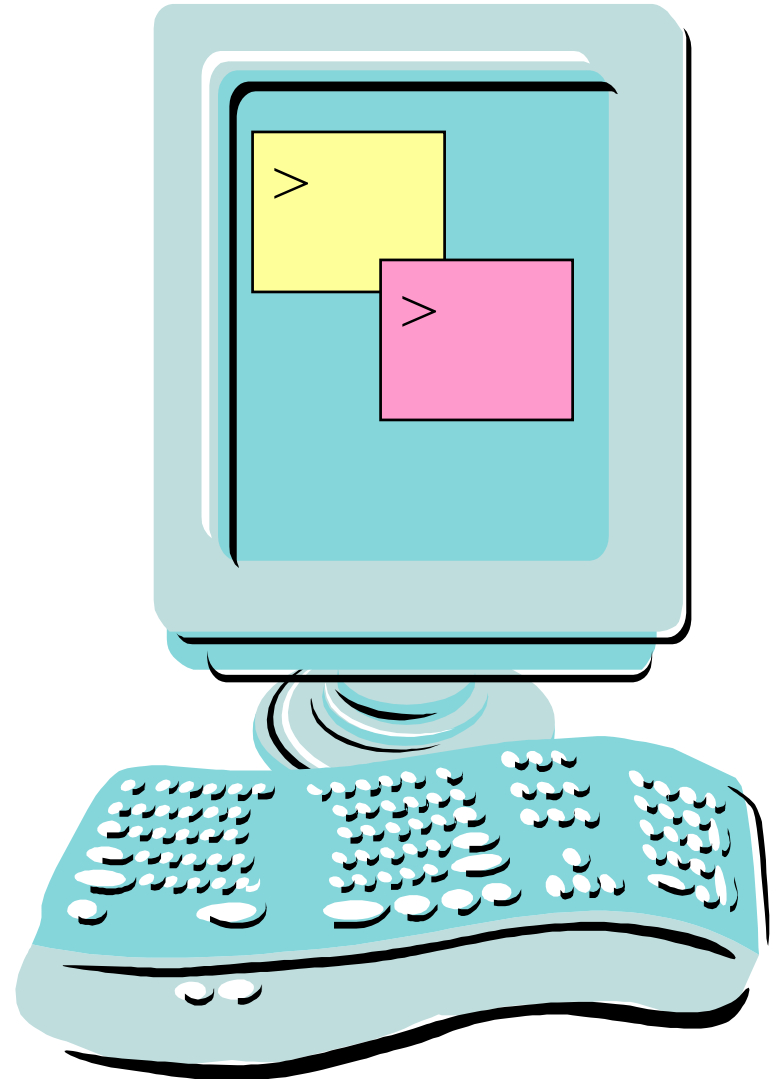
初期(～1990年)のUNIX環境(3)

- **コンソール**はマシンに直結されているモニタと入力装置を指す.
- セキュリティ上の理由等から特権的な処理(システムの停止等)はコンソールからでないとできないようになっている場合が多かった.



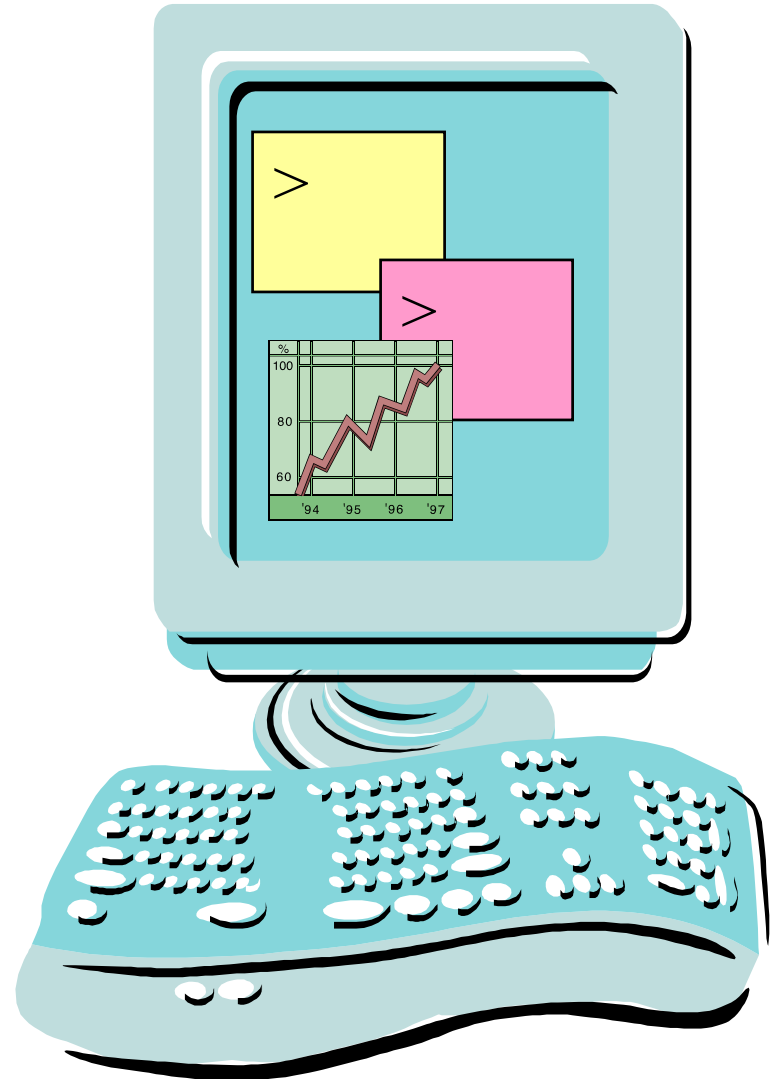
今のUNIX系OS環境(1)

- マシンを個人が占有するようになった。
 - マルチユーザー, マルチプロセスの意識が希薄になった.
- コンソールを直接使うようになった。
 - 初期にはコンソールは管理目的以外には使わないことが多かった.
- コンソールはマルチウィンドウシステムを採用するようになった.



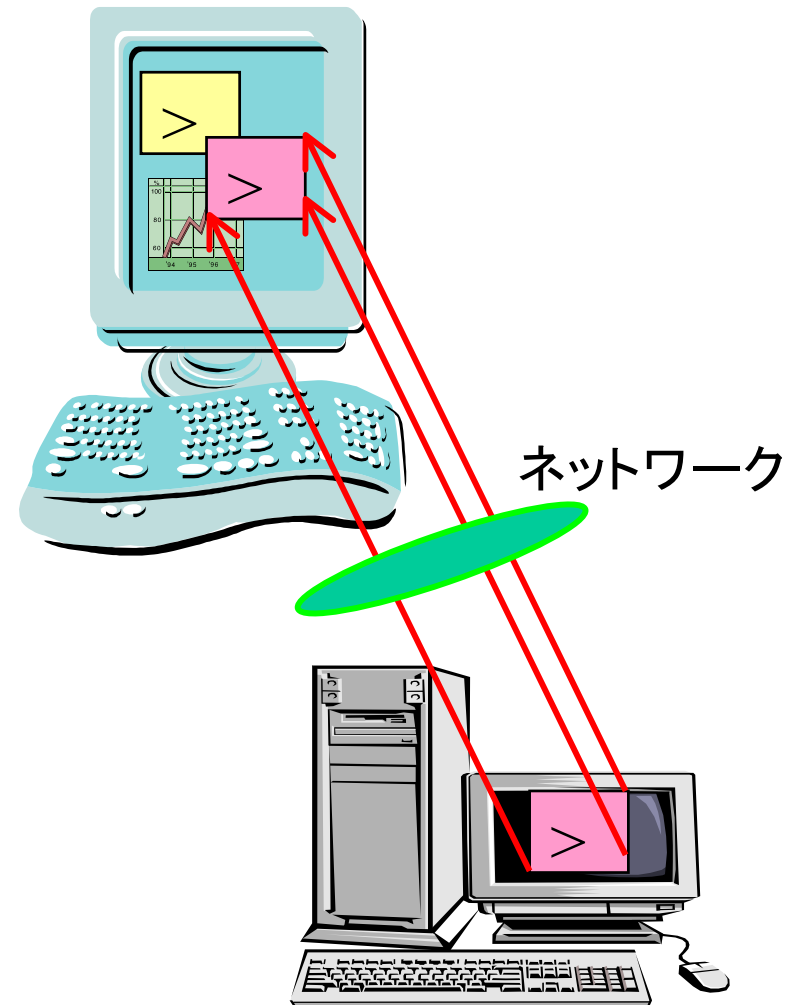
今のUNIX系OS環境(2)

- コンソール中に**仮想端末**
(**擬似端末 pseudo terminal, 普通ptyと呼ぶ**)
を作るようになった.
 - アプリケーションの1つとして動作している.
 - 昔の端末を模倣している.
- 無論, 端末的なWindowだけでなく, 表計算ソフト等, 多様なアプリも動く.



今のUNIX系OS環境(3)

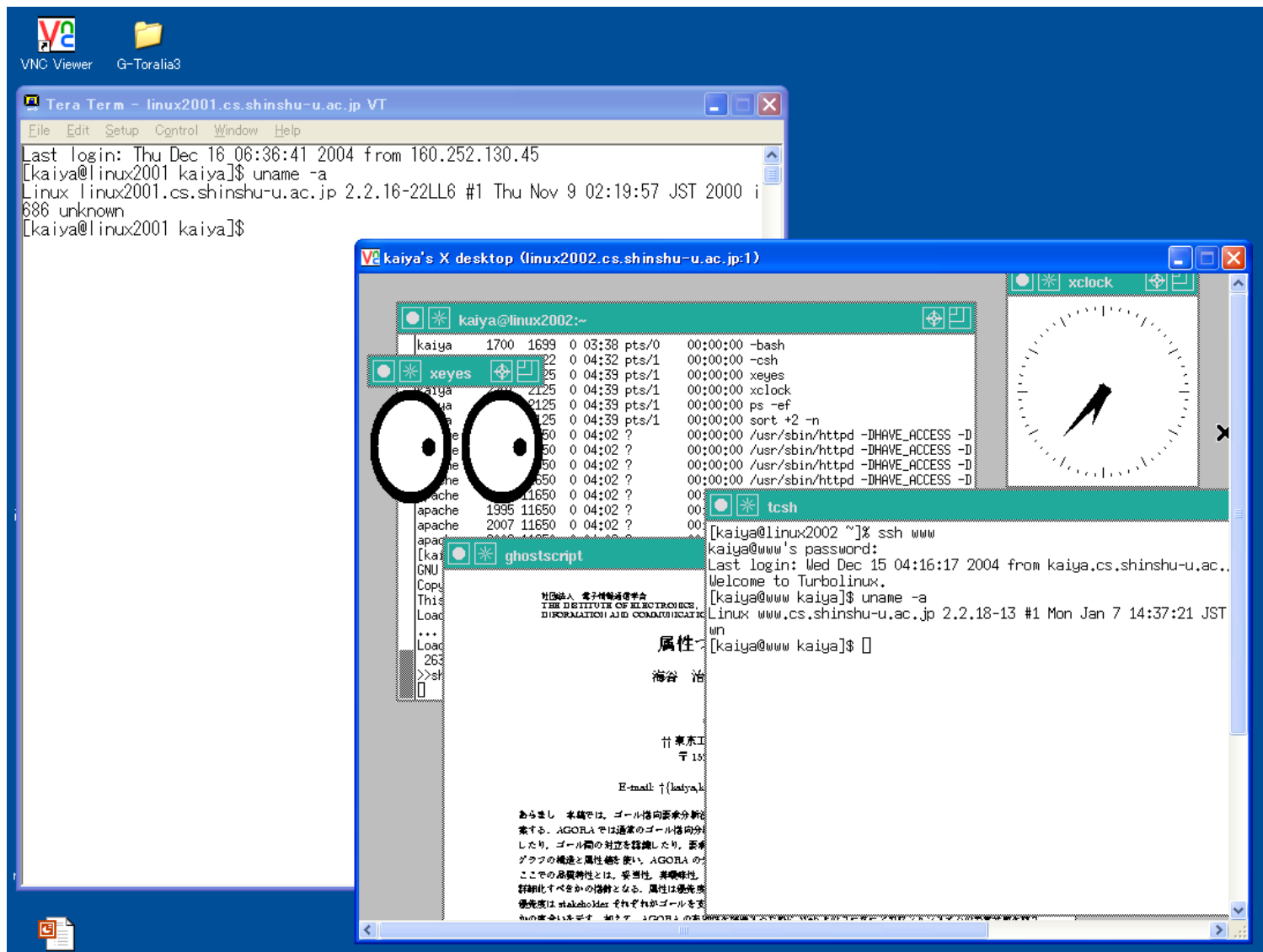
- ネットワーク経由で他のマシンのptyを呼び出すことが可能となっている.
 - 昔の端末みたい.
- 一般に他のマシンのサービスを利用する方法は現在では多数存在する.



他のマシンのサービスを利用

- 仮想端末を利用
 - telnet や rsh, ssh を利用する方法がコレ.
 - 旧来の端末利用を模倣している点で旧人類にも受け入れやすい.
- Windowシステムごと呼ぶ.
 - 他のマシンのWindowシステムの一部を呼び出す方法. VNCやMSも似たようなソフトを提供していたはず.
 - UNIX標準のX window システムはGUIをリモートで呼び出す機能を開発当初から備えていた.
 - Windowsはこの辺が弱い.
- 関数呼び出しレベルで他のマシンに処理を依頼.
 - RPCやRMIと呼ばれるリモート手続き呼び出しの類.
 - HTTP等に基づくウェブサービスも分類的にはコレ.
 - FTPクライアントもコレかな？微妙.

他マシン呼び出しの複雑な呼び出し例



前ページの構成

- ユーザーはWindows PCを利用
 - 擬似端末呼び出しソフト(Teraterm)でlinux2001の擬似端末を呼び出す.
 - VNCでlinux2002上のWindowシステムを呼び出す.
 - 目玉のアプリ (xeyes)
 - 時計 (xclock)
 - 文書表示 (ghostscript)
 - kterm (擬似端末)
 - kterm (擬似端末)
 - ココからさらにsshしてホストwwwの擬似端末を呼び出す.
- 以上, 前ページの画面には4つの異なるマシンのインタフェースが見えている.

(擬似)端末装置のOS的な説明

- 端末(モニタとキーボード)の制御はOSが行っている.
 - 詳細は次回だが, この制御を行うOSの部分をttyドライバ (デバイスドライバの一種)と呼ぶ.
 - ttyはTele Type writer が由来らしい.
- 端末で起動されるアプリからは,
 - キーボード ⇒ ファイルディスクリプタ0番
 - stdin
 - モニタ ⇒ ファイルディスクリプタ1と2番
 - stdout と stderrとして見えている.

端末の3つの特殊機能

通常のファイルI/Oとは異なり，端末の入出力では以下の3つを考慮しなければならない．

- エコーバック機能

- キーボードから入力された文字が画面に表示される．

- バッファ機能

- エンターを押すまで文字をアプリに送付しない．
- ファイルストリームのバッファとはまた違う(涙)

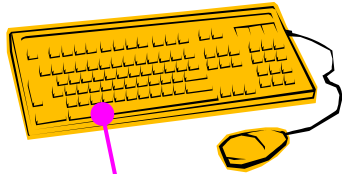
- シグナル機能

- 特定のキー入力により，アプリにシグナルを送ることができる．
 - 例: コントロールCで強制終了，コントロールZで中断等．

上記機能はシステムコール(ioctl)によって無効にすることもできる(が，かなり高度で危険なプログラミング)．

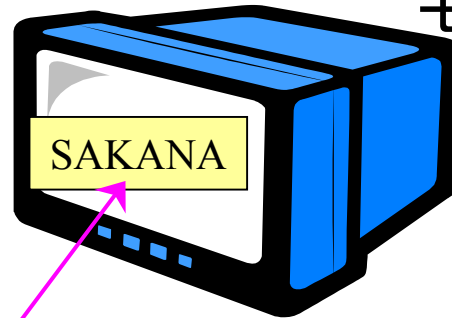
エコーバックとバッファの例

キーボード

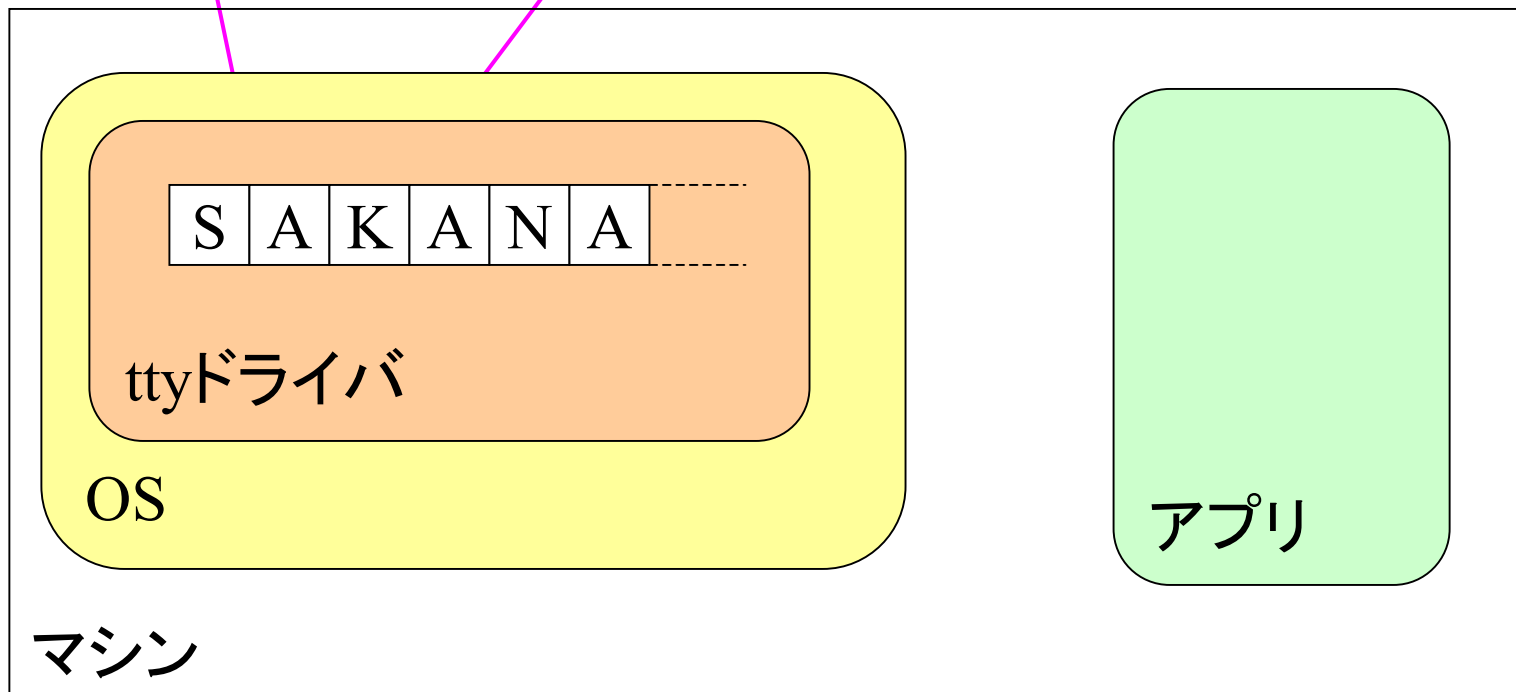


SAKANA

モニタ

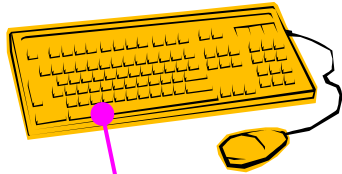


打ち込んだだけではアプリに伝わっていないが、画面には表示される。



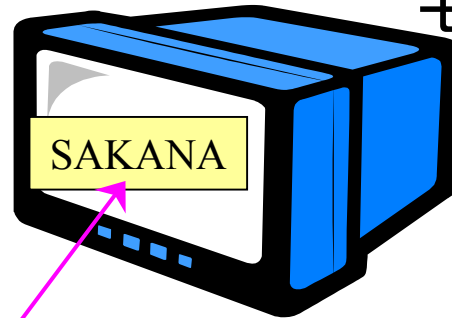
バッファの例

キーボード

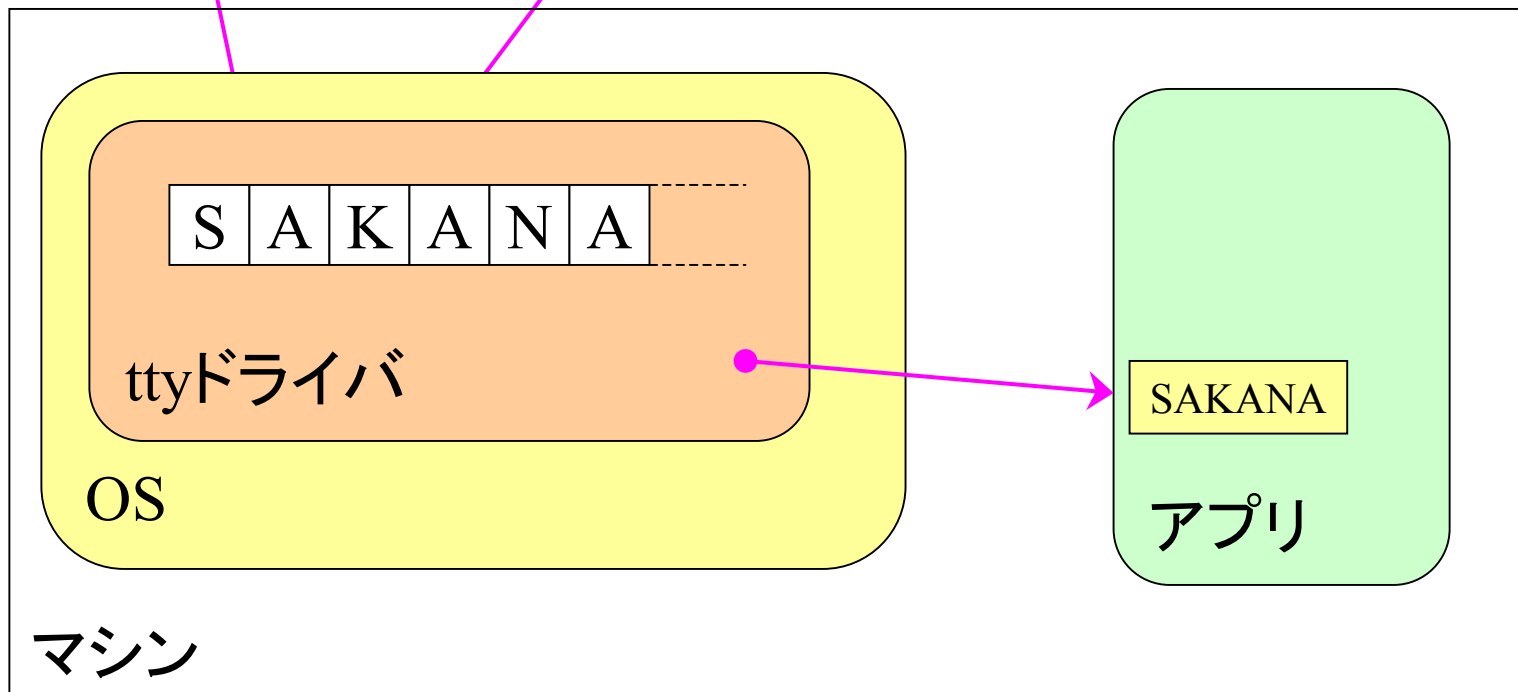


SAKANA
エンター

モニタ

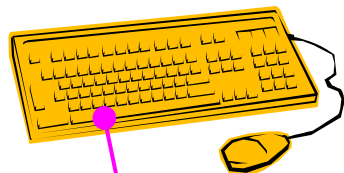


エンターを押すと、やっとアプリに伝わる。



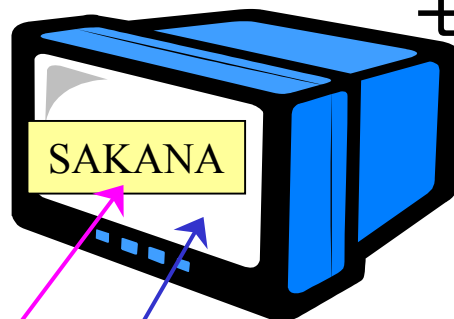
結果を標準出力に返すなら

キーボード

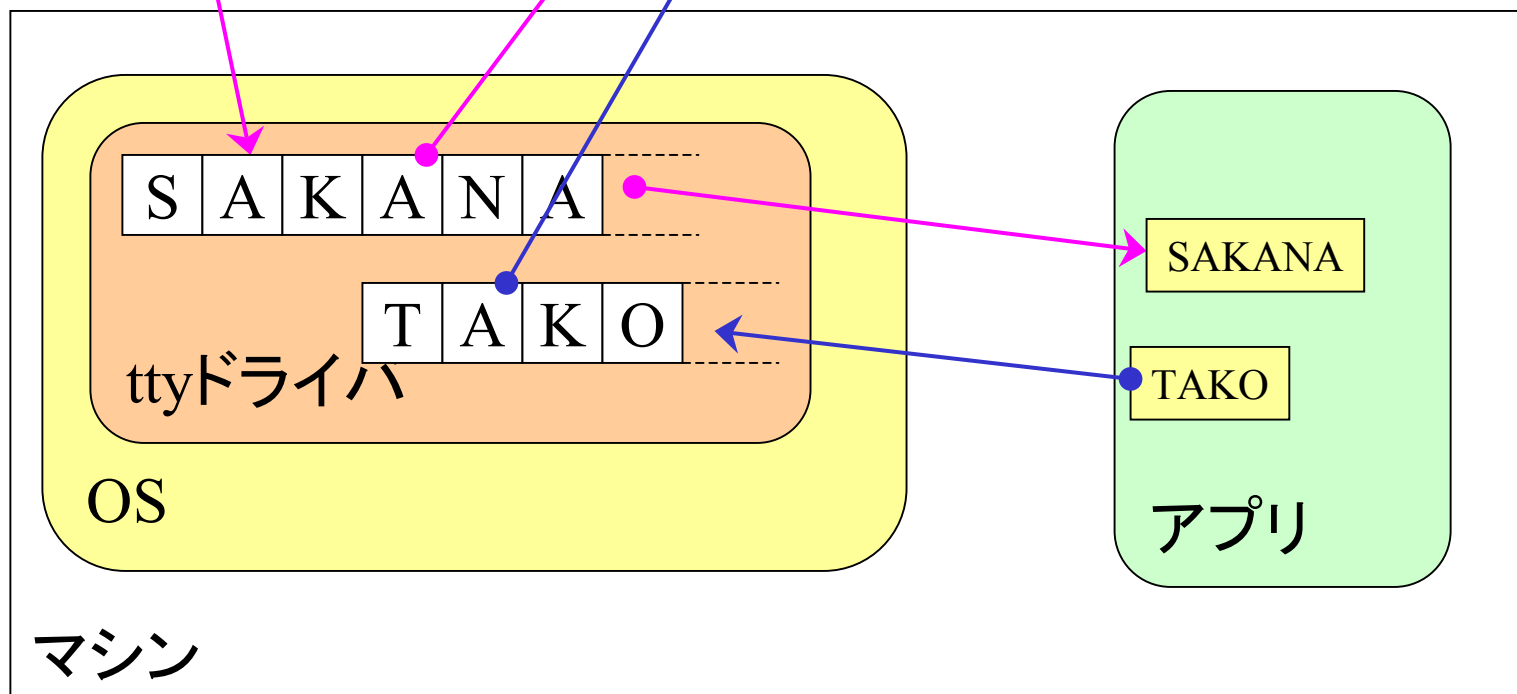


SAKANA
エンター

モニタ

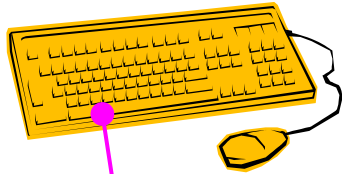


やはりttyドライ
バを経由して画
面に返す.



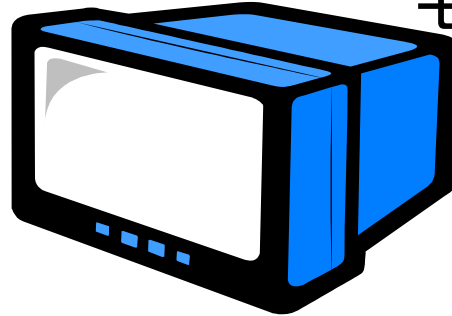
シグナルの例

キーボード

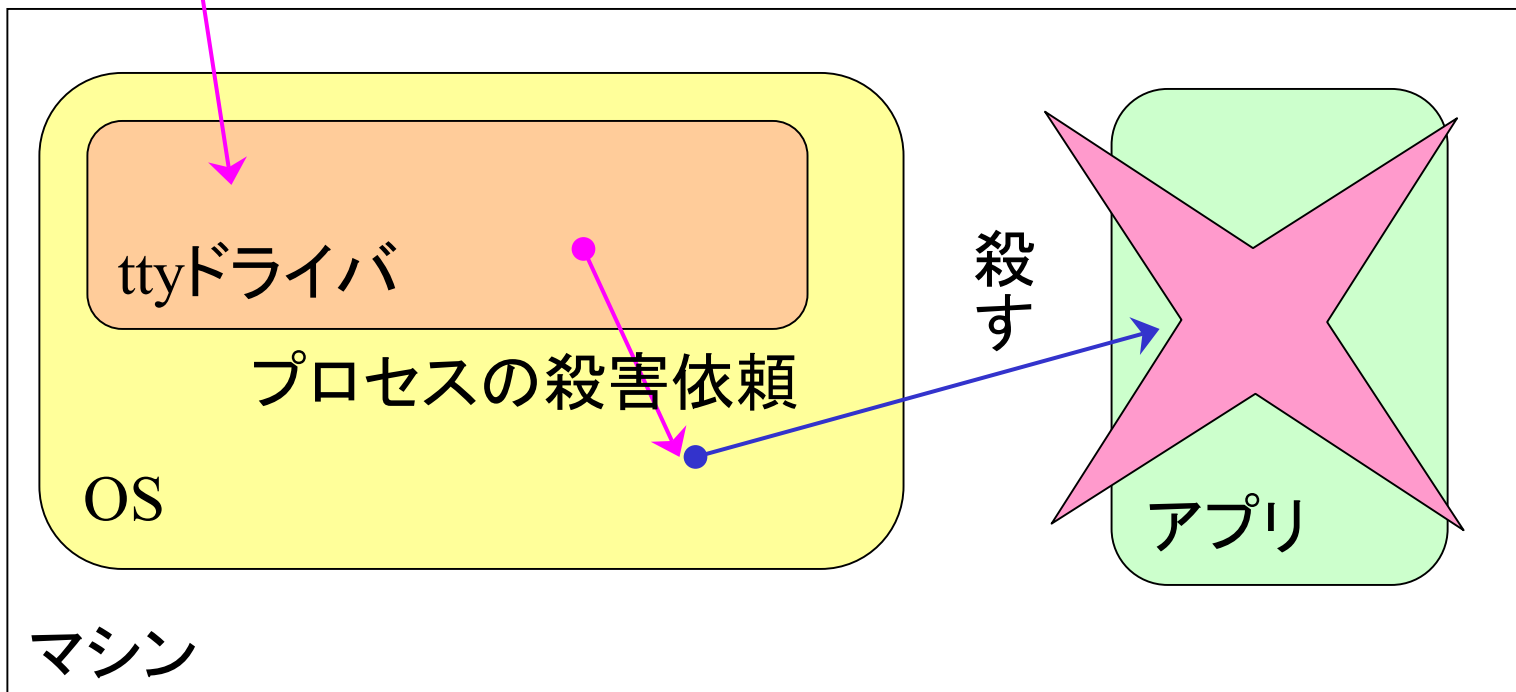


コントロールC

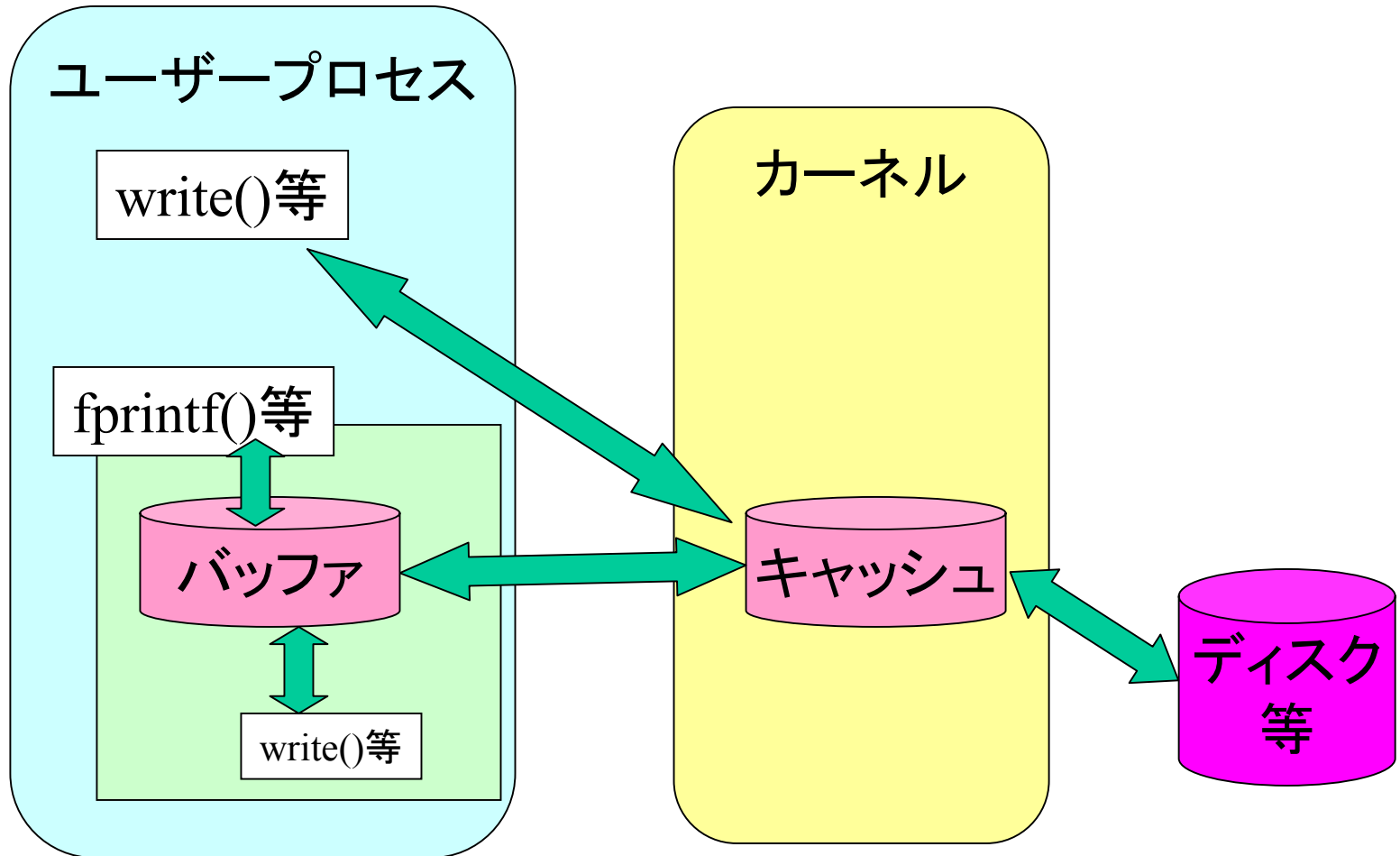
モニタ



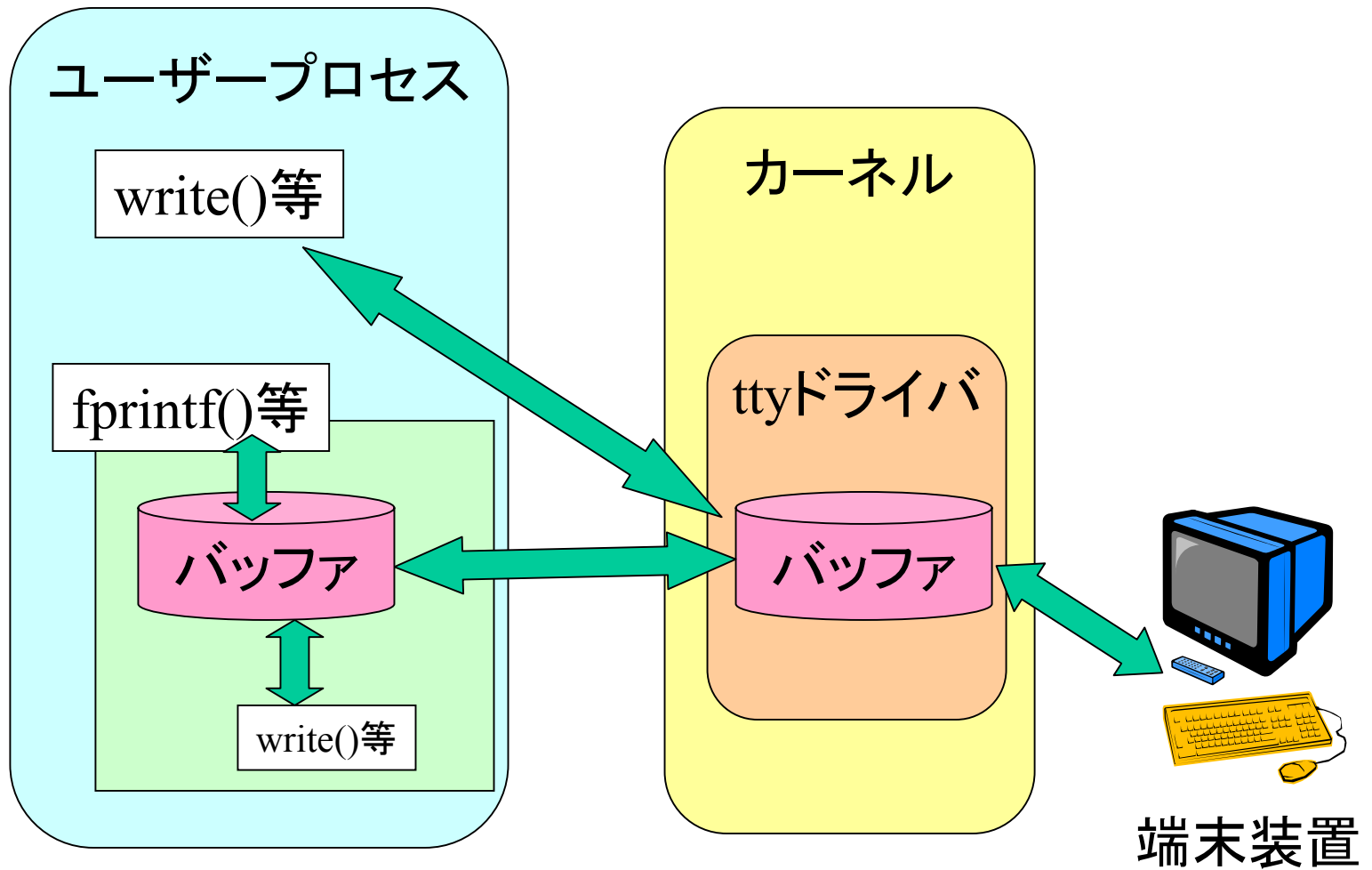
キーボードでプロセスの制御(一部)を行える。



データがディスク等が届くまで 再録



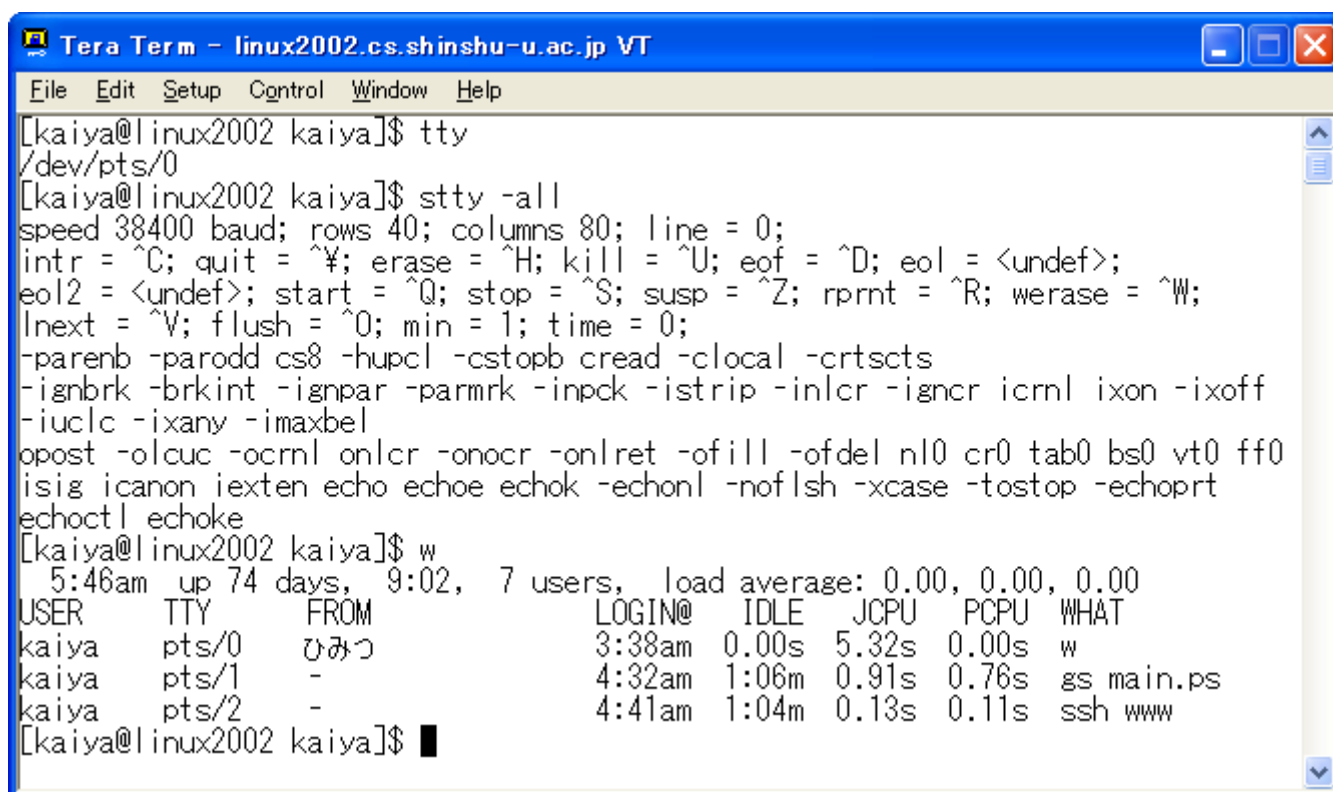
データが端末に届くまで



端末のIDと制御

- 端末装置はファイルの一種として識別できる.
- ttyというコマンドで該当する端末のIDとなるファイルを知ることができる.
- sttyというコマンドで前述の3つの機能をある程度, 制御できる.
 - 制御だけでなく現状の設定もしることができる.

ttyとsttyの利用例



```
Tera Term - linux2002.cs.shinshu-u.ac.jp VT
File Edit Setup Control Window Help
[kaiya@linux2002 kaiya]$ tty
/dev/pts/0
[kaiya@linux2002 kaiya]$ stty -all
speed 38400 baud; rows 40; columns 80; line = 0;
intr = ^C; quit = ^¥; erase = ^H; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; start = ^Q; stop = ^S; susp = ^Z; rpmt = ^R; werase = ^W;
lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff
-iuclc -ixany -imaxbel
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echopr
echoctl echoke
[kaiya@linux2002 kaiya]$ w
 5:46am  up 74 days,  9:02,  7 users,  load average: 0.00, 0.00, 0.00
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU   WHAT
kaiya     pts/0    ひみつ          3:38am   0.00s  5.32s  0.00s  w
kaiya     pts/1    -               4:32am   1:06m  0.91s  0.76s  gs main.ps
kaiya     pts/2    -               4:41am   1:04m  0.13s  0.11s  ssh www
[kaiya@linux2002 kaiya]$
```

端末入出力のまとめ

- 歴史的経緯もあり結構複雑.
- 初級のプログラム開発練習は端末上で行われるが, 初心者には理解不能な挙動をたまにするのは, ここで話したような複雑な構造があるため.
- しかし, プロのコンピュータ技術者ならこの程度の理解は必須.

shell, コマンドインタプリタ

- 通常, 端末の上ではテキストによるコマンドを実行するための対話型プログラムが動作している.
- このようなソフトの総称をshellとかコマンドインタプリタとか呼ぶ.
- 代表例
 - bash (/bin/bash) Linuxでは標準らしい.
 - tcsh (/bin/tcsh) 別のshell

リダイレクション

- shellからコマンドを呼ぶ場合, < や > の記号を使って,
 - 本来ならキーボードから入力するデータをファイルから入力する (<)
 - 本来ならモニタに出力されるデータをファイルに出力する (>)ということができる.
- これらの機能？をリダイレクション(redirection)と呼ぶ.
- リダイレクションを <や>の記号で実行できるのは、あくまでshellの機能であり、OSの機能ではない.

例



```
Tera Term - linux2002.cs.shinshu-u.ac.jp VT
File Edit Setup Control Window Help
[kaiya@linux2002 ~]% ./a.out < fd2.e2fs
[kaiya@linux2002 ~]% ./a.out /home/kaiya/cfp/compsac.txt < fd2.e2fs
30, 31, 32, 33, 34, 35, 36,
[kaiya@linux2002 ~]% ./a.out /home/kaiya/cfp/compsac.txt < fd2.e2fs > result
[kaiya@linux2002 ~]% cat result
30, 31, 32, 33, 34, 35, 36,
[kaiya@linux2002 ~]%
```

リダイレクションの実現

以下のシステムコールを使って実現されている.

1. openで読み先(書く先)を開ける.
2. closeで標準入力(出力)を閉める.
 - ttyドライバとの接続が切れる.
3. dupで1で開けたディスクリプタの複製を標準入力(出力)が接続されていたディスクリプタに繋ぐ.
4. 1で開けた本来のファイルディスクリプタを閉じる.
 - これは開けたままにしておく場合もある.

例：読み先の変更

// 前略: ex4brd.c 演習4の解答例の改造版

```
main(int argc, char* argv[]){
```

// 中略

```
int fd;
```

```
if((fd=open("fd2.e2fs", O_RDONLY))<0) exit(5);
```

```
close(0);
```

```
dup(fd);
```

```
close(fd);
```

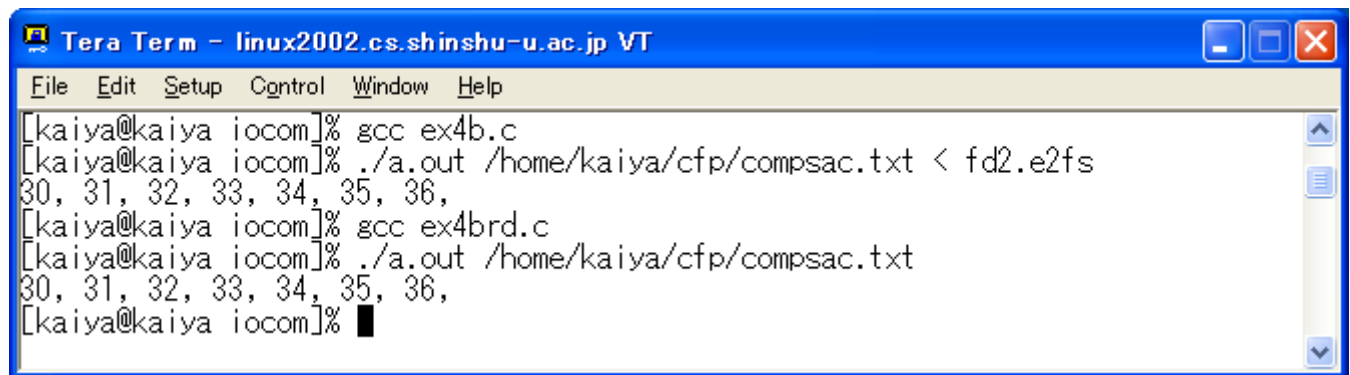
// 中略

// load all blocks in the file system.

```
if(read(0, blocks, ALLBLOCKSIZE)!=ALLBLOCKSIZE) exit(3);
```

// 以下略

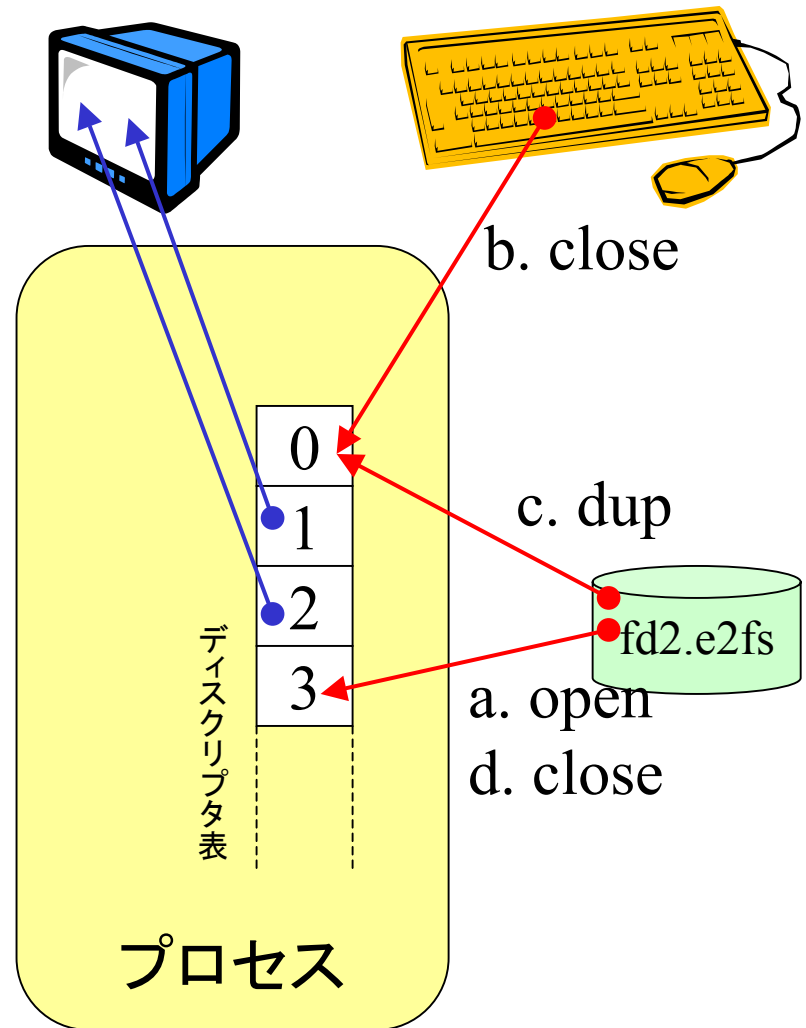
```
}
```



```
Tera Term - linux2002.cs.shinshu-u.ac.jp VT
File Edit Setup Control Window Help
[kaiya@kaiya iocom]% gcc ex4b.c
[kaiya@kaiya iocom]% ./a.out /home/kaiya/cfp/compsac.txt < fd2.e2fs
30, 31, 32, 33, 34, 35, 36,
[kaiya@kaiya iocom]% gcc ex4brd.c
[kaiya@kaiya iocom]% ./a.out /home/kaiya/cfp/compsac.txt
30, 31, 32, 33, 34, 35, 36,
[kaiya@kaiya iocom]% █
```

解説

- a. openでファイルをあける.
- b. closeで標準入力(キーボード)を切断.
- c. dupで複製.
- d. もとのディスクリプタをcloseで閉じる.



shellは何をしているか？

- shellが < > の記号とファイル名を受け取った場合,
- 前述のopen, close, dup, 等を使ったプログラムと同じことを処理してくれている.
- 結果として, UNIX流のプログラムは, 標準入力からデータを得て, 出力へデータを示すように(簡易に)プログラムしても實際上, 不便がないし汎用性がある.
 - 「ファイル名をいれてください」などとアプリ側で対処するプログラムは汎用的でない.

パイプ

- shellにおいて、複数のコマンドを | (縦棒？) で繋ぐことで,
 - | の前にあるコマンドが標準出力に送るはずのデータを,
 - | の後にあるコマンドに受け取らせることができる.
- このような機能をパイプとかパイプラインとか呼ぶ.
- パイプを | の記号で実行できるのは、あくまで shell の機能であり、OS の機能ではない.

例

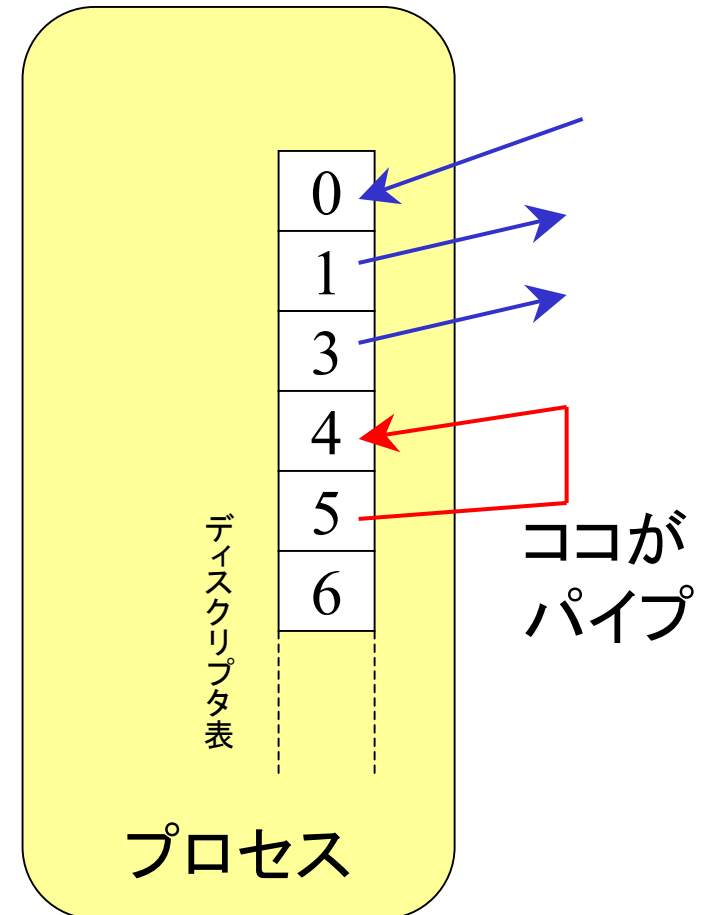
The image displays two screenshots of a Tera Term terminal window, illustrating a process listing example.

Top Screenshot: The terminal window title is "Tera Term - linux2002.cs.shinshu-u.ac.jp VT". It shows a system-wide process listing with columns for user, PID, PPID, flags, start time, time, and command. The listing includes processes like ypbind, sshd, gpm, cannaserver, cron, xfs, atd, and minigetty.

Bottom Screenshot: The terminal window title is "Tera Term - linux2002.cs.shinshu-u.ac.jp VT". It shows a user-specific process listing for the user "kaiya". The command `ps -ef | grep kaiya` is entered and executed, resulting in a list of processes owned by "kaiya", including `-csh`, `Xvnc`, `xterm`, `twm`, `-csh`, `xeyes`, `xclock`, `gs main.ps`, `kterm`, `-bin/tcsh`, `ssh www`, `ps -ef`, and `grep kaiya`.

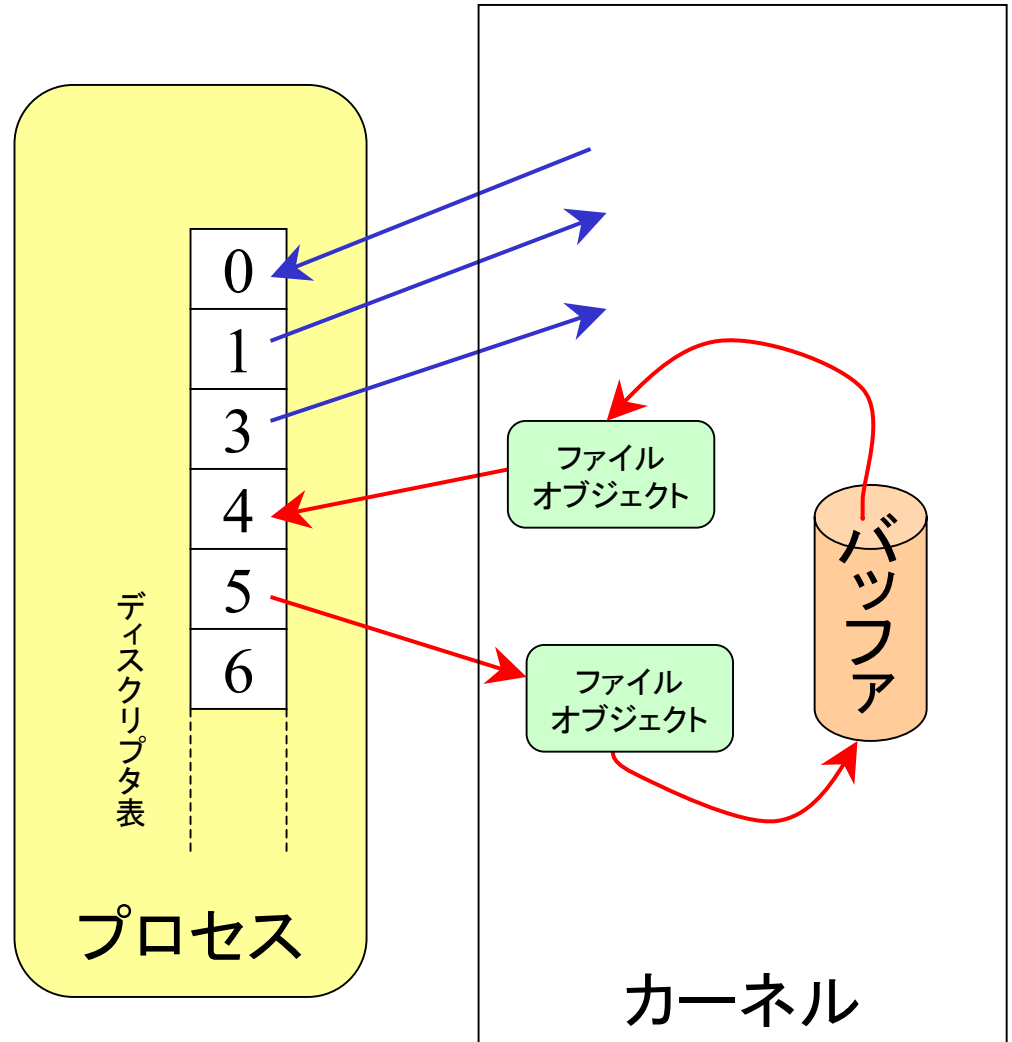
pipeシステムコール

- 読み用と書き用のファイルディスクリプタを生成し,
- 書き側にデータを書くと, 読み側からそのデータを読めるような接続を確立する.
- 単一プロセスでpipeを生成してもほとんど意味が無い(涙)
 - 何故意味ないかは左図参照.



OSから見たpipe

- pipeを通るデータはカーネル内でバッファリングされている.
- pipe自体はiノード番号がつけられる.
- 読み書きそれぞれにファイルオブジェクトが割り当てられる.



例 (はげしく無意味なプログラム)

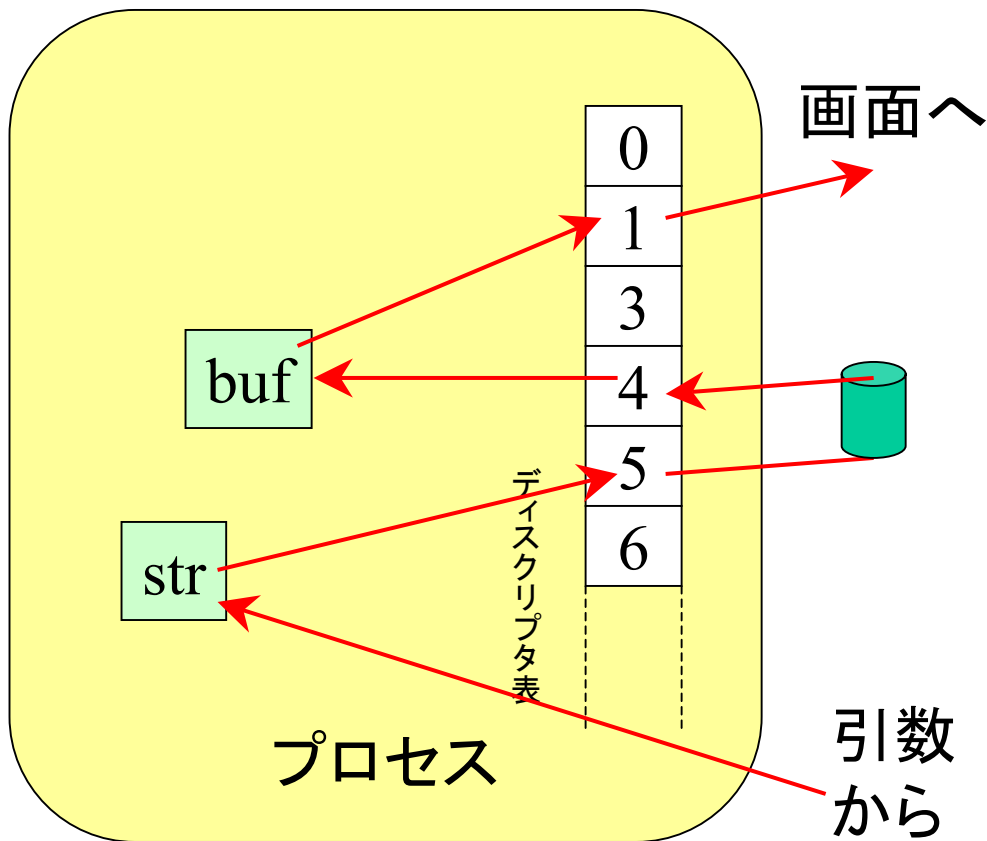
```
// pipe1.c
main(int argc, char* argv[]){
int pipes[2];
char* str;
char buf[100];
int r, w;

if(argc<2) exit(2);
str=argv[1];

if(pipe(pipes)<0) exit(1);

w=write(pipes[1], str, strlen(str));
r=read(pipes[0], buf, w);
write(1, buf, r);

printf(", data %d bytes, write %d bytes, read %d bytes.¥n", strlen(str), w, r);
}
```

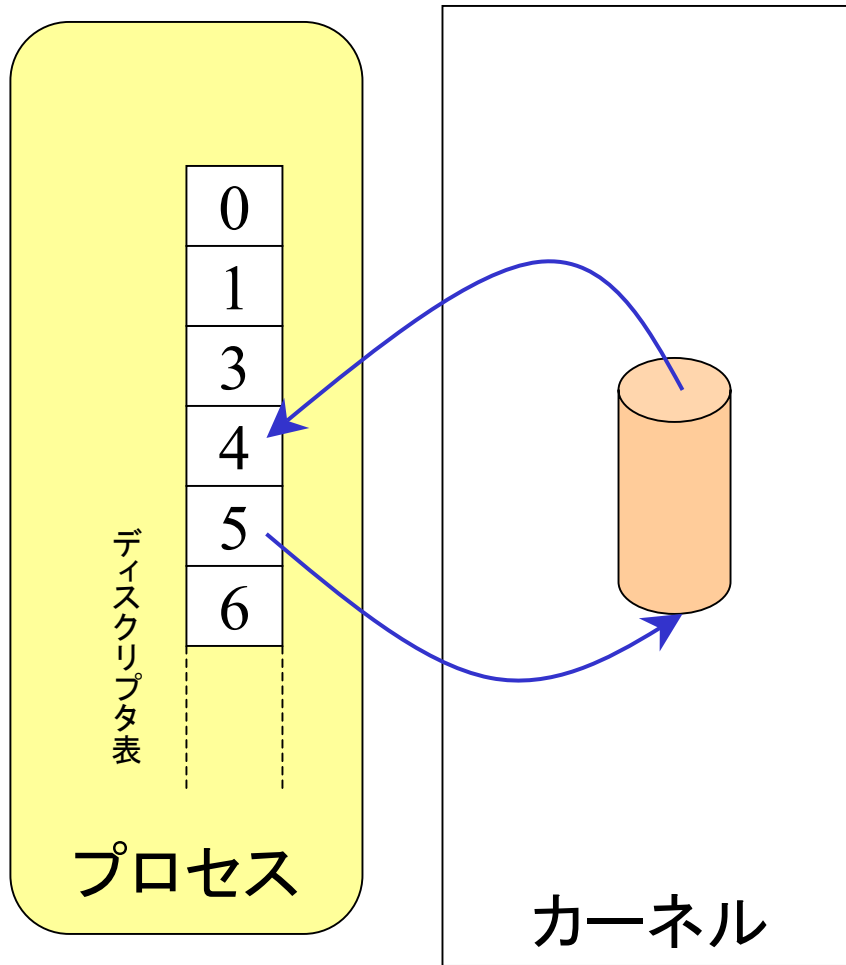


forkとの連携

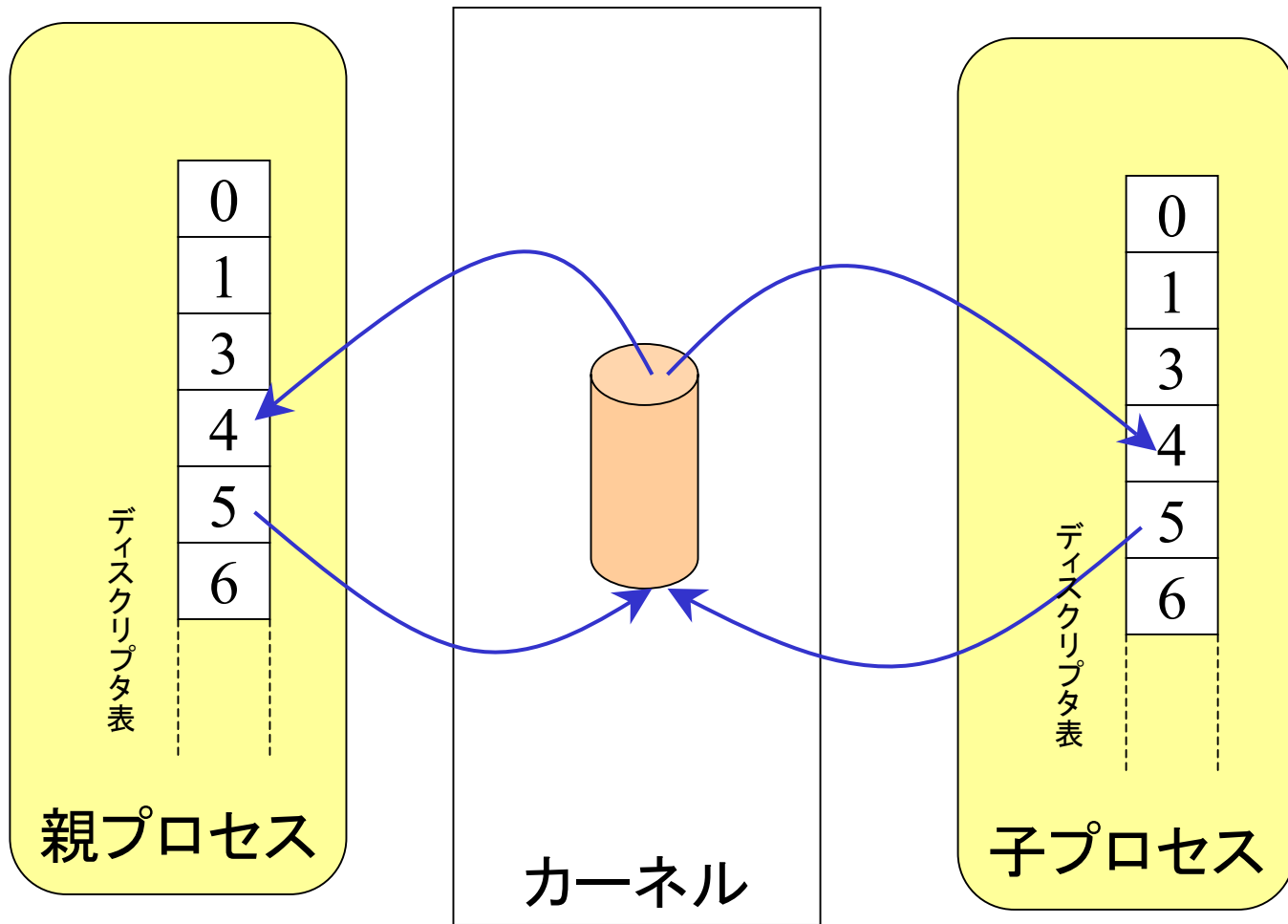
shell上のパイプ(|)を実現するためには、通常、以下のような処理がされる。

1. pipeを作る。
2. forkしてプロセスを2つに複製する。
複製してもpipeは共有されている！
3. 子プロセスの読みパイプを閉じる。
4. 親プロセスの書きパイプを閉じる。
5. 子から親にパイプを通してデータ通信ができる。
ファイルの読み書きと同様の手順で。
6. 本来の標準入力, 出力を閉じて, Dupする。
 - オプション
 - 親子それぞれのプロセスはpipeで作ったディスクリプタではなく、標準の入出力ディスクリプタを使うことができる。

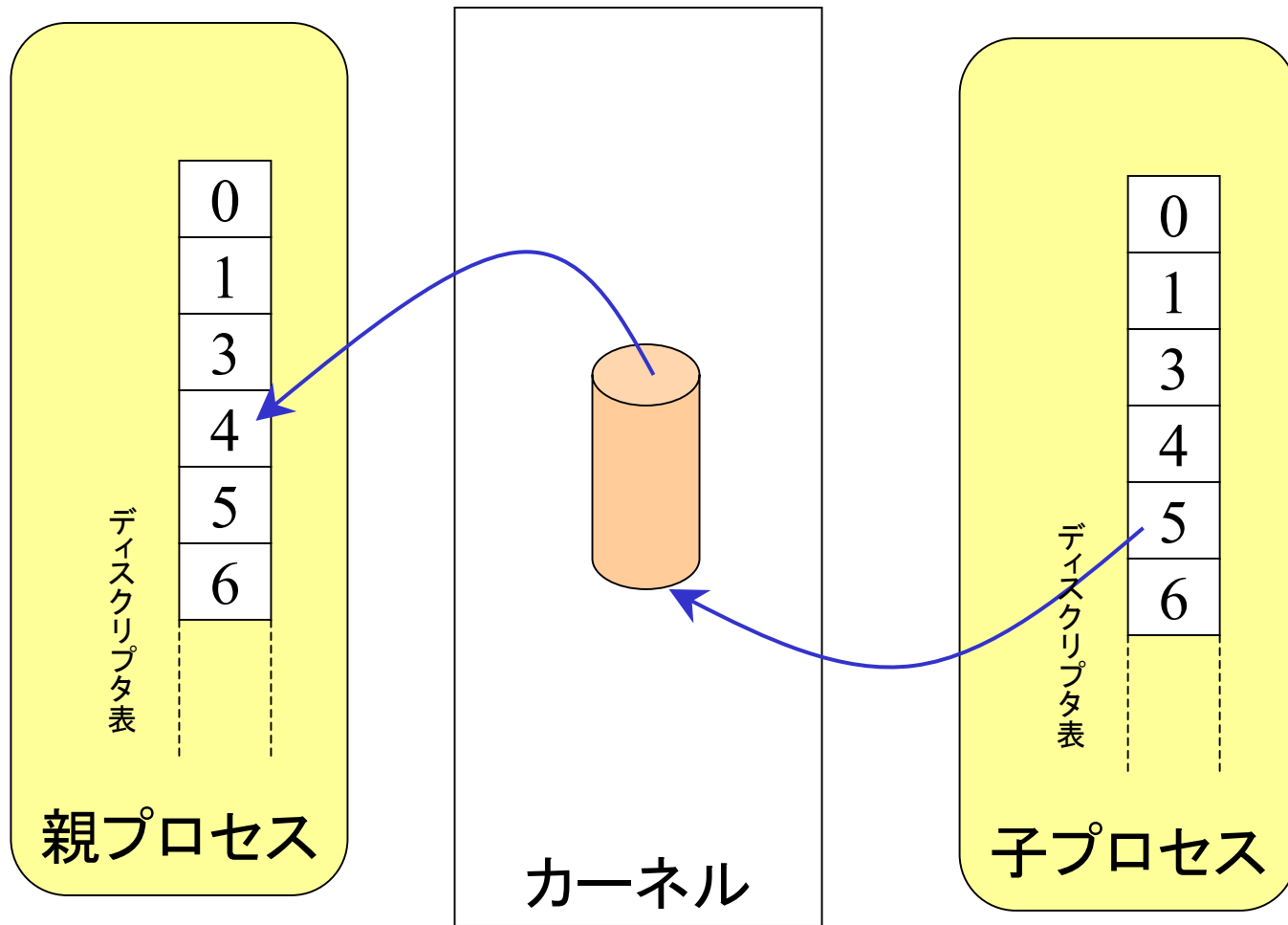
ステップ1 パイプを作る



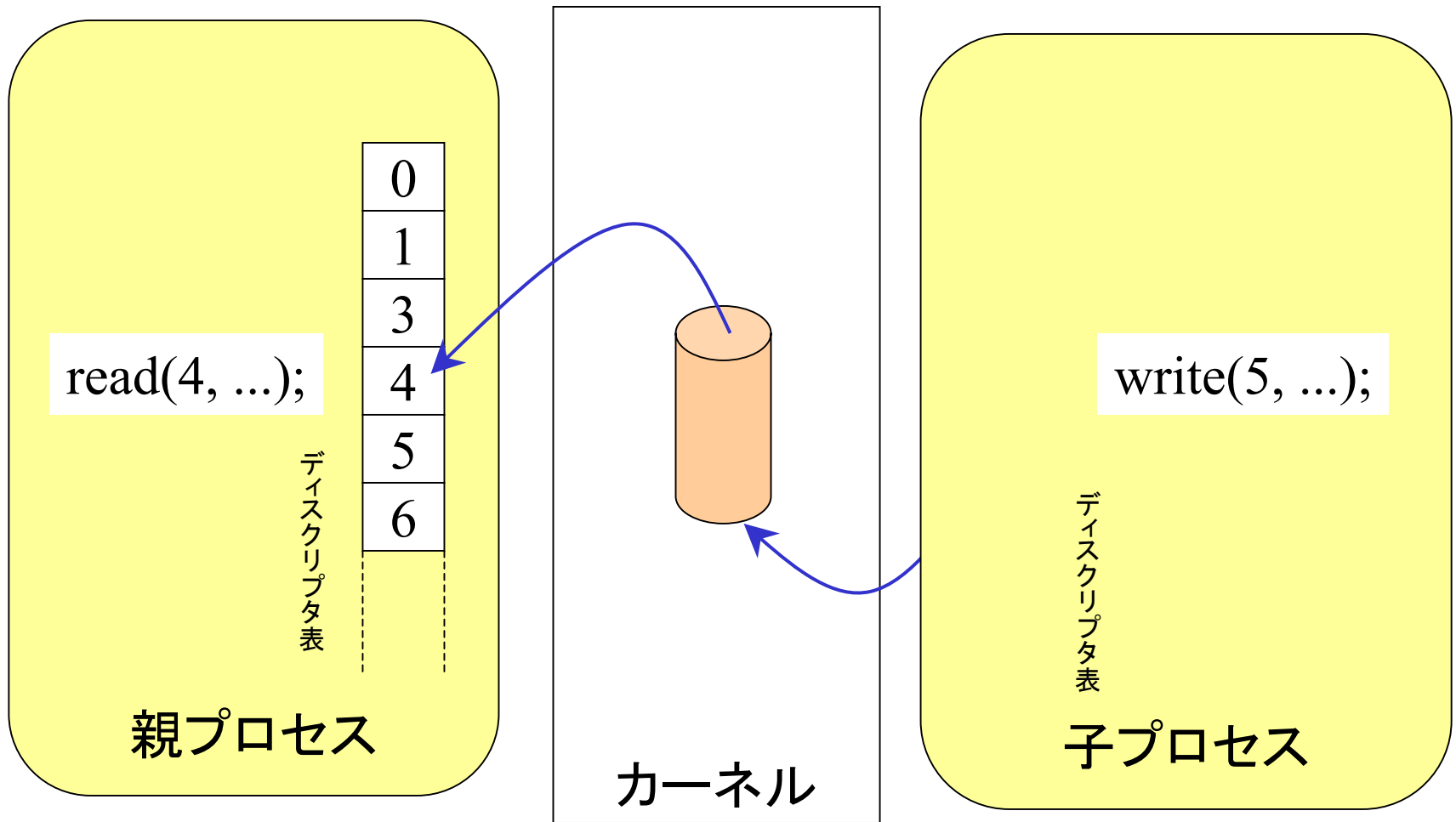
ステップ2 フォーク



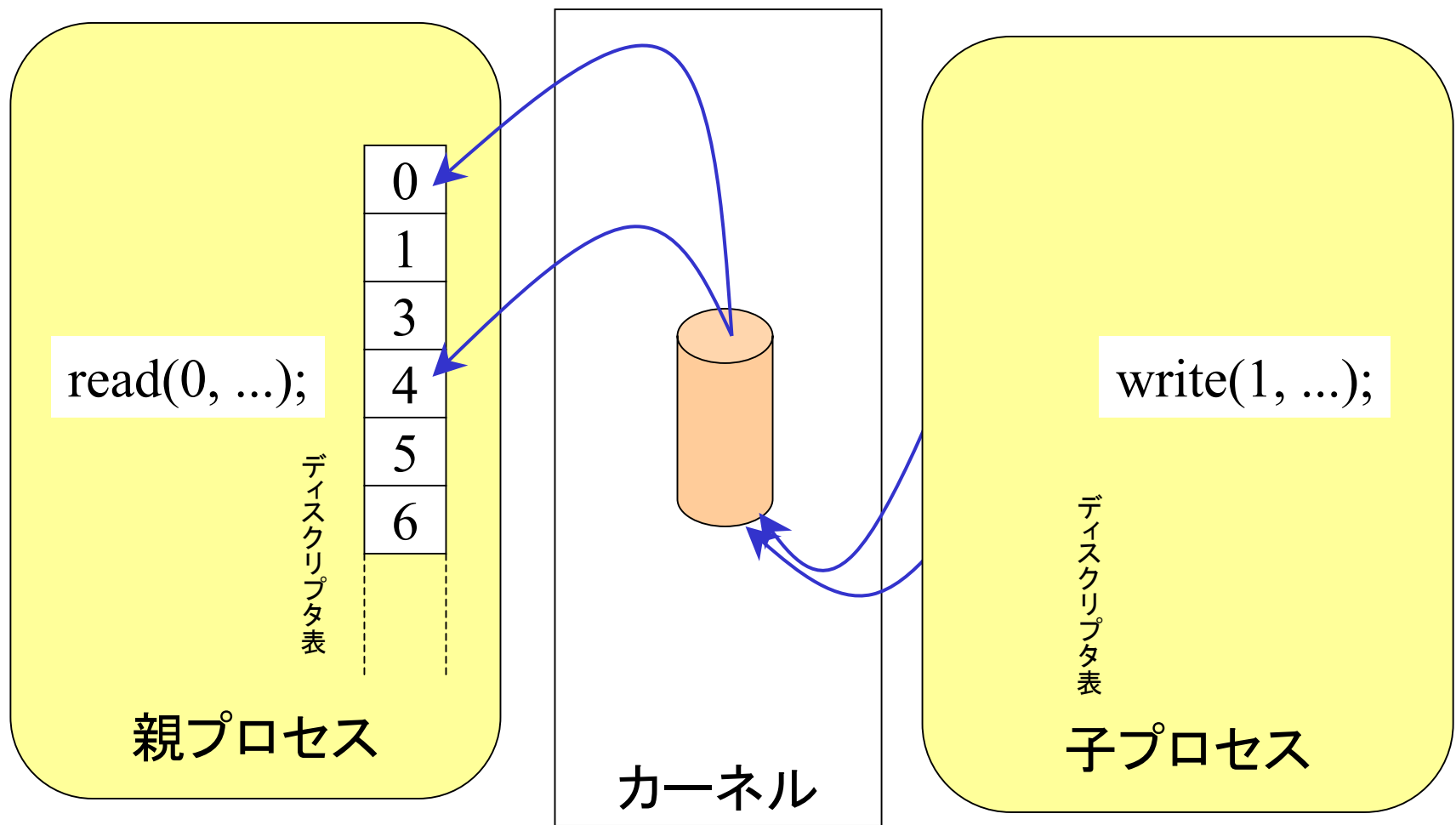
ステップ3,4 不要なFDを閉じる



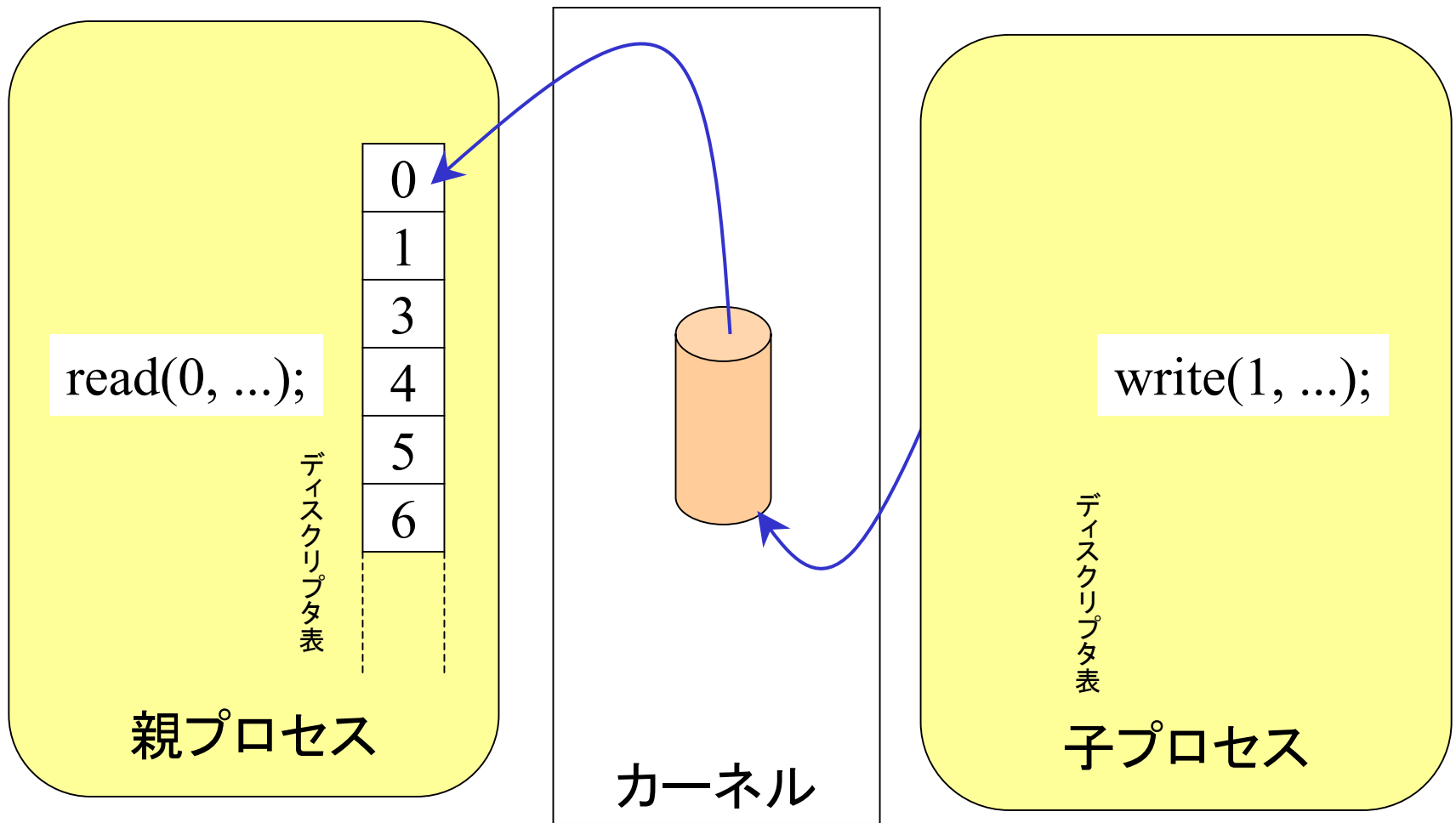
ステップ5 通信



ステップ6 Close, Dup



不要なFDは切ってもよい



例

```
// pipe2.c
main(int argc, char* argv[]) { // ps -ef | grep kaiya とやってることは同じ
int pipes[2]; pid_t pid;

if(pipe(pipes)<0) exit(1);

if((pid=fork())==0) { // in child for writing
    close(pipes[0]); // close read
    close(1);
    dup(pipes[1]);
    execl("/bin/ps", "ps", "-ef", NULL);
} else if(pid>0) { // in parent for reading
    close(pipes[1]); // close write
    close(0);
    dup(pipes[0]);
    execl("/bin/grep", "grep", "kaiya", NULL);
}
}
```

簡易プロセス間通信

- popen, pclose関数を使って, pipeを直接使わずに, 簡単なフィルタを生成することができる.
- プログラムの処理結果を特定の他プログラムに入力したり,
- 特定の他プログラムの結果からデータを読んだりする場合, 簡単にプログラムが書ける.

例

// popen1.c psの結果を読み, その一部を抜き出して, cat -nに出力する.

```
main(int argc, char* argv[]){  
    FILE* fp;  
    FILE* ofp;  
    char buf[100];  
    if((fp=popen("/bin/ps -ef", "r"))==NULL) exit(1);  
    if((ofp=popen("/bin/cat -n", "w"))==NULL) exit(2);  
    while(fgets(buf, 100, fp)!=NULL){  
        int pid;  
        char name[100];  
        if(sscanf(buf, "%s %d", name, &pid)==2){  
            fprintf(ofp, "%d %s¥n", pid, name);  
        }  
    }  
    fclose(ofp);  
}
```