Users, Files, and the Manual :

`who`

류은경
ekryu@knu.ac.kr

Objectives

- **Ideas and Skills**
  - The role and use of on-line documentation
  - The Unix file interface: **`open`, `read`, `write`, `lseek`, `close`**
  - Reading, creating, and writing files
  - File descriptors
  - Buffering: user level and kernel level
  - Kernel mode, user mode, and the cost of system calls
  - How Unix represents time, how to format Unix time
  - Using the **`utmp`** file to find list of current users
  - Detecting and reporting errors in system calls
- **System Calls and Functions**
  - `open`, `read`, `write`, `create`, `lseek`, `close`
  - `perror`
- **Commands**
  - `man`, `who`, `cp`, `login`

2.2 Asking about `who`

2.3 What Does `who` Do?

2.4 How Does `who` Do It?

2.5 Can I Write `who`?

2.6 Writing `cp` (read and write)

2.7 More Efficient File I/O: Buffering

2.8 Buffering and the Kernel

2.10 What to Do with System-Call Errors

Commands are Programs

- Almost all Unix **commands** are simply **programs** written by a variety of people, usually in C.

- **Adding new commands** to Unix:
  - You write a new program and have the executable file stored in one of the standard system directories: `/bin`, `/usr/bin`, `/usr/local/bin`.

ls
cd
pwd
cp
mv
rm
mkdir
touch
man
echo
…

Q1: What Does `who` Do?

```
$ who
heckerl     ttyp1       Jul 21 19:51    (tide75.surfcity.com)
nlopez      ttyp2       Jul 21 18:11    (roam163-141.student.ivy.edu)
dgsulliv    ttyp3       Jul 21 14:18    (h004005a8bd64.ne.mediaone.net)
ackerman    ttyp4       Jul 15 22:40    (asd1-254.fas.state.edu)
wwchen      ttyp5       Jul 21 19:57    (circle.square.edu)
barbier     ttyp6       Jul  8 13:08    (labpc18.elsie.special.edu)
ramakris    ttyp7       Jul 13 08:51    (roam157-97.student.ivy.edu)
czhu        ttyp8       Jul 21 12:47    (spa.sailboat.edu)
bpsteven    ttyp9       Jul 21 18:26    (207.178.203.99)
molay       ttypa       Jul 21 20:00    (xyz73-200.harvard.edu)
$
```

one log-in session

username

terminal name

login time

login host

Reading the Manual

```
$ man who
who(1)

NAME

  who - Identifies users currently logged in

SYNOPSIS

  who [-a]  |[-AbdhHlmMpqrstTu] [file]

  who am i

  who am I

  whoami
    ...
```

Q2: How Does `who` Do It?

- **To learn more about Unix from Unix**
  - Read the manual
  - Search the manual
  - Read the `.h` files
  - Follow `SEE ALSO` links

Read the Manual

$man who

DESCRIPTION

The who utility can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current UNIX system user. It examines the /var/adm/utmp file to obtain its information. If file is given, that file (which must be in utmp(4) format) is examined. Usually, file will be /var/adm/wtmp, which contains a history of all the logins since the file was last created.

Search the Manual

- **`$ man -k utmp`**

```
endutent        getutent (3c)   - access utmp file entry
endutxent       getutxent (3c)  - access utmpx file entry
getutent        getutent (3c)   - access utmp file entry
getutid         getutent (3c)   - access utmp file entry
getutline       getutent (3c)   - access utmp file entry
getutmp         getutxent (3c)  - access utmpx file entry
getutmpx        getutxent (3c)  - access utmpx file entry
getutxent       getutxent (3c)  - access utmpx file entry
getutxid        getutxent (3c)  - access utmpx file entry
getutxline      getutxent (3c)  - access utmpx file entry
pututline       getutent (3c)   - access utmp file entry
pututxline      getutxent (3c)  - access utmpx file entry
setutent        getutent (3c)   - access utmp file entry
setutxent       getutxent (3c)  - access utmpx file entry
ttyslot         ttyslot (3c)    - find the slot in the utmp file of the
                                  current user
updwtmp         getutxent (3c)  - access utmpx file entry
updwtmpx        getutxent (3c)  - access utmpx file entry
utmp            utmp (4)        - utmp and wtmp entry formats
utmp2wtmp       acct (1m)       - overview of accounting and
                                  miscellaneous accounting commands
utmpd           utmpd (1m)      - utmp and utmpx monitoring daemon
utmpname        getutent (3c)   - access utmp file entry
utmpx           utmpx (4)       - utmpx and wtmpx entry formats
utmpxname       getutxent (3c)  - access utmpx file entry
wtmp            utmp (4)        - utmp and wtmp entry formats
wtmpx           utmpx (4)       - utmpx and wtmpx entry formats
$
```

$ **man 4 utmp**

NAME

utmp, wtmp - Login records

SYNOPSIS

```
#include <utmp.h>
```

DESCRIPTION

The utmp file records information about who is currently using the system.
The file is a sequence of utmp entries, as defined in struct utmp in the utmp.h file.

The utmp structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of time(3). The ut_type field is the type of entry, which can specify several symbolic constant values. The symbolic constants are defined in the utmp.h file.

The wtmp file records all logins and logouts. A null user name indicates a logout on the associated terminal. A terminal referenced with a tilde (~) indicates that the system was rebooted at the indicated time. The adjacent pair of entries with terminal names referenced by a vertical bar (|) or a right brace (}) indicate the system-maintained time just before and just after a date command has changed the system's time frame.

The wtmp file is maintained by login(1) and init(8). Neither of these pro-grams creates the file, so, if it is removed, record keeping is turned off. See ac(8) for information on the file.

FILES

/usr/include/utmp.h

/var/adm/utmp

more (88%)

Read the .h files

- **$ more /usr/include/utmp.h**

```
#define UTMP_FILE       "/var/adm/utmp"
#define WTMP_FILE       "/var/adm/wtmp"

#include <sys/types.h>  /* for pid_t, time_t */

/*
 * Structure of utmp and wtmp files.
 *
 * Assuming these numbers is unwise.
 */

#define ut_name ut_user                 /* compatibility */
struct utmp {
        char    ut_user[32];            /* User login name */
        char    ut_id[14];              /* /etc/inittab id- IDENT_LEN in
                                         * init */
        char    ut_line[32];            /* device name (console, lnxx) */
        short   ut_type;                /* type of entry */
        pid_t   ut_pid;                 /* process id */
        struct exit_status {
            short       e_termination;  /* Process termination status */
            short       e_exit;         /* Process exit status */
        } ut_exit;                      /* The exit status of a process
                                         * marked as DEAD_PROCESS.
                                         */
        time_t  ut_time;                /* time entry was made */
        char    ut_host[64];            /* host name same as
                                         * MAXHOSTNAMELEN */
};
/* Definitions for ut_type                                          */
```

utmp.h (60%)

```
struct utmp {
    short   ut_type;              /* Type of record */
    pid_t   ut_pid;               /* PID of login process */
    char    ut_line[UT_LINESIZE]; /* Device name of tty - "/dev/" */
    char    ut_id[4];             /* Terminal name suffix,
                                     or inittab(5) ID */
    char    ut_user[UT_NAMESIZE]; /* Username */
    char    ut_host[UT_HOSTSIZE]; /* Hostname for remote login, or
                                     kernel version for run-level
                                     messages */
    struct  exit_status ut_exit;  /* Exit status of a process
                                     marked as DEAD_PROCESS; not
                                     used by Linux init (1 */
    /* The ut_session and ut_tv fields must be the same size when
       compiled 32- and 64-bit.  This allows data files and shared
       memory to be shared between 32- and 64-bit applications. */
#if __WORDSIZE == 64 && defined __WORDSIZE_COMPAT32
    int32_t ut_session;           /* Session ID (getsid(2)),
                                     used for windowing */
    struct {
        int32_t tv_sec;           /* Seconds */
        int32_t tv_usec;          /* Microseconds */
    } ut_tv;                      /* Time entry was made */
#else
     long   ut_session;           /* Session ID */
     struct timeval ut_tv;        /* Time entry was made */
#endif

    int32_t ut_addr_v6[4];        /* Internet address of remote
                                     host; IPv4 address uses
                                     just ut_addr_v6[0] */
    char __unused[20];            /* Reserved for future use */
};
```

$ man 5 utmp

$ utmpdump /var/run/utmp

How who works

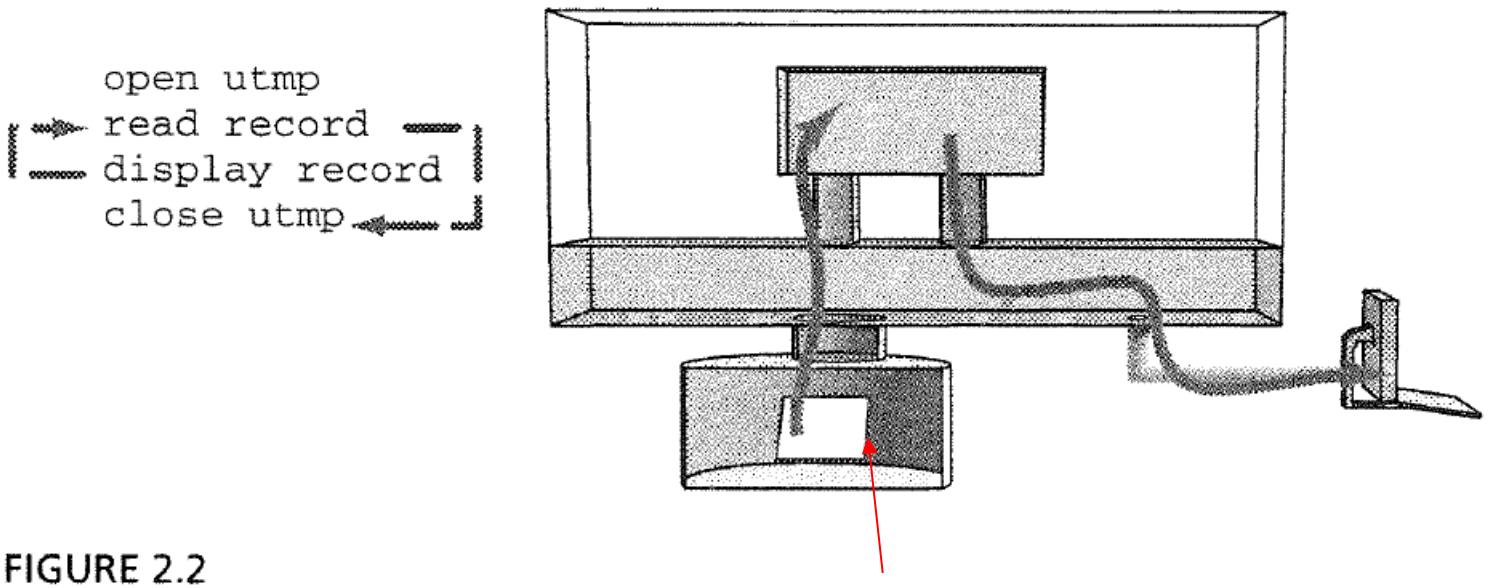


FIGURE 2.2

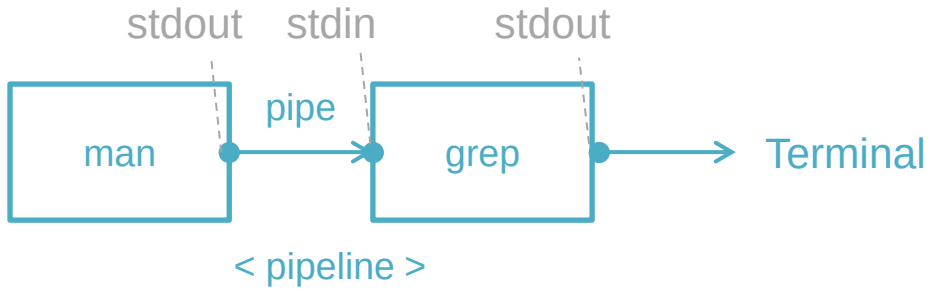
Data flow in the who command.

Can I Write `who`?

- ***Two tasks* we need to program**
  - Read structs from a file
  - Display the information stored in a struct

Q: How do I read structs from a file?

- **Let's Read the Manual!**



pipe

```
$ man -k file | grep read
_llseek (2)           - reposition read/write file offset
fileevent (n)         - Execute a script when a channel becomes readable
                        or writable
gftype (1)            - translate a generic font file for humans to read
lseek (2)             - reposition read/write file offset
macsave (1)           - Save Mac files read from standard input
read (2)              - read from a file descriptor
readprofile (1)       - a tool to read kernel profiling information
scr_dump, scr_restore, scr_init, scr_set (3) - read (write) a curses
screen from (to) a file
tee (1)               - read from standard input and write to standard
                        output and files
$
```

```
$ man 2 read
READ(2)                          System calls                          READ(2)

NAME
       read - read from a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t read(int fd, void *buf, size_t count);

DESCRIPTION
       read() attempts to read up to count bytes from file
       descriptor fd into the buffer starting at buf.


RELATED INFORMATION (called SEE ALSO in some versions)

  Functions: fcntl(2), creat(2), dup(2), ioctl(2), getmsg(2), lockf(3),
  lseek(2), mtio(7), open(2), pipe(2), poll(2), socket(2), socketpair(2),
  termios(4), streamio(7), opendir(3) lockf(3)

  Standards: standards(5)
```

ANS: we use `open`, `read`, and `close`

- **Opening a file: `open()`**
  - It is a **kernel service;** system call

```
fd = open(name, mode)
          ↑      ↑
        char *   O_RDONLY
                 O_WRONLY
                 O_RDWR

returns -1 on error
OR an int on success
```

```c
int fd = open("file.txt", O_RDONLY);
if (fd == -1) {
    perror("open");
    return -1;
}

close(fd);
```

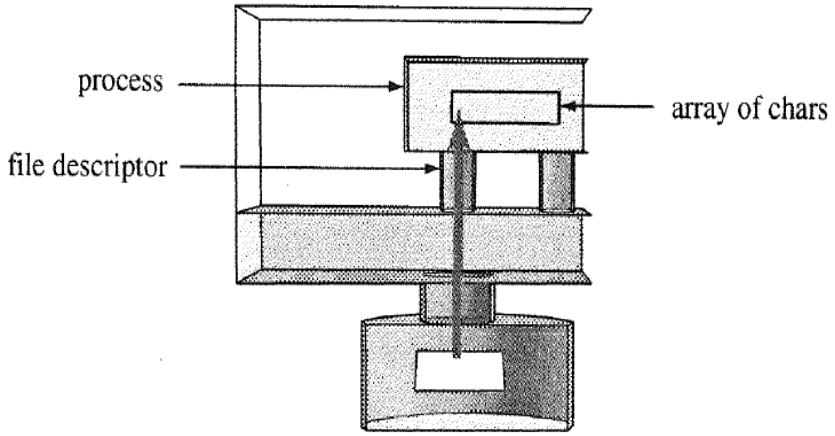


FIGURE 2.3

A file descriptor is a connection to a file.

```c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main() {
    int fd = open("file_open.c", O_RDONLY);
    if (fd == -1) return 1;

    char buffer[100];
    ssize_t bytesRead = read(fd, buffer, sizeof(buffer) - 1);

    if (bytesRead > 0) {
        buffer[bytesRead] = '\0';
        printf("%s\n", buffer);
    }

    close(fd);
    return 0;
}
```

file_open.c

open	
PURPOSE	Creates a connection to a file
INCLUDE	#include <fcntl.h>
USAGE	int fd = open(char *name, int how)
ARGS	name — name of file how — O_RDONLY, O_WRONLY, or O_RDWR
RETURNS	-1 — on error int — on success

read	
PURPOSE	Transfer up to qty bytes from fd to buf
INCLUDE	#include <unistd.h>
USAGE	ssize_t numread = read(int fd, void *buf, size_t qty)
ARGS	fd — source of data buf — destination for data qty — number of bytes to transfer
RETURNS	-1 — on error numread — on success

close	
PURPOSE	Closes a file
INCLUDE	#include <unistd.h>
USAGE	int result = close(int fd)
ARGS	fd — file descriptor
RETURNS	-1 — on error 0 — on success

Writing `who1.c`

```
/* who1.c  - a first version of the who program
 *              open, read UTMP file, and show results
 */
#include        <stdio.h>
#include        <utmp.h>
#include        <fcntl.h>
#include        <unistd.h>
#incldue        <stdlib.h>
#define SHOWHOST        /* include remote machine on output */
void show_info( struct utmp* );
int main()
{
        struct utmp     current_record; /* read info into here       */
        int             utmpfd;         /* read from this descriptor */
        int             reclen = sizeof(current_record);

        if ( (utmpfd = open(UTMP_FILE, O_RDONLY)) == -1 ){
                perror( UTMP_FILE );    /* UTMP_FILE is in utmp.h    */
                exit(1);
        }
        while ( read(utmpfd, &current_record, reclen) == reclen )
                show_info(&current_record);
        close(utmpfd);
        return 0;                       /* went ok */
}
```

```
/*
 *  show info()
 *      displays contents of the utmp struct in human readable form
 *      *note* these sizes should not be hardwired
 */
void show_info( struct utmp* utbufp);
{
        printf("%-8.8s", utbufp->ut_name);      /* the logname  */
        printf(" ");                            /* a space      */
        printf("%-8.8s", utbufp->ut_line);      /* the tty      */
        printf(" ");                            /* a space      */
        printf("%10ld", utbufp->ut_time);       /* login time   */
        printf(" ");                            /* a space      */
#ifdef  SHOWHOST
        printf("(%s)", utbufp->ut_host);        /* the host     */
#endif
        printf("\n");                           /* newline      */
}
```

- **Compile and run it:**

```
$ cc who1.c -o who1
$ who1                    $ ./who1
          system b  952601411 ()
          run-leve  952601411 ()
                    952601416 ()
                    952601416 ()
                    952601417 ()
                    952601417 ()
                    952601419 ()
                    952601419 ()
                    952601423 ()
                    952601566 ()
LOGIN     console   952601566 ()
          ttyp1     958240622 ()
shpyrko   ttyp2     964318862 (nas1-093.gas.swamp.org)
acotton   ttyp3     964319088 (math-guest04.williams.edu)
          ttyp4     964320298 ()
spradlin  ttyp5     963881486 (h002078c6adfb.ne.rusty.net)
dkoh      ttyp6     964314388 (128.103.223.110)
spradlin  ttyp7     964058662 (h002078c6adfb.ne.rusty.net)
king      ttyp8     964279969 (blade-runner.mit.edu)
berschba  ttyp9     964188340 (dudley.learned.edu)
rserved   ttypa     963538145 (gigue.eas.ivy.edu)
dabel     ttypb     964319455 (roam193-27.student.state.edu)
          ttypc     964319645 ()
```

```
2007200_40634@Ubuntu-023:~/다운로드$ ./a.out
reboot   ~         1725334869 (5.15.0-94-generic)
runlevel ~         1725334877 (5.15.0-94-generic)
2007200  127.0.0. 1725511855 (127.0.0.1:100)
```

```
2007200_40634@Ubuntu-023:~/다운로드$ ./a.out
reboot   ~         1725334869 (5.15.0-94-generic)
runlevel ~         1725334877 (5.15.0-94-generic)
2007200  127.0.0.  1725511855 (127.0.0.1:100)
```

```
2007200_40634@Ubuntu-023:~/다운로드$ who
2007200_40634 127.0.0.1:100 2024-09-05 13:50 (127.0.0.1:100)
```

Comparison

```
$ who
shpyrko     ttyp2     Jul 22 22:21     (nas1-093.gas.swamp.edu)
acotton     ttyp3     Jul 22 22:24     (math-guest04.williams.edu)
spradlin    ttyp5     Jul 17 20:51     (h002078c6adfb.ne.rusty.net)
dkoh        ttyp6     Jul 22 21:06     (128.103.223.110)
spradlin    ttyp7     Jul 19 22:04     (h002078c6adfb.ne.rusty.net)
king        ttyp8     Jul 22 11:32     (blade-runner.mit.edu)
berschba    ttyp9     Jul 21 10:05     (dudley.learned.edu)
rserved     ttypa     Jul 13 21:29     (gigue.eas.ivy.edu)
dabel       ttypb     Jul 22 22:30     (roam193-27.student.state.edu)
rserved     ttypd     Jul 13 21:31     (gigue.eas.harvard.edu)
dkoh        ttype     Jul 22 16:46     (128.103.223.110)
molay       ttyq0     Jul 22 20:03     (xyz73-200.harvard.edu)
cweiner     ttyq8     Jul 21 16:40     (roam175-157.student.stats.edu)
$
```

What We Need to Do

- Suppress blank records
- Get the log-in times correct

Writing who2.c

- **Suppressing blank records**
  - /usr/include/utmp.h

```
/*      Definitions for ut_type                                 */

#define EMPTY           0
#define RUN_LVL         1
#define BOOT_TIME       2
#define OLD_TIME        3
#define NEW_TIME        4
#define INIT_PROCESS    5       /* Process spawned by "init" */
#define LOGIN_PROCESS   6       /* A "getty" process waiting for login */
#define USER_PROCESS    7       /* A user process */
#define DEAD_PROCESS    8
```

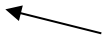
※ represents the user logged into the system.

- **Modification :**

```
show_info( struct utmp *utbufp )
{
        if ( utbufp->ut_type != USER_PROCESS )      /* users only ! */
                return;
        printf("%-8.8s", utbufp->ut_name);          /* the username  */
```

Displaying Log-in Time in Human-Readable Form

```
$ man -k time

$ man -k time | grep transform
$ man -k time | grep -i convert
```



the **-i** option :
a *case-insensitive* match

- *How unix stores times*: **`time_t`**
- *Making a* **`time_t`** *readable*: **`ctime`**

$ **man 3 ctime**

CTIME(3) Linux Programmer's Manual CTIME(3)

NAME

asctime, ctime, gmtime, localtime, mktime - transform binary date and time to ASCII

SYNOPSIS

```
#include <time.h>

char *asctime(const struct tm *timeptr);

char *ctime(const time_t *timep);
```

...

The ctime() function converts the calendar time timep into a string of the form

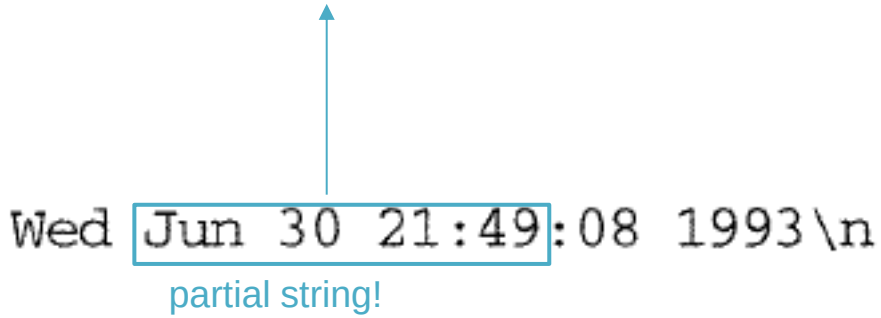
```
"Wed Jun 30 21:49:08 1993\n"
```

- **Putting it All together**

```
/* who2.c  -  read /var/adm/utmp and list info therein
 *         - suppresses empty records
 *         - formats time nicely
 */
#include        <stdio.h>
#include        <unistd.h>
#include        <utmp.h>
#include        <fcntl.h>
#include        <time.h>
#incldue         <stdlib.h>
/* #define      SHOWHOST */

void showtime(long);
void show_info(struct utmp *);

int main()
{
        struct utmp     utbuf;          /* read info into here */
        int             utmpfd;         /* read from this descriptor */
```

```
        if ( (utmpfd = open(UTMP_FILE, O_RDONLY)) == -1 ){
                perror(UTMP_FILE);
                exit(1);
        }

        while( read(utmpfd, &utbuf, sizeof(utbuf)) == sizeof(utbuf) )
                show_info( &utbuf );
        close(utmpfd);
        return 0;
}
/*
 *      show info()
 *                      displays the contents of the utmp struct
 *                      in human readable form
 *                      * displays nothing if record has no user name
 */
void show_info( struct utmp *utbufp )
{
        if ( utbufp->ut_type != USER_PROCESS )
                return;

        printf("%-8.8s", utbufp->ut_name);      /* the logname  */
        printf(" ");                            /* a space      */
        printf("%-8.8s", utbufp->ut_line);      /* the tty      */
        printf(" ");                            /* a space      */
        showtime( utbufp->ut_time );            /* display time */
#ifdef SHOWHOST
        if ( utbufp->ut_host[0] != '\0' )
                printf(" (%s)", utbufp->ut_host);/* the host    */
#endif
        printf("\n");                           /* newline      */
}
```

```
void showtime( long timeval )
/*
 *      displays time in a format fit for human consumption
 *      uses ctime to build a string then picks parts out of it
 *      Note: %12.12s prints a string 12 chars wide and LIMITS
 *      it to 12chars.
 */
{
        char    *cp;                    /* to hold address of time      */
        cp = ctime(&timeval);           /* convert time to string       */
                                        /* string looks like            */
                                        /* Mon Feb  4 00:46:40 EST 1991 */
                                        /* 0123456789012345.            */
        printf("%12.12s", cp+4 );       /* pick 12 chars from pos 4     */
}
```



Testing who2.c

```
$ cc who2.c -o who2
$ who2                  $ ./who2
rlscott  ttyp2    Jul 23 01:07
acotton  ttyp3    Jul 22 22:24
spradlin ttyp5    Jul 17 20:51
spradlin ttyp7    Jul 19 22:04
king     ttyp8    Jul 22 11:32
berschba ttyp9    Jul 21 10:05
rserved  ttypa    Jul 13 21:29
rserved  ttypd    Jul 13 21:31
molay    ttyq0    Jul 22 20:03
cweiner  ttyq8    Jul 21 16:40
mnabavi  ttyx2    Apr 10 23:11
$ who
rlscott      ttyp2        Jul 23 01:07
acotton      ttyp3        Jul 22 22:24
spradlin     ttyp5        Jul 17 20:51
spradlin     ttyp7        Jul 19 22:04
king         ttyp8        Jul 22 11:32
berschba     ttyp9        Jul 21 10:05
rserved      ttypa        Jul 13 21:29
rserved      ttypd        Jul 13 21:31
molay        ttyq0        Jul 22 20:03
cweiner      ttyq8        Jul 21 16:40
mnabavi      ttyx2        Apr 10 23:11
$
```

```
2007200_40634@Ubuntu-023:~/다운로드$ ./a.out
reboot   ~          1725334869 (5.15.0-94-generic)
runlevel ~          1725334877 (5.15.0-94-generic)
2007200_ 127.0.0.   1725511855 (127.0.0.1:100)
```

```
2007200_40634@Ubuntu-023:~/다운로드$ ./a.out
2007200_ 127.0.0.   Sep  5 13:50 (127.0.0.1:100)
```

Q1: What does `cp` do?

- **`cp` makes a copy of a file**

```
$ cp source-file target-file
```

  - If there is no target file, `cp` *creates* it.
  - If there is a target file, `cp` *replaces* the contents of that file with the contents of the source file.

Q2: How Does cp Create and Write?

- **Creating/Truncating a File:**

```
fd = creat(name, 0644);
```

open("file.txt", O_CREAT | O_WRONLY | O_TRUNC, 0644);

creat

PURPOSE	Create or zero a file	
INCLUDE	#include <fcntl.h>	
USAGE	int fd = creat(char *filename, mode_t mode)	
ARGS	filename:	the name of the file
	mode:	access permission
RETURNS	-1	on error
	fd	on success

- **Writing to a File**

```
n = write(fd, buffer, num);
```

write		
PURPOSE	Send data from memory to a file	
INCLUDE	#include <unistd.h>	
USAGE	ssize_t result = write(int fd, void *buf, size_t amt)	
ARGS	fd	a file descriptor
	buf	an array
	amt	how many bytes to write
RETURNS	-1	on error
	num written	on success

Q3: Can I Write cp?

- **Program outline**

```
    open sourcefile for reading
    open copyfile   for writing
+-> read from source to buffer -- eof? -+
|__ write from buffer to copy           |
                                        |
    close sourcefile       <------------+
    close copyfile
```

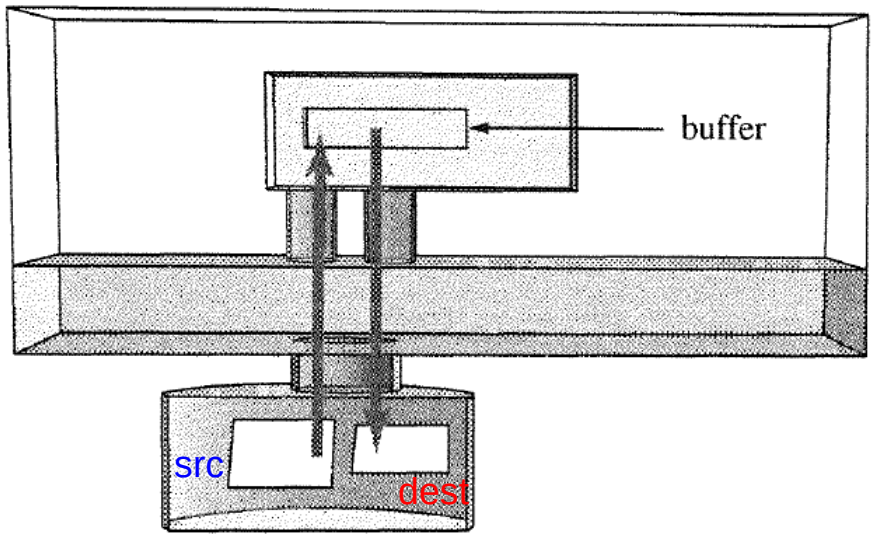


FIGURE 2.4

Copying a file by reading and writing.

```
/** cp1.c
 *      version 1 of cp - uses read and write with tunable buffer size
 *
 *      usage: cp1 src dest
 */
#include        <stdio.h>
#include        <unistd.h>
#include        <fcntl.h>
#incldue        <stdlib.h>
#define BUFFERSIZE      4096
#define COPYMODE        0644

void oops(char *, char *);

main(int ac, char *av[])
{
        int     in_fd, out_fd, n_chars;
        char    buf[BUFFERSIZE];
                                                /* check args   */
        if ( ac != 3 ){
                fprintf( stderr, "usage: %s source destination\n", *av);
                exit(1);
        }
                                                /* open files   */
        if ( (in_fd=open(av[1], O_RDONLY)) == -1 )
                oops("Cannot open ", av[1]);

        if ( (out_fd=creat( av[2], COPYMODE)) == -1 )
                oops( "Cannot creat", av[2]);
                                                /* copy files   */
```

```
        while ( (n_chars = read(in_fd , buf, BUFFERSIZE)) > 0 )
                if ( write( out_fd, buf, n_chars ) != n_chars )
                        oops("Write error to ", av[2]);
        if ( n_chars == -1 )
                        oops("Read error from ", av[1]);

                                                /* close files  */

        if ( close(in_fd) == -1 || close(out_fd) == -1 )
                oops("Error closing files","");
}

void oops(char *s1, char *s2)
{
        fprintf(stderr,"Error: %s ", s1);
        perror(s2);
        exit(1);
}
```

```
$ cc cp1.c -o cp1
$ cp1 cp1 copy.of.cp1    ※ $ ./cp1 cp1 copy.of.cp1
$ ls -l cp1 copy.of.cp1
-rw-r--r--   1 bruce    bruce       37419 Jul 23 03:12 copy.of.cp1
-rwxrwxr-x   1 bruce    bruce       37419 Jul 23 03:08 cp1
$ cmp cp1 copy.of.cp1    ※ $ man cmp
$
```

2.2 Asking about who

2.3 What Does who Do?

2.4 How Does who Do It?

2.5 Can I Write who?

2.6 Writing cp (read and write)

2.7 More Efficient File I/O: Buffering

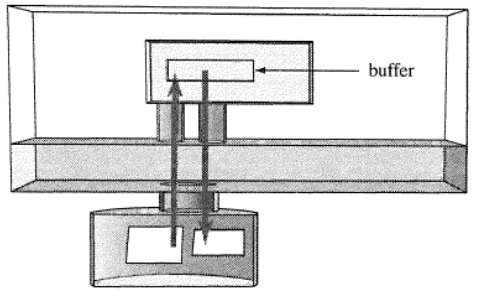
2.8 Buffering and the Kernel

2.10 What to Do with System-Call Errors

Does the Size of the Buffer Matter?

- **Execution time for cp1**
  - read(in_fd, buf, BUFFERSIZE);

buffersize	execution time in seconds
1	50.29
4	12.81
16	3.28
64	0.96
128	0.56
256	0.37
512	0.27
1024	0.22
2048	0.19
4096	0.18
8192	0.18
16384	0.18




```
Ex: Filesize = 2500 bytes

If buffer = 100 bytes then
copy requires 25 read() and 25 write() calls

If buffer = 1000 bytes then
copy requires 3 read() and 3 write() calls
```

Why System Calls are Time Consuming?

- **Not only does transferring data take time but Mode Change takes time**
  - It runs various Kernel functions
  - It also requires a shift from USER MODE to KERNEL MODE and back;

- Thus, try to **minimize system calls**

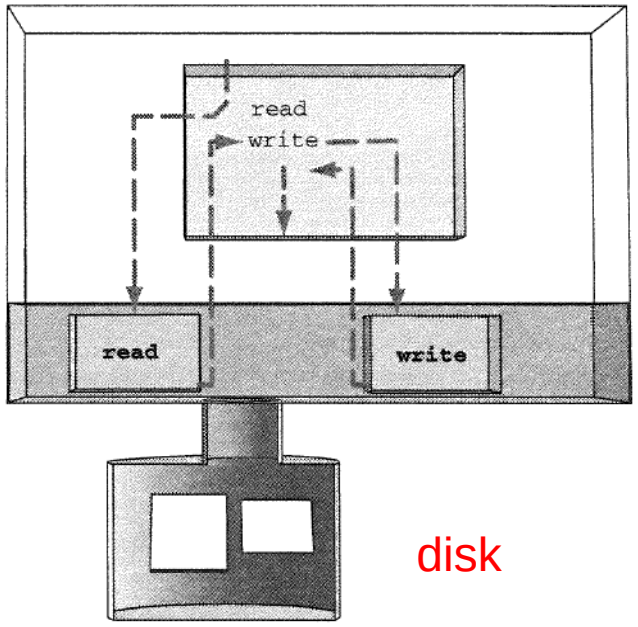


FIGURE 2.5
Control flow during system calls.

who2.c is Inefficient?

- **who2.c use one system call for each utmp record**

```c
struct utmp utbuf;                          /* read info into here */
int utmpfd;                                 /* read from this descriptor */

if( ( utmpfd = open(UTMP_FILE, O_RDONLY)) == -1 ) {
    perror( UTMP_FILE );                    /* UTMP_FILE is in utmp.h */
    exit(1);
}

while( read(utmpfd, &utbuf, sizeof(utbuf)) ) == sizeof(utbuf) )
    show_info(&utbuf);
close(utmpfd);
```

  - A beffer idea: …

Adding Buffering : who3.c

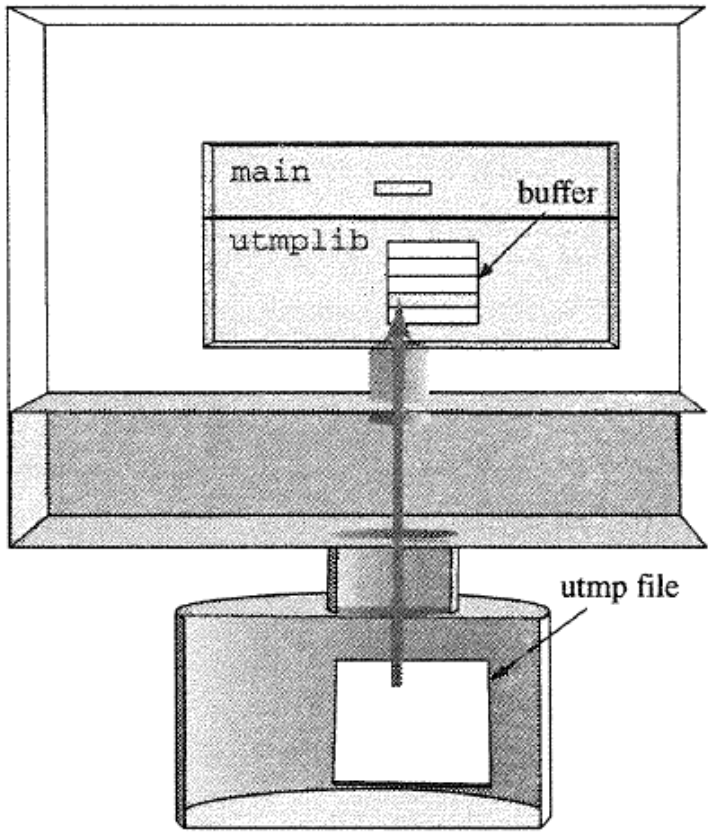


FIGURE 2.6

Buffering disk data in user space.

File buffering with utmplib

`main` calls a function in `utmplib.c` to get the next struct utmp.

Functions in `utmplib.c` read structs 16 at a time from the disk into an array.

The kernel is called only when all 16 are used up.

```
$ gcc who3.c umtplib.c -o who3
$ ./who3
```

```
/* who3.c - who with buffered reads
 *        - surpresses empty records
 *        - formats time nicely
 *        - buffers input (using utmplib)
 */
#include        <stdio.h>
#include        <sys/types.h>
#include        <utmp.h>
#include        <fcntl.h>
#include        <time.h>
#incldue        <stdlib.h>
#define SHOWHOST

void show_info(struct utmp *);
void showtime(time_t);

int main()
{
        struct utmp     *utbufp,        /* holds pointer to next rec    */
                        *utmp_next();   /* returns pointer to next      */

        if ( utmp_open( UTMP_FILE ) == -1 ){
                perror(UTMP_FILE);
                exit(1);
        }
        while ( ( utbufp = utmp_next() ) != ((struct utmp *) NULL) )
                show_info( utbufp );
        utmp_close( );
        return 0;
}
/*
 *      show info()
 ...
```

```
struct utmp utbuf;
int         utmpfd;

if( ( utmpfd = open(UTMP_FILE, O_RDONLY)) == -1 ) {
        perror( UTMP_FILE );
        exit(1);
}

while( read(utmpfd, &utbuf, sizeof(utbuf) ) == sizeof(utbuf) )
        show_info(&utbuf);
close(utmpfd);
```

```
/* utmplib.c  - functions to buffer reads from utmp file
 *
 *       functions are
 *              utmp_open( filename )    - open file
 *                      returns -1 on error
 *              utmp_next( )             - return pointer to next struct
 *                      returns NULL on eof
 *              utmp_close()             - close file
 *
 *       reads NRECS per read and then doles them out from the buffer
 */
```

```c
#include        <stdio.h>
#include        <fcntl.h>
#include        <sys/types.h>
#include        <utmp.h>

#define NRECS   16
#define NULLUT  ((struct utmp *)NULL)
#define UTSIZE  (sizeof(struct utmp))

static  char    utmpbuf[NRECS * UTSIZE];                /* storage      */
static  int     num_recs;                               /* num stored   */
static  int     cur_rec;                                /* next to go   */
static  int     fd_utmp = -1;                           /* read from    */

utmp_open( char *filename )
{
        fd_utmp = open( filename, O_RDONLY );           /* open it      */
        cur_rec = num_recs = 0;                         /* no recs yet  */
        return fd_utmp;                                 /* report       */
}

struct utmp *utmp_next()
{
        struct utmp *recp;
        if ( fd_utmp == -1 )                            /* error ?      */
                return NULLUT;
        if ( cur_rec==num_recs && utmp_reload()==0 )    /* any more ?   */
                return NULLUT;
                                /* get address of next record    */
        recp = ( struct utmp *) &utmpbuf[cur_rec * UTSIZE];
        cur_rec++;
        return recp;
}
```

```
int utmp_reload()
/*
 *      read next bunch of records into buffer
 */
{
        int     amt_read;
                                                /* read them in         */
        amt_read = read( fd_utmp , utmpbuf, NRECS * UTSIZE );

                                                /* how many did we get? */
        num_recs = amt_read/UTSIZE;
                                                /* reset pointer        */
        cur_rec  = 0;
        return num_recs;
}



utmp_close()
{
        if ( fd_utmp != -1 )                    /* don't close if not   */
                close( fd_utmp );               /* open                 */
}
```

If Buffering Is So Smart, Why Doesn't the Kernel Do It?

- **It does!**
  - **Kernel buffer** keeps copies of disk block in memory
  - The **`read()`** call copies data into a process **from a kernel buffer**
  - The **`write()`** copies data from the process **to a kernel buffer**
  - if not in a kernel buffer? …

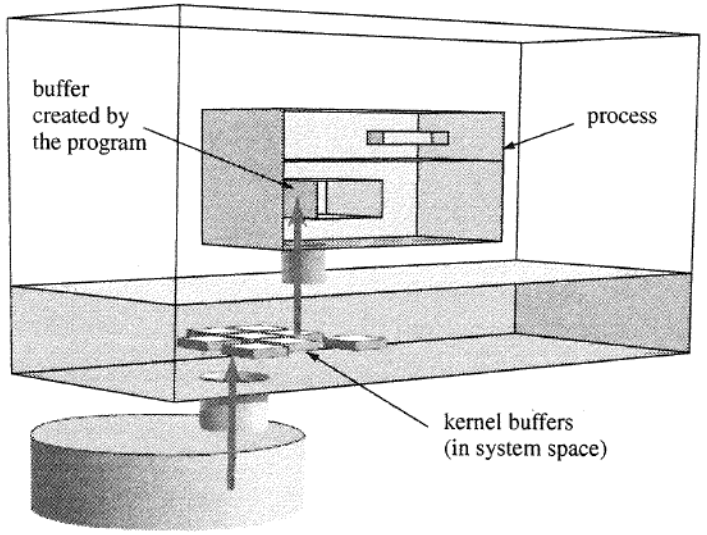


FIGURE 2.7

Buffering disk data in the kernel.

Consequences of Kernel Buffering

- Faster "disk" I/O
- Optimized disk writes
- Need to write buffers to disk before shutdown

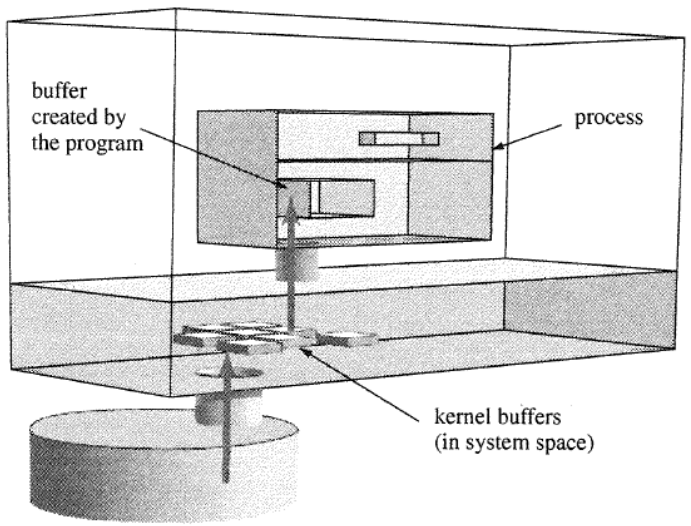


FIGURE 2.7

Buffering disk data in the kernel.

What to Do with System-Call Errors

- **System Calls and Errors:**
  - `open`, `read`, and `lseek` return -1 when an error occurs.

- **How to identify what went wrong : `errno`**
  - The kernel tells your program the cause of the error *by storing an* **global variable** *called* ***`errno`***.
  - Every program contains this variable.

- **Common error codes include:**

  `$ man 3 errno`

```
#define EPERM           1       /* Operation not permitted */
#define ENOENT          2       /* No such file or directory */
#define ESRCH           3       /* No such process */
#define EINTR           4       /* Interrupted system call */
#define EIO             5       /* I/O error */
```

```c
#include  <errno.h>

extern int errno;

int sample()
{
    int fd;
    fd = open("file", O_RDONLY);
    if ( fd == -1 )
    {
        printf("Cannot open file: ");
        if ( errno == ENOENT )
            printf("There is no such file.");
        else if ( errno == EINTR )
            printf("Interrupted while opening file.");

        else if ( errno == EACCESS )
            printf("You do not have permission to open file.");
        ...
```

```c
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>

int main() {
    int fd;

    fd = open("nonexistentfile.txt", O_RDONLY);

    if (fd == -1) {
        printf("Error %d: %s\n", errno, strerror(errno));
    } else {
        printf("File opened.\n");
    }

    return 0;
}
```

```
2007200_40634@Ubuntu-037:~/다운로드$ ./a.out
Error 2: No such file or directory
2007200_40634@Ubuntu-037:~/다운로드$
```

- **Reporting Errors: $ man 3 perror**
  - Print a system error message

```
void perror(const char *s);
char *strerror(int errnum);
```

```
int sample()
{
    int fd;
    fd = open("file", O_RDONLY);
    if ( fd == -1 )
    {
        perror("Cannot open file");
        return;
    }
    ...
```

Users, Files, and the Manual

2.2 Asking about **`who`**

2.3 What Does `who` Do?

2.4 How Does `who` Do It?

2.5 Can I Write who?

2.6 Writing **`cp`** (read and write)

2.7 More Efficient File I/O: **Buffering**

2.8 Buffering and the Kernel

2.10 What to Do with **System-Call Errors**