Unix System Programming:
The Big Picture

류은경
ekryu@knu.ac.kr

The Simple Program Model

- There are many sorts of programs; many programs are based on the model below:

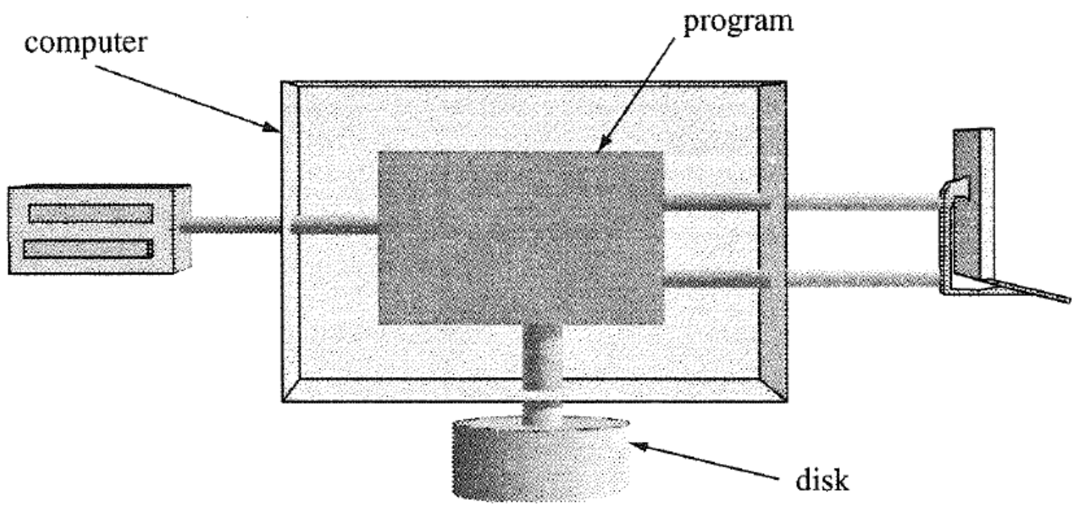


FIGURE 1.1

An application program in a computer.

Typical Program in this Model

```
/* copy from stdin to stdout */
main()
{
   int c;
   while( ( c = getchar() ) != ECF )
       putchar(c);
}
```

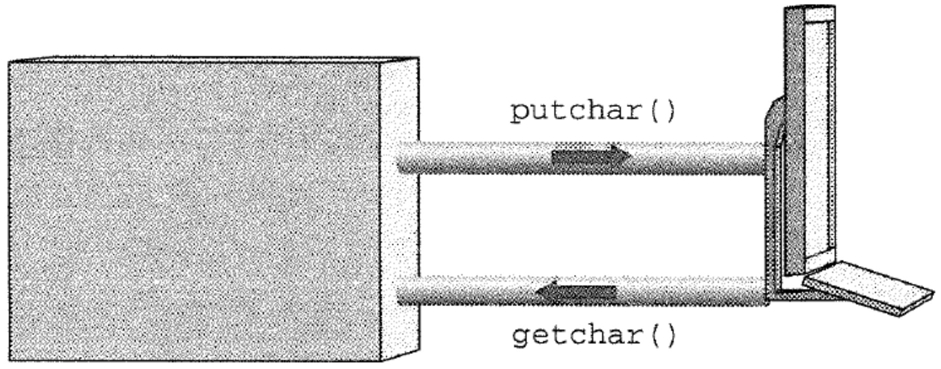


FIGURE 1.2

How application programs see user I/O.

Reality

- **Wha if you log into a multiuser system?**
  - like a typical Unix machine

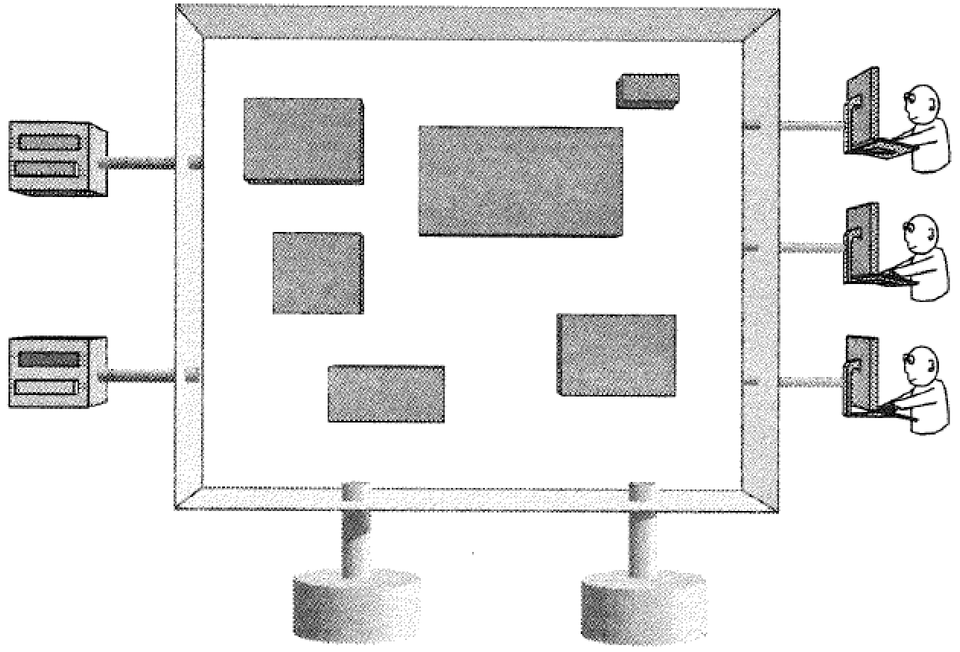


FIGURE 1.3

Reality: many users, programs, and devices.

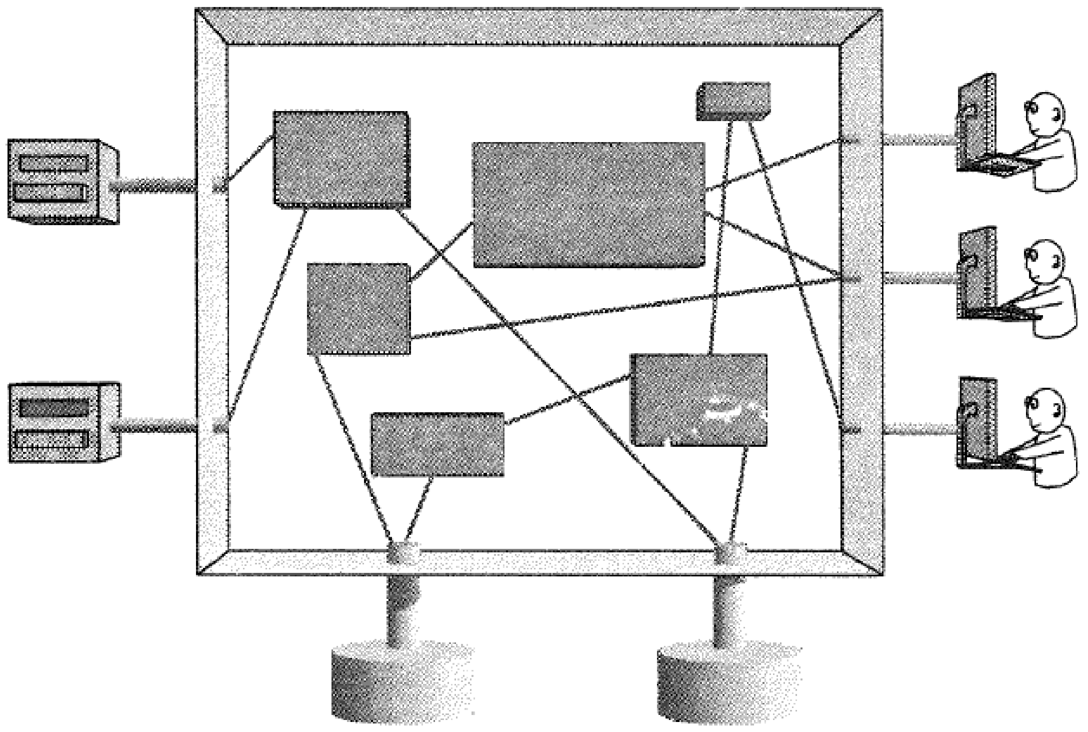


FIGURE 1.4

How are they all connected?

Operating System

- **To manage and protect all the resources**
- **To *connect* the various *devices* to the *programs***

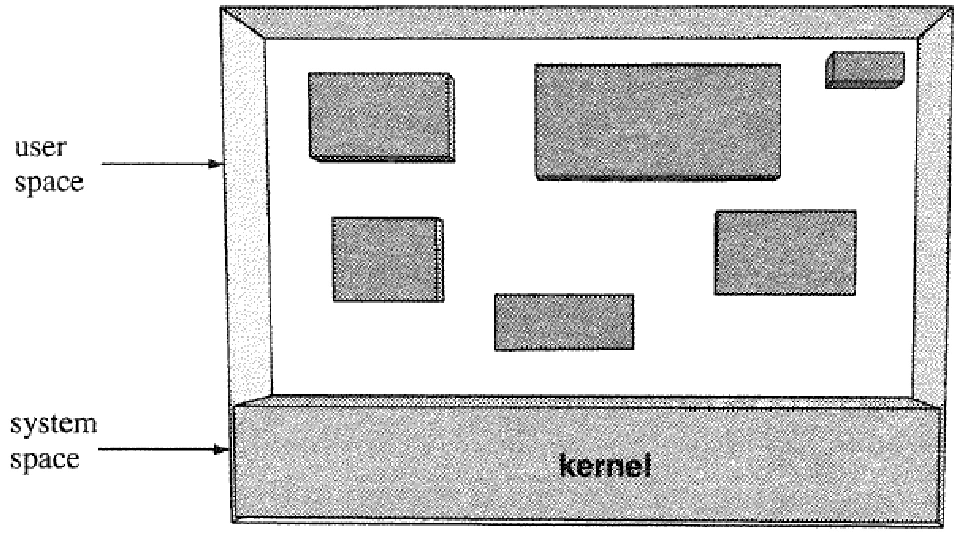


FIGURE 1.5

An operating system is a program.

Providing Services to Programs

- **The kernel is the only program with access to devices**
  - Programs request *services* from the OS
  - The OS contains *code* to provide services

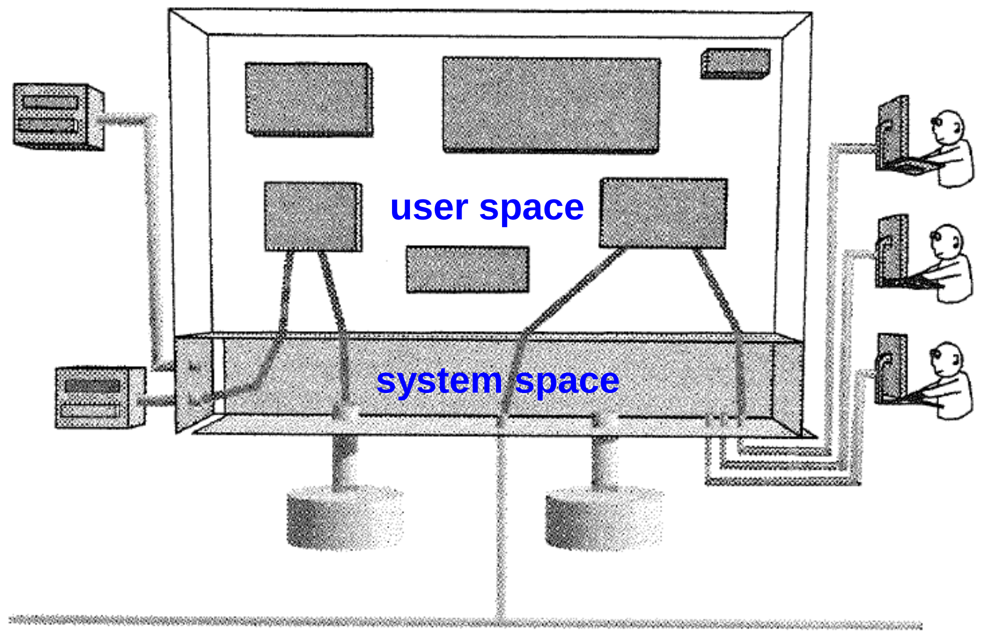


FIGURE 1.6
The kernel manages all connections.

Unix System Programming

System Resources

- **The services provided by the kernel :**
  - Processors: …
  - Input / Output: …
  - Process management: …
  - Memory: …
  - Devices: …
  - Timers: …
  - Inter-process communication: …
  - Networking: …

Understanding Systems Programming

- **Understanding how the kernel works and how to write programs that use these services**
  - What are the details of each type of the kernel service?
  - How does data get from a device to the program and back?

- **System programs use those services directly**
  - Also known as **system utilities**
  - Provide a convenient environment for program development and execution

Our Method: Three Simple Steps

- **We shall learn about *Unix services* by**
  1. Looking at “real” programs : ls, cat, grep …
  2. Looking at the ***system calls*** to invoke the servies : fork(), exec(), open() …
  3. Writing our own version

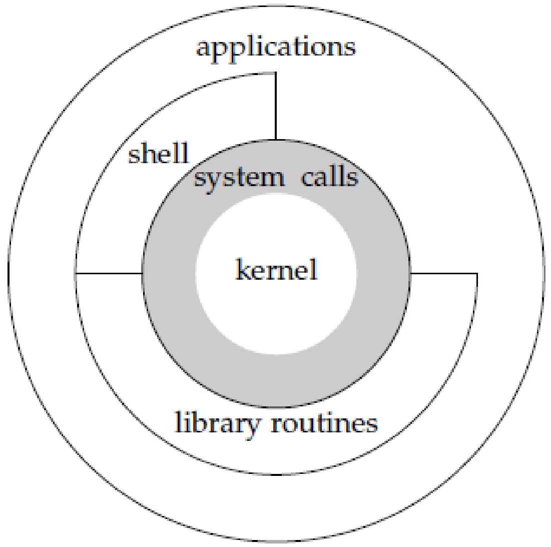


Figure 1.1 Architecture of the UNIX operating system

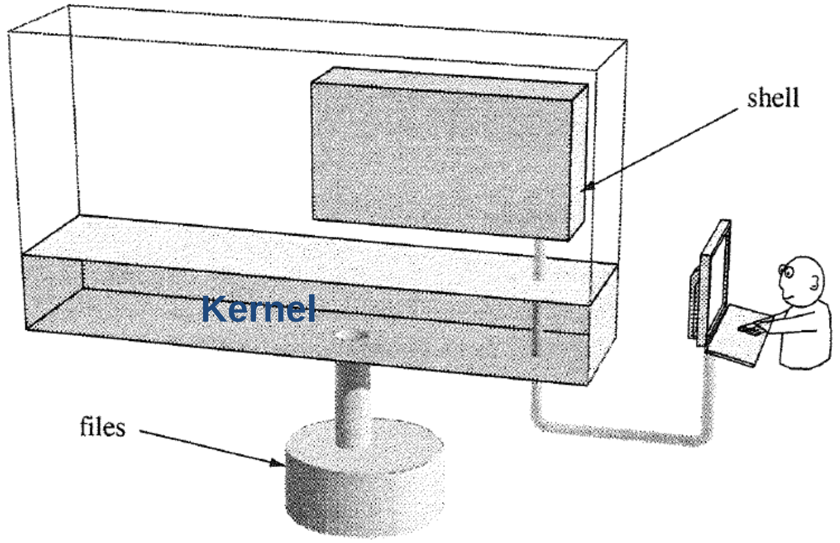
Unix System Programming

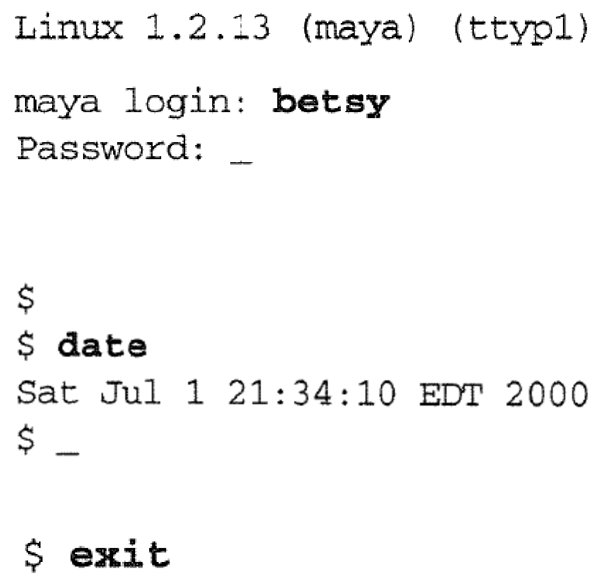
What does Unix do?

- **Log In – Run Programs – Log Out**
- **Working with Directories**
  - mkdir, rmdir, cd, ls
- **Working with Files**
  - touch, rm, cp, mv

Log In – Run Programs – Log out

- **How Does That work?**
  - Login: …
  - Run programs: ….
  - Logout: …





```
Linux 1.2.13 (maya) (ttyp1)

maya login: betsy
Password: _


$
$ date
Sat Jul 1 21:34:10 EDT 2000
$ _

$ exit
```

FIGURE 1.7

A user logged into a computer.

Working with Directories (1/2)

- **Unix organizes *files* into a tree-structured dir system**

- **A Tree of Directories :**

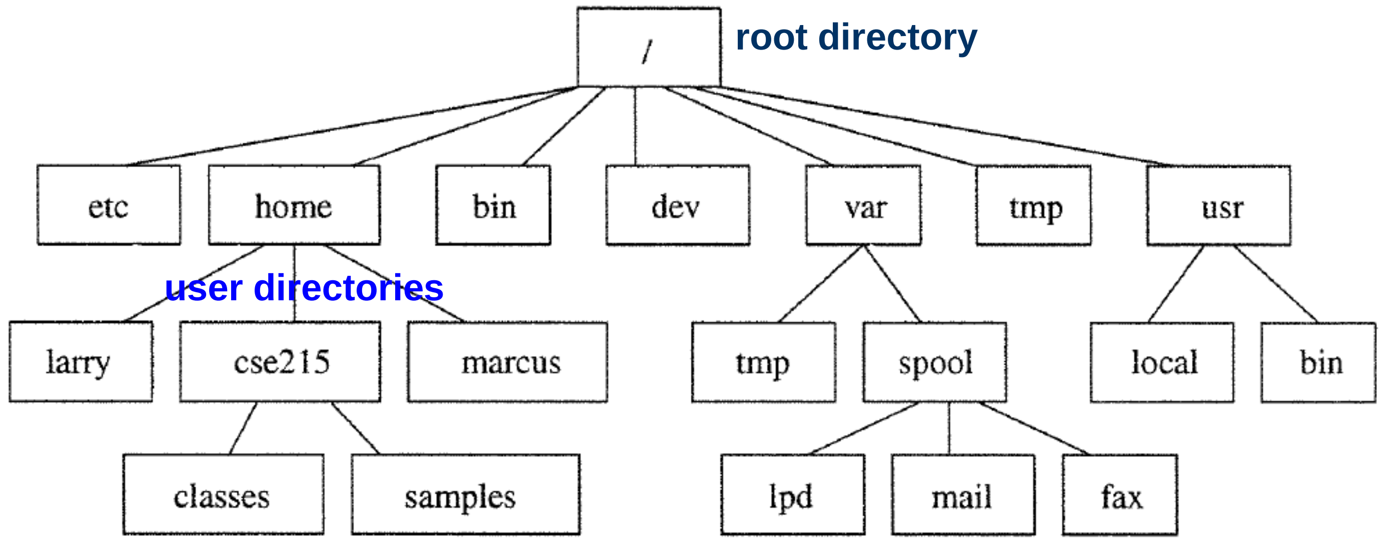


FIGURE 1.8

Part of the directory tree.

Working with Directories (2/2)

- **Commands for Working with Directories**
  - ls – list directory contents
    ```
    $ ls /etc
    $ ls /
    ```
  - cd – change to a different directory
    ```
    $ cd /bin
    $ cd ..
    $ cd
    ```
  - pwd – print path to current directory
    ```
    $ pwd
    ```
  - mkdir, rmdir – make and remove directories
    ```
    $ cd
    $ mkdir jokes
    $ rmdir jokes
    ```

Working with Files (1/2)

- **Names of Files:** up to about 255 characters
- **cat, more, less – examine file contents**

```
$ cat shopping-list
soap
cornflakes
milk
apples
jam
$
$ more longfile
```

- **cp – make a copy of a file**

```
$ cp shopping-list last.week.list
```

Working with Files (2/2)

- **rm – delete a file**

```
$ rm old.data junk shopping.june1992
```

- **mv – rename or move a file**

```
$ mv prog1.c first_program.c
```

Rename

```
$ mkdir mycode
$ mv first_program.c mycode
```

Move

- **lpr, lp – print file on paper**

```
$ lpr filename
```

File Permission Attributes

- **To allow users to control access to their files :**
  - Unix assigns to each file several attributes

- **ls –l shows attributes of a file:**

```
$ ls -l outline.01
-rwxr-x---    1 molay     users    1064 Jun 29 00:39 outline.01
  file           user     group     size  modification date and
  permission     name     name            time
  attributes     (owner/
                 creater)


 - r w x    r w x    r w x        r: read, w: write, x:execute
   user     group    other
```

Unix System Programming

Larger Perspective

- **The *larger system* is one that consists of**
  - more than one user,
  - more than one program,
  - more than one computer, and
  - the connections among these people, programs, and computers.

- **Ex)**
  - Internet bridge tournaments
  - bc/dc program

Internet Bridge Tournaments

- **This bridge example introduces the *three topics* in Unix system programming:**
  - Communication: …
  - Process Coordination: …
  - Network Access: ..

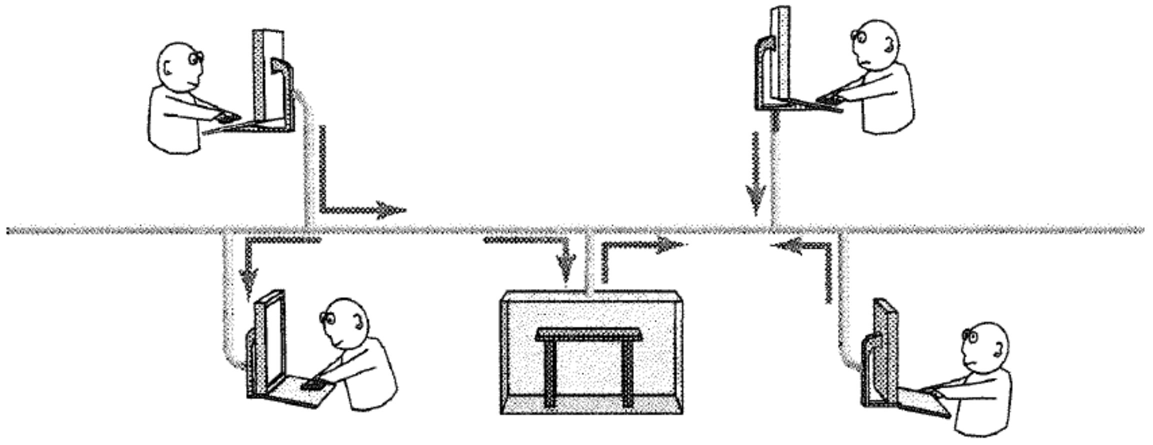
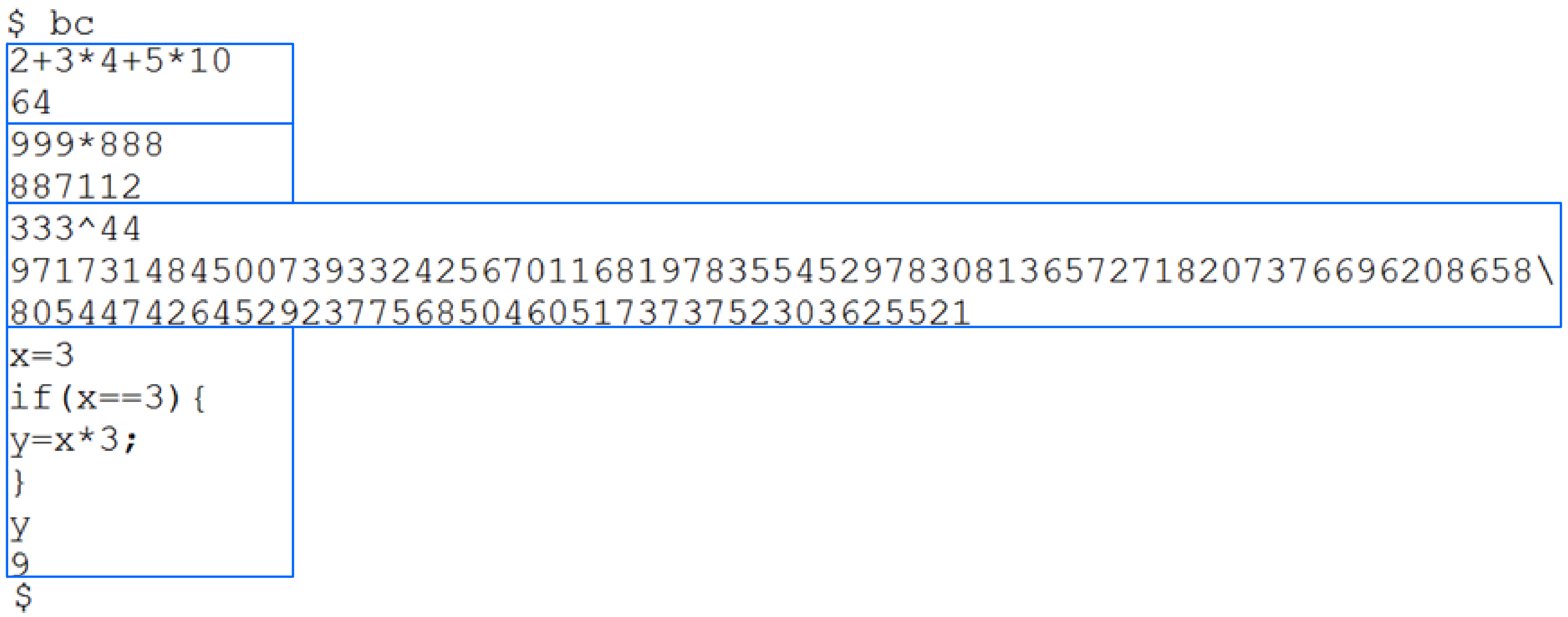


FIGURE 1.11

Separate programs send messages to each other.

Online Bridge

bc Calculator



```
$ bc
2+3*4+5*10
64
999*888
887112
333^44
971731484500739332425670116819783554529783081365727182073766962086 58\
80544742645292377568504605173737523036255 21
x=3
if(x==3){
y=x*3;
}
y
9
$
```

To exit from bc, press **Ctrl-D or quit**

```
echo Hello, World!
echo "4 * (2 + 3)" | bc
echo "This is a test" > test.txt
```

```
$ bc
2 + 3
5
<-- press Ctrl-Z here

Stopped
$ ps
```

※ lists the processes you are running

```
PID   TTY    S          TIME CMD
25102 ttyp2  T       0:00.02 bc
27081 ttyp2  T       0:00.01 dc -
27560 ttyp2  I       0:00.59 -bash
27681 ttyp2  T       0:00.00 bc
$ fg
<-- press Ctrl-D here
```

The **fg** continues a stopped job by running it in the foreground.

```
$ man dc
User Commands dc(1)

NAME
     dc - desk calculator

SYNOPSIS
     dc [ filename ]

DESCRIPTION
     dc is an arbitrary precision arithmetic package. Ordinarily
     it operates on decimal integers, but one may specify an
     input base, output base, and a number of fractional digits
     to be maintained. The overall structure of dc is a stacking
     (reverse Polish) calculator. If an argument is given, input
     is taken from that file until its end, then from the standard
     input.
```

```
$ dc
2
3
+
p
5
```

※ postfix notation

`bc` Calculator

- **It communicates with a process running `dc`**
  - through a communication system called ***pipes***

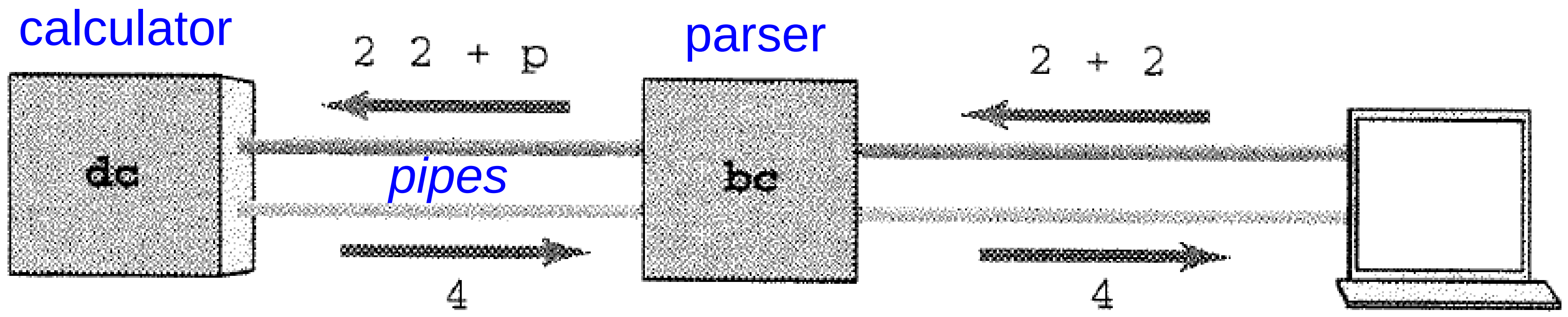


FIGURE 1.12

Programs send messages to each other.

※ The GNU version of bc uses an internal stack-based calculator instead of dc.

`bc/dc`

- **This `bc/dc` is another program that involves**
  - different processes
  - some sort of communication
  - cooperation

- **Learning system programming consists of learning**
  - how to build these separate programs and
  - how to build the connections and cooperation.

Unix System Programming

“What does `more` do?” “How does `more` work?”

- **It displays a file one screenful at a time**

```
$ more /usr/include/utmp.h
```

- **USAGE:**

```
$ more filename
$ command | more
$ more < filename
```

- **Logic**

```
+----> show 24 lines from input
| +--> print [more?] message
| |    Input Enter, SPACE, or q
| +--  if Enter, advance one line
+----  if SPACE
       if q --> exit
```

1st version : more01.c

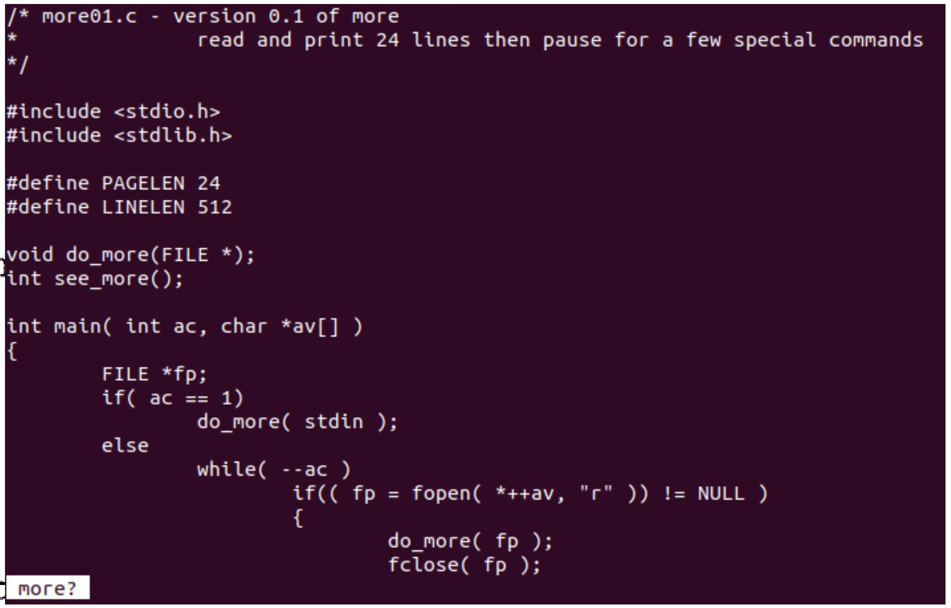
- **Compile and run it**

```
                                   < Linux >
$ cc more01.c -o more01            ※ cc → gcc
$ more01 more01.c        ──→       ※ $ ./more01 more01.c
```

```c
/* more01.c - version 0.1 of more
 *      read and print 24 lines then pause for a few special commands
 */
#include        <stdio.h>
#include        <stdlib.h>
#define PAGELEN 24
#define LINELEN 512

void do_more(FILE *);
int see_more();

int main( int ac , char *av[] )
{
        FILE *fp;
        if ( ac == 1 )
                do_more( stdin );
        else
                while ( --ac )
                        if ( (fp = fopen( *++av , "r" )) != NULL )
                        {
                                do_more( fp ) ;
                                fclose( fp );
                        }
                        else
                                exit(1);
        return 0;
}
```

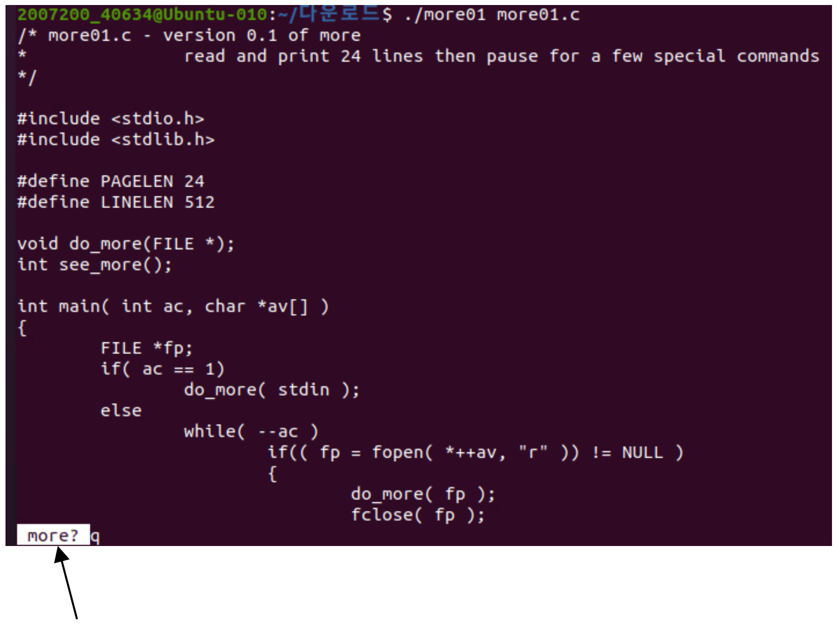
```
$ ./more01 filename
```

```
void do_more( FILE *fp )
/*
 * read PAGELEN lines, then call see_more() for further instruction
 */
{
       char    line[LINELEN];
       int     num_of_lines = 0;
       int     see_more(), reply;

       while ( fgets( line, LINELEN, fp ) ){            /* more input
              if ( num_of_lines == PAGELEN ) {          /* full screen? */
                     reply = see_more();                /* y: ask user  */
                     if ( reply == 0 )                  /*    n: done   */
                            break;
                     num_of_lines -= reply;             /* reset count  */
              }
              if ( fputs( line, stdout ) == EOF )       /* show line    */
                     exit(1);                           /* or die       */
              num_of_lines++;                           /* count it     */
       }

}
```



```
/* more01.c - version 0.1 of more
*               read and print 24 lines then pause for a few special commands
*/

#include <stdio.h>
#include <stdlib.h>

#define PAGELEN 24
#define LINELEN 512

void do_more(FILE *);
int see_more();

int main( int ac, char *av[] )
{
        FILE *fp;
        if( ac == 1)
                do_more( stdin );
        else
                while( --ac )
                        if(( fp = fopen( *++av, "r" )) != NULL )
                        {
                                do_more( fp );
                                fclose( fp );
 more? 
```

```
int see_more()
/*
 *      print message, wait for response, return # of lines to advance
 *      q means no, space means yes, CR means one line
 */
{
        int     c;

        printf("\033[7m more? \033[m");          /* reverse on a vt100  */
        while( (c=getchar()) != EOF )                   /* get response */
        {
                if ( c == 'q' )                  /* q -> N              */
                        return 0;
                if ( c == ' ' )                  /* ' ' => next page    */
                        return PAGELEN;          /* how many to show    */
                if ( c == '\n' )                 /* Enter key => 1 line */
                        return 1;
        }
        return 0;
}
```

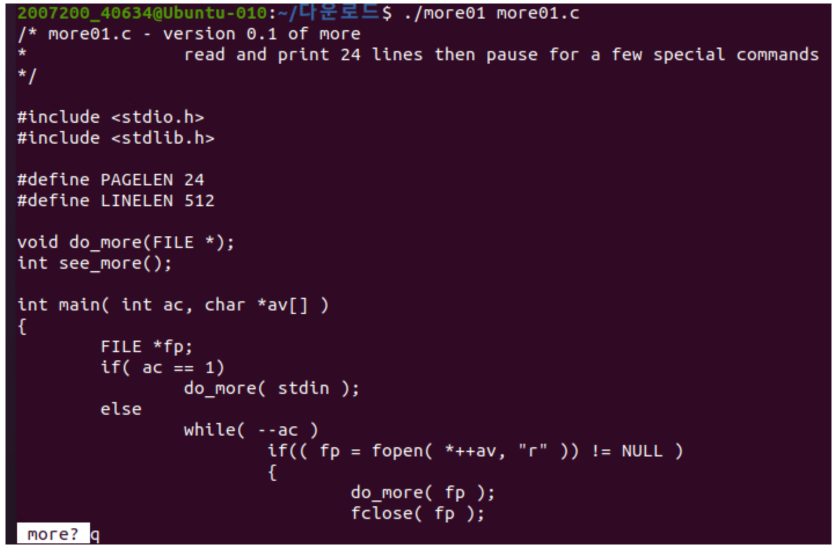


```
2007200_40634@Ubuntu-010:~/다운로드$ ./more01 more01.c
/* more01.c - version 0.1 of more
*               read and print 24 lines then pause for a few special commands
*/

#include <stdio.h>
#include <stdlib.h>

#define PAGELEN 24
#define LINELEN 512

void do_more(FILE *);
int see_more();

int main( int ac, char *av[] )
{
        FILE *fp;
        if( ac == 1)
                do_more( stdin );
        else
                while( --ac )
                        if(( fp = fopen( *++av, "r" )) != NULL )
                        {
                                do_more( fp );
                                fclose( fp );
 more? q
```

```
+----> show 24 lines from input
| +--> print [more?] message
| |    Input Enter, SPACE, or q
| +--  if Enter, advance one line
+----  if SPACE
       if q --> exit
```

1st version : `more`

- **Subtle problems:**
  - If you press the space bar or the **“q” key**, nothing happens **until you press Enter.**
  - Also, the little “more?” message is still there.



```
2007200_40634@Ubuntu-010:~/다운로드$ ./more01 more01.c
/* more01.c - version 0.1 of more
*               read and print 24 lines then pause for a few special commands
*/

#include <stdio.h>
#include <stdlib.h>

#define PAGELEN 24
#define LINELEN 512

void do_more(FILE *);
int see_more();

int main( int ac, char *av[] )
{
        FILE *fp;
        if( ac == 1)
                do_more( stdin );
        else
                while( --ac )
                        if(( fp = fopen( *++av, "r" )) != NULL )
                        {
                                do_more( fp );
                                fclose( fp );
 more? q
```

Using Pipeline

```
$ who | more
$ ls /bin |./more01
```

```
$ ls /bin | more

$ ls /bin | ./more01
```

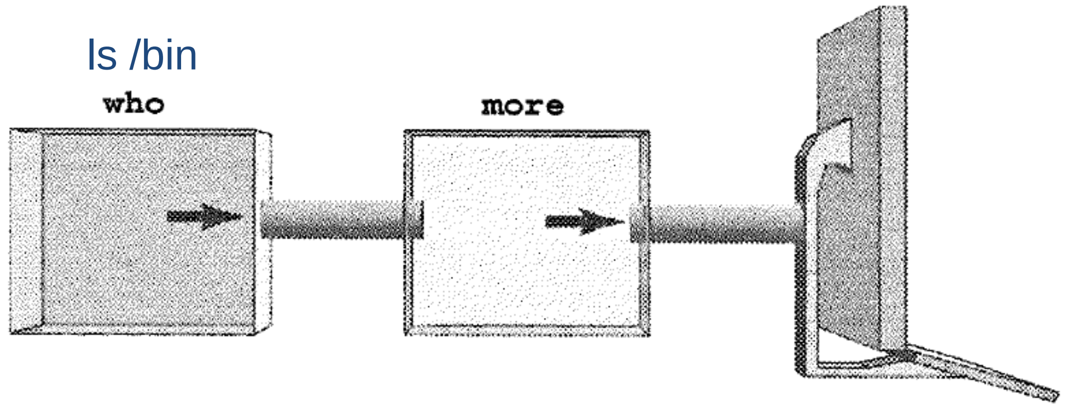


FIGURE 1.14

more reads stdin.

- **`more01`** does **not pause** after 24 lines;
- Reason? …

- **How does the real `more` solve this problem?**
  - **Read from the keyboard directly**!
  - There is a special file, called **/dev/tty**, which is actually a connection to the keyboard and screen.

```
$ ls /bin | more

$ ls /bin | ./more01
```

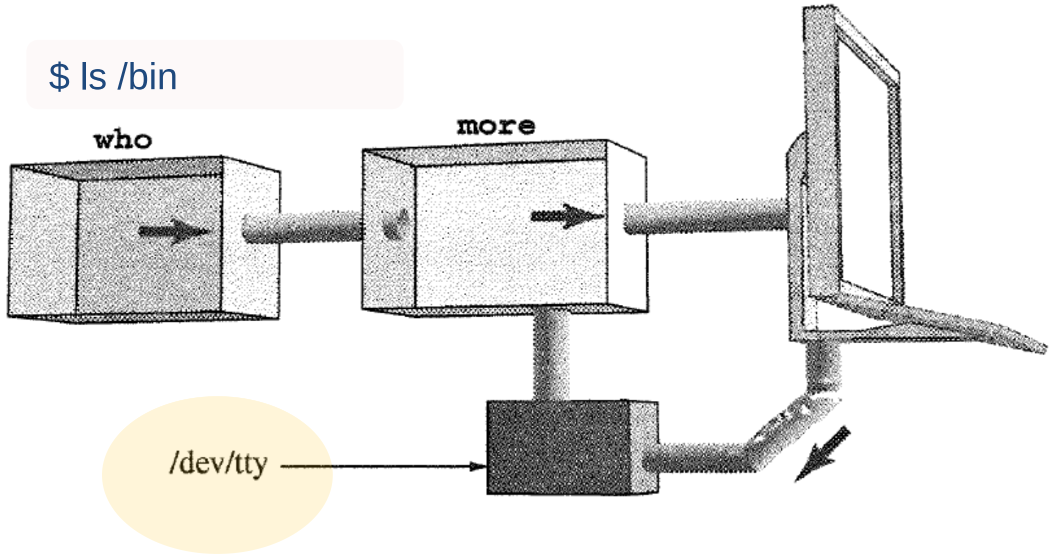


FIGURE 1.15

who reads user input from a terminal.

```
/* more02.c - version 0.2 of more
 *      read and print 24 lines then pause for a few special commands
 *      feature of version 0.2: reads from /dev/tty for commands
*/
#include        <stdio.h>
#include        <stdlib.h>
#define PAGELEN 24
#define LINELEN 512

void do_more(FILE *);
int see_more(FILE *);

int main( int ac , char *av[] )
{
        FILE *fp;

        if ( ac == 1 )
                do_more( stdin );
        else
                while ( --ac )
                        if ( (fp = fopen( *++av , "r" )) != NULL )
                        {
                                do_more( fp ) ;
                                fclose( fp );
                        }
                        else
                                exit(1);
        return 0;
}
```

```
void do_more( FILE *fp )
/*
 * read PAGELEN lines, then call see_more() for further instructions
 */
{
        char    line[LINELEN];
        int     num_of_lines = 0;
        int     see_more(FILE *), reply;
        FILE    *fp_tty;

        fp_tty = fopen( "/dev/tty", "r" );          /* NEW: cmd stream   */
        if ( fp_tty == NULL )                       /* if open fails     */
                exit(1);                            /* no use in running*/

        while ( fgets( line, LINELEN, fp ) ){          /* more input    */
                if ( num_of_lines == PAGELEN ) {       /* full screen? */
                        reply = see_more(fp_tty);   /* NEW: pass FILE * */
                        if ( reply == 0 )              /*    n: done    */
                                break;
                        num_of_lines -= reply;         /* reset count */
                }
                if ( fputs( line, stdout ) == EOF )    /* show line   */
                        exit(1);                       /* or die      */
                num_of_lines++;                        /* count it    */
        }
}
```

```
int see_more(FILE *cmd)                             /* NEW: accepts arg */
/*
 * print message, wait for response, return # of lines to advance
 * q means no, space means yes, CR means one line
 */
{
       int     c;
       printf("\033[7m more? \033[m");            /* reverse on a vt100  */
       while( (c=getc(cmd)) != EOF )              /* NEW: reads from tty */
       {
              if ( c == 'q' )                     /* q -> N              */
                     return 0;
              if ( c == ' ' )                     /* ' ' => next page    */
                     return PAGELEN;              /* how many to show    */
              if ( c == '\n' )                    /* Enter key => 1 line */
                     return 1;
       }
       return 0;
}
```

2nd version : `more`

- **Compile and run it**

```
$ cc -o more02 more02.c
$ ls /bin |./more02
```

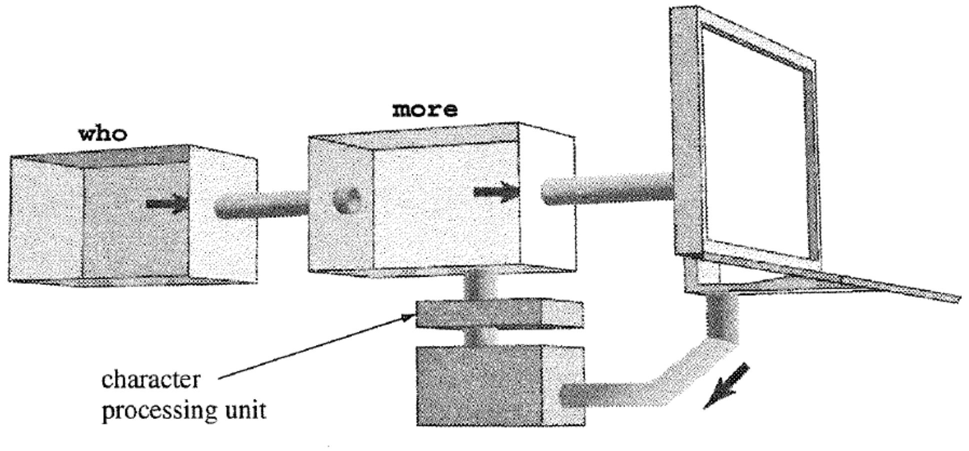
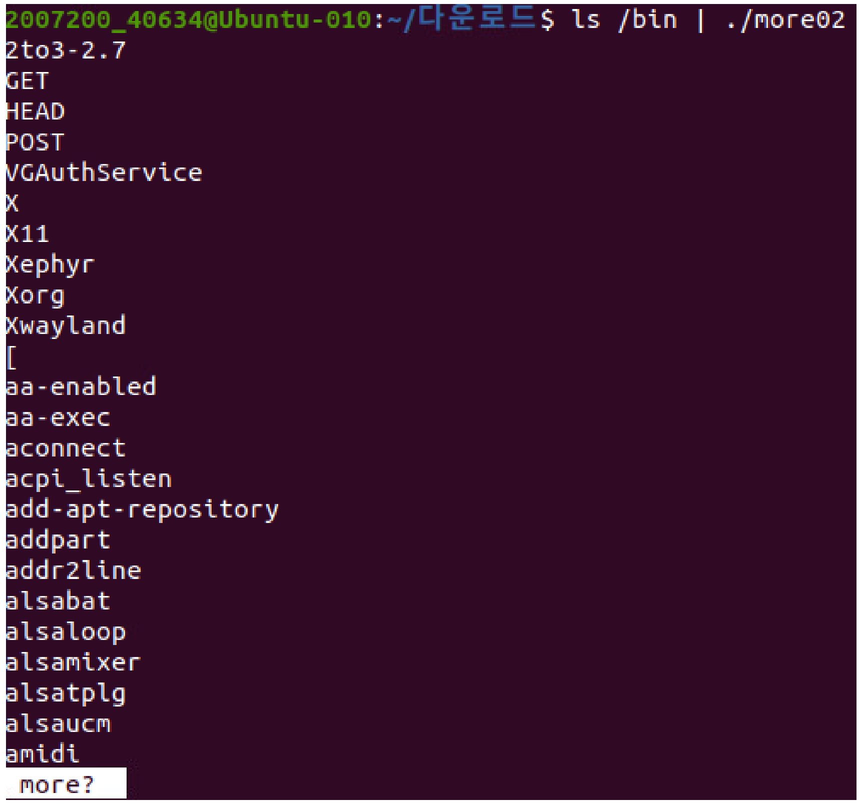


FIGURE 1.16

The connection to the terminal has settings.



```
2007200_40634@Ubuntu-010:~/다운로드$ ls /bin | ./more02
2to3-2.7
GET
HEAD
POST
VGAuthService
X
X11
Xephyr
Xorg
Xwayland
[
aa-enabled
aa-exec
aconnect
acpi_listen
add-apt-repository
addpart
addr2line
alsabat
alsaloop
alsamixer
alsatplg
alsaucm
amidi
 more?
```

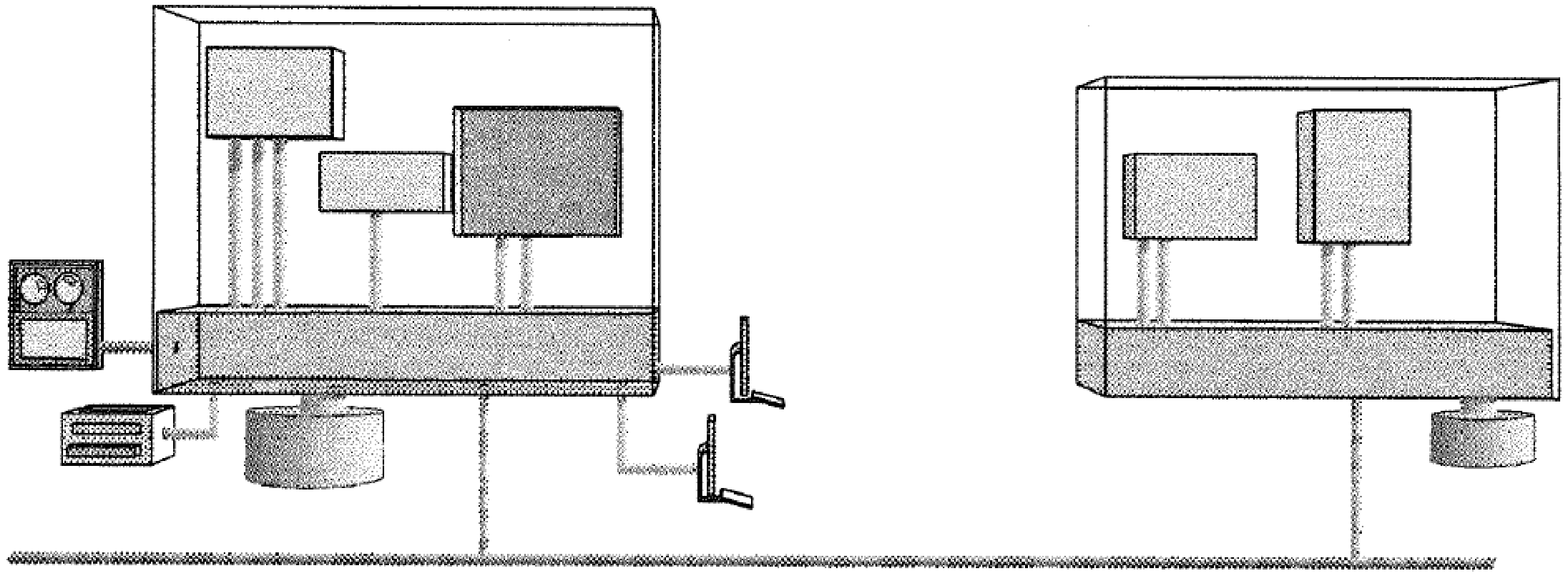


FIGURE 1.17

A diagram of the main structure of a Unix system.

SUMMARY

- **Computer systems that run several programs for several users at the same time require a central management program.**

- **The Unix kernel is a program that schedules programs and controls access to resources.**
  - User programs ask the kernel for access to resources.
  - Some Unix programs consist of separate programs that share or exchange data.

- **Writing systems programs requires an understanding of the structure and use of kernel services.**