Users, Files, and the Manual
File I/O

Prof. Seokin Hong

Kyungpook National University

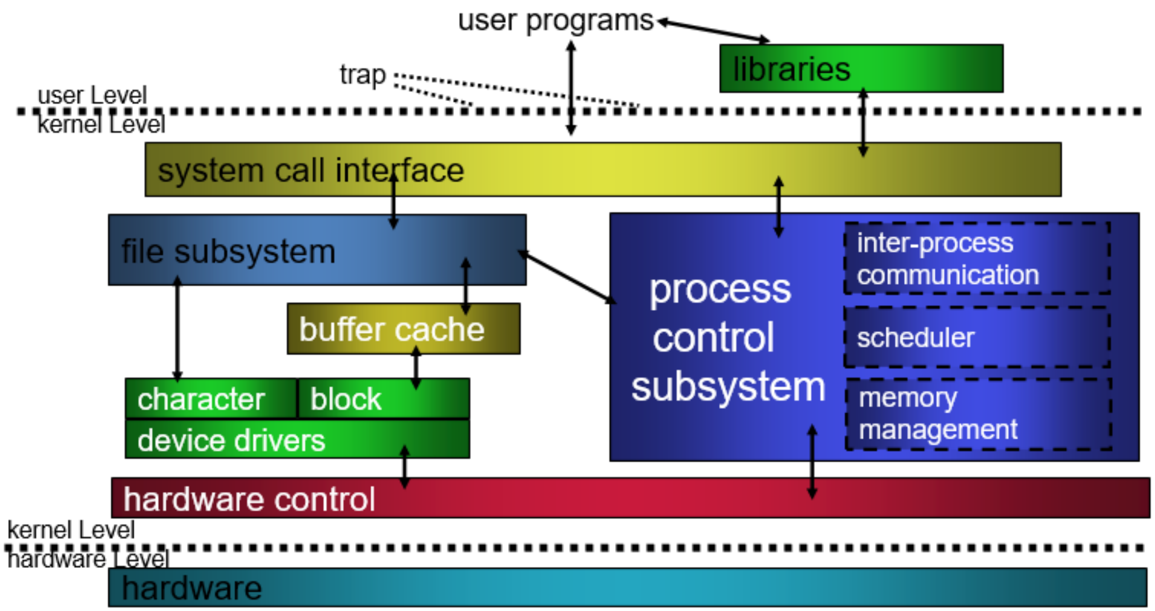
Fall 2019

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
- **UNIX/LINUX Commands**
  - man, who, cp, login

System Call

- System calls are an operating system's API
  - The set of functions that the operating system exposes to processes
- If you want to the OS to do something, you tell it via a system call



What are System Calls used for?

- Anything to do with
  - Accessing devices
  - Accessing files
  - Requesting memory
  - Setting/Changing access permissions
  - Communicating with other processes
  - Stopping/starting processes
  - Setting a timer

- You need a system call to
  - **Open a file**
  - Get data from the network
  - Kill a process

Three step to learn Unix/Linux system programming

- Looking at “real” programs : What does that do?
- Looking at the system calls : How does that work?
- Writing our own version : Can I try to do it?

Agenda

- 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
- 2.9 Reading and Writing a File
- 2.10 What to Do with System-Call Errors

Asking About **who**

- What does **who** do?
- How does **who** work?
- Can I write **who**?

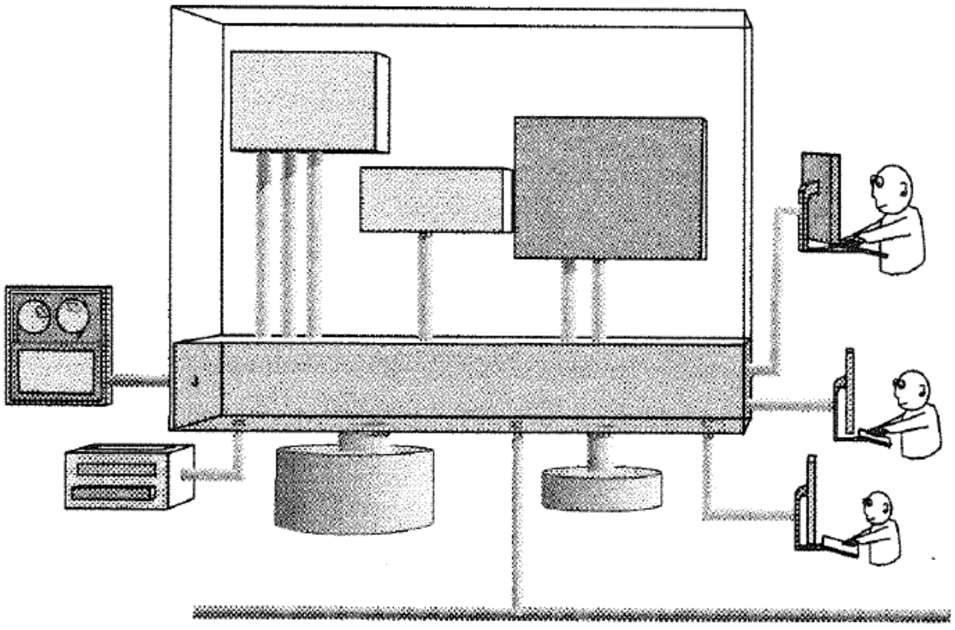


FIGURE 2.1

Users, files, processes, devices, and kernel.

Commands are Programs

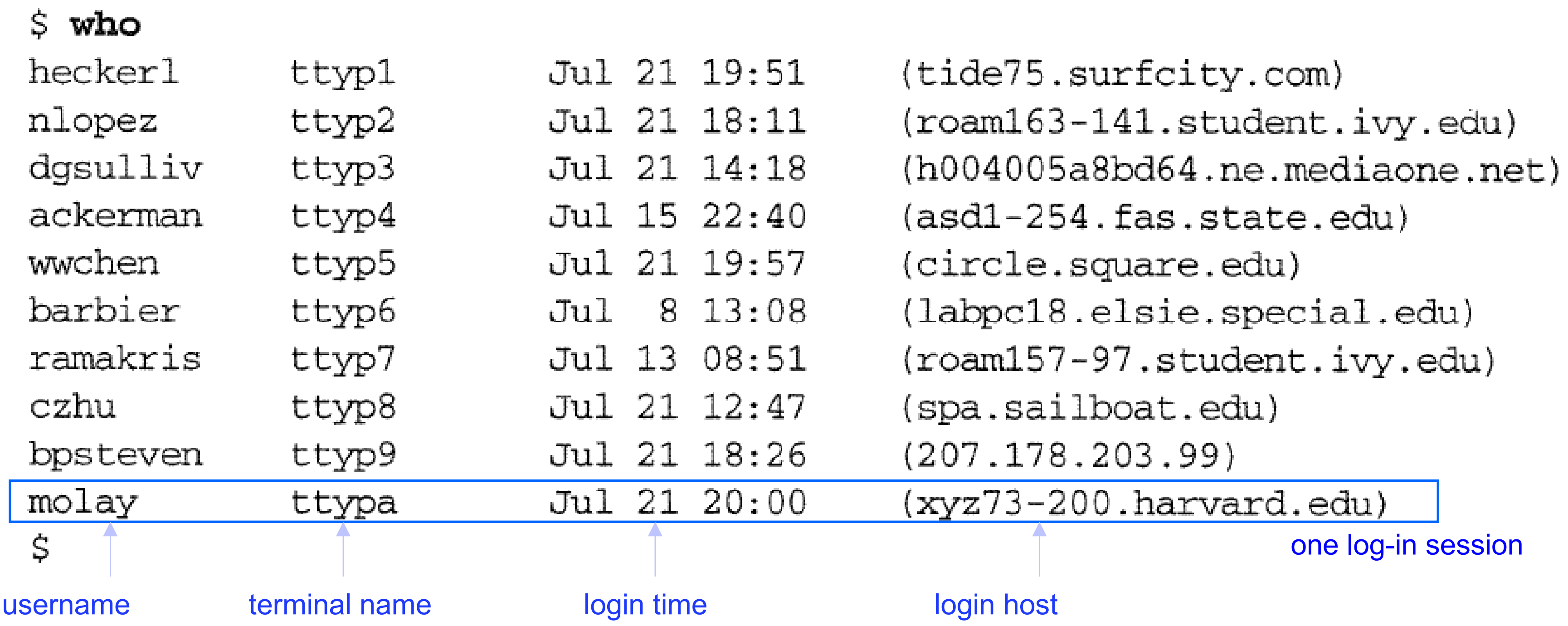
- Almost all Unix commands are simply programs written by a variety of people, usually in C.

- When you type **ls**, you are asking your **shell** to run the program named **ls**.

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

Q1: What Does who Do?

- The who command displays information about users and processes on the local system



```
$ 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

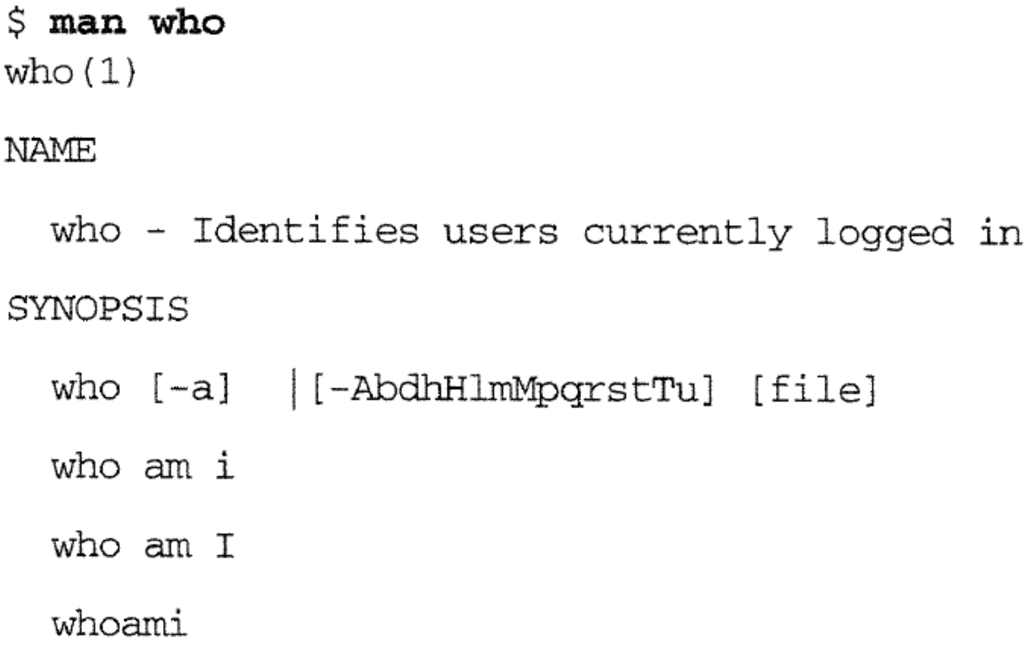
terminal name

login time

login host

Reading the Manual

- Every Unix system comes with manual for all commands
- The manual is on the disk and the command to read a page from the manual is **man**



```
$ man who
who(1)

NAME

  who - Identifies users currently logged in

SYNOPSIS

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

  who am i

  who am I

  whoami
```

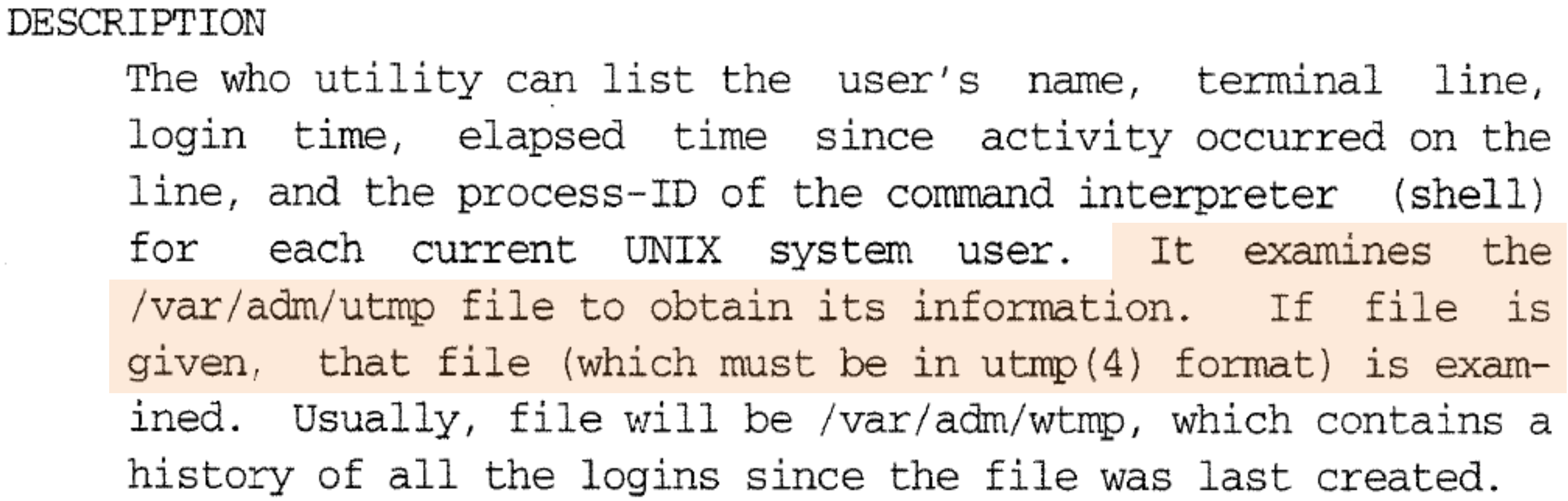
...

Q2: How Does who Do It?

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

Read the Manual

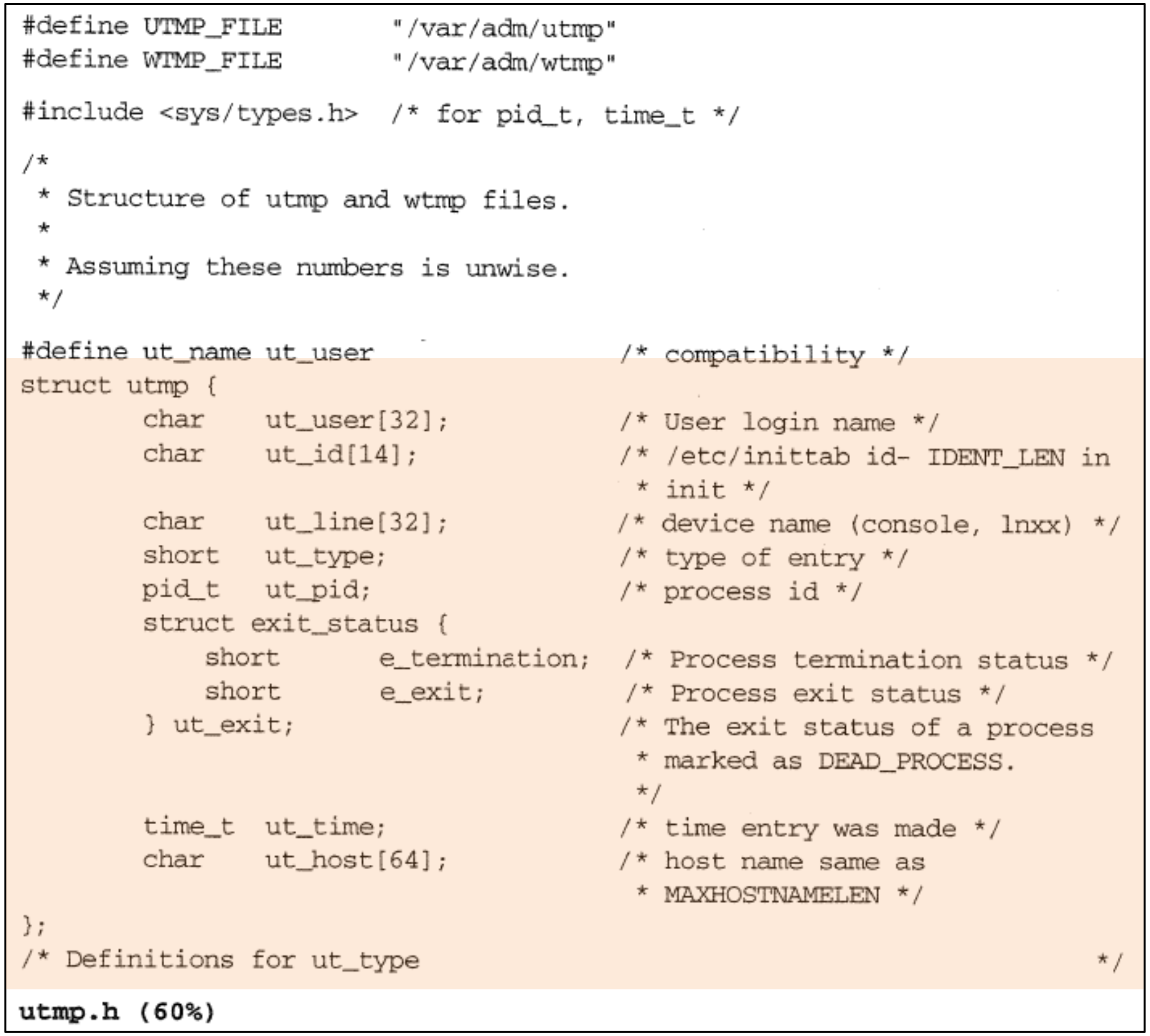
- $ man who



```
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 exam-
     ined.  Usually, file will be /var/adm/wtmp, which contains a
     history of all the logins since the file was last created.
```

Read the .h files

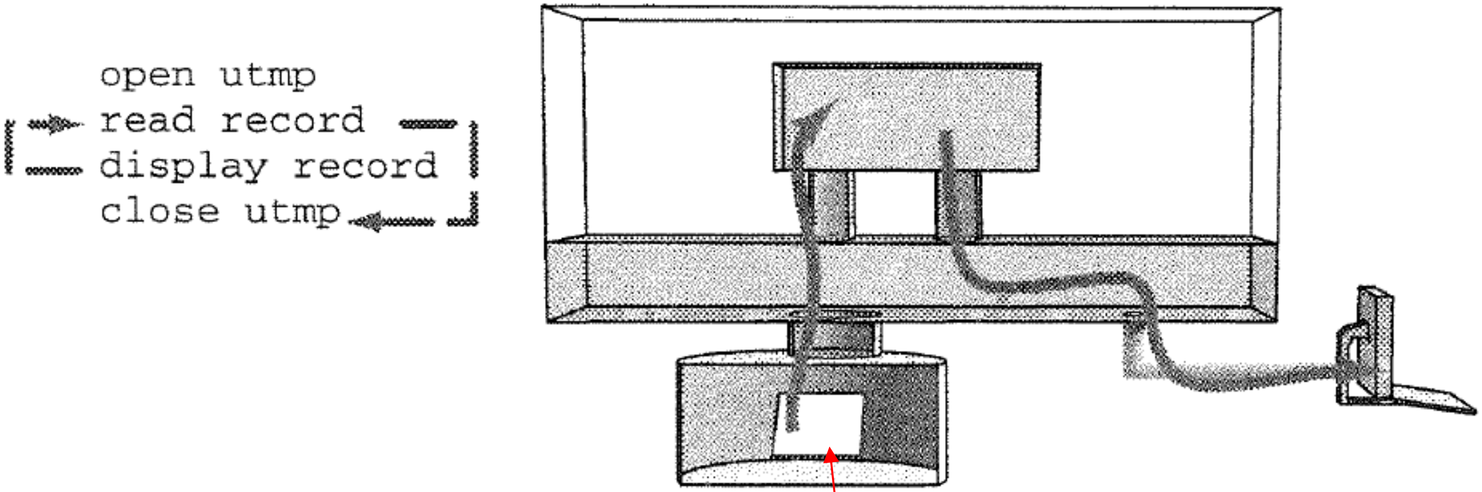
- $ more /usr/include/utmp.h



```
#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%)
```

We now know how who works

- who works by:



```
    open utmp
 -> read record     --
 |  display record   |
 -- close utmp    <--
```

FIGURE 2.2

Data flow in the who command.

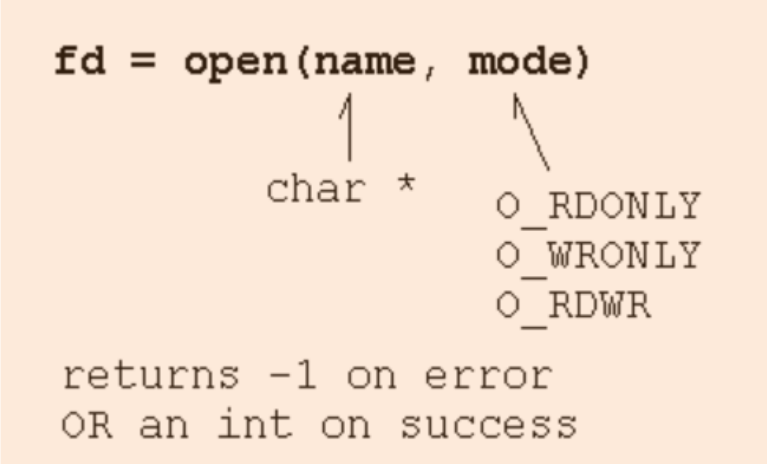
utmp

Can I Write who?

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

We use open, read, and close

- Opening a file: **open()**
  - Opening a file is a **kernel service**;
  - open() **system call** is a request from your program to the kernel



```
fd = open(name, mode)

            char *     O_RDONLY
                       O_WRONLY
                       O_RDWR

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

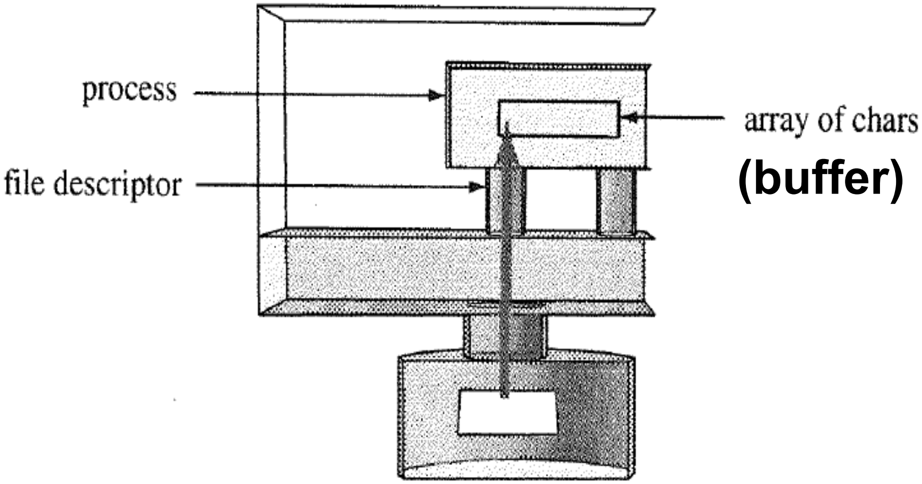
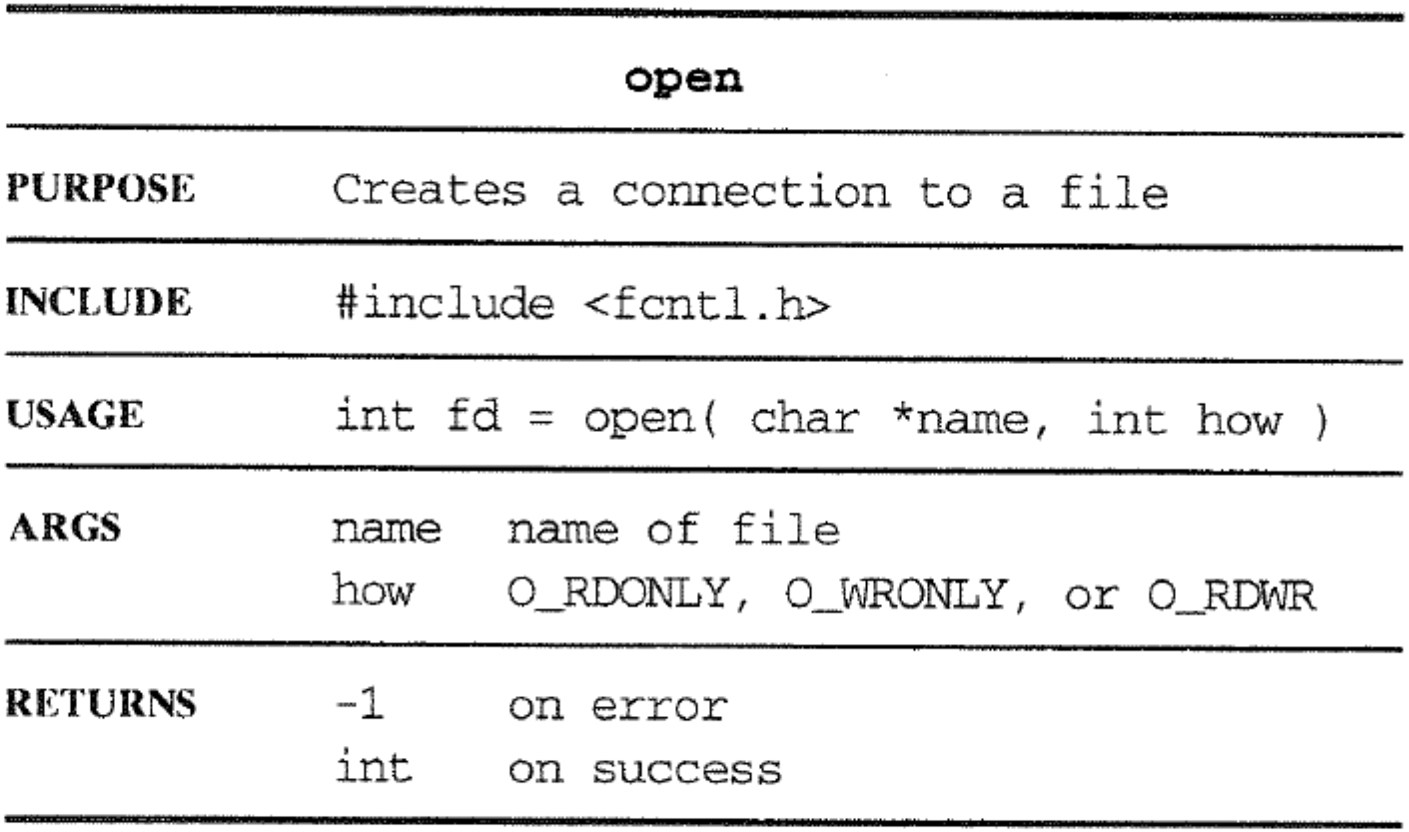


FIGURE 2.3

A file descriptor is a connection to a file.

More about open()



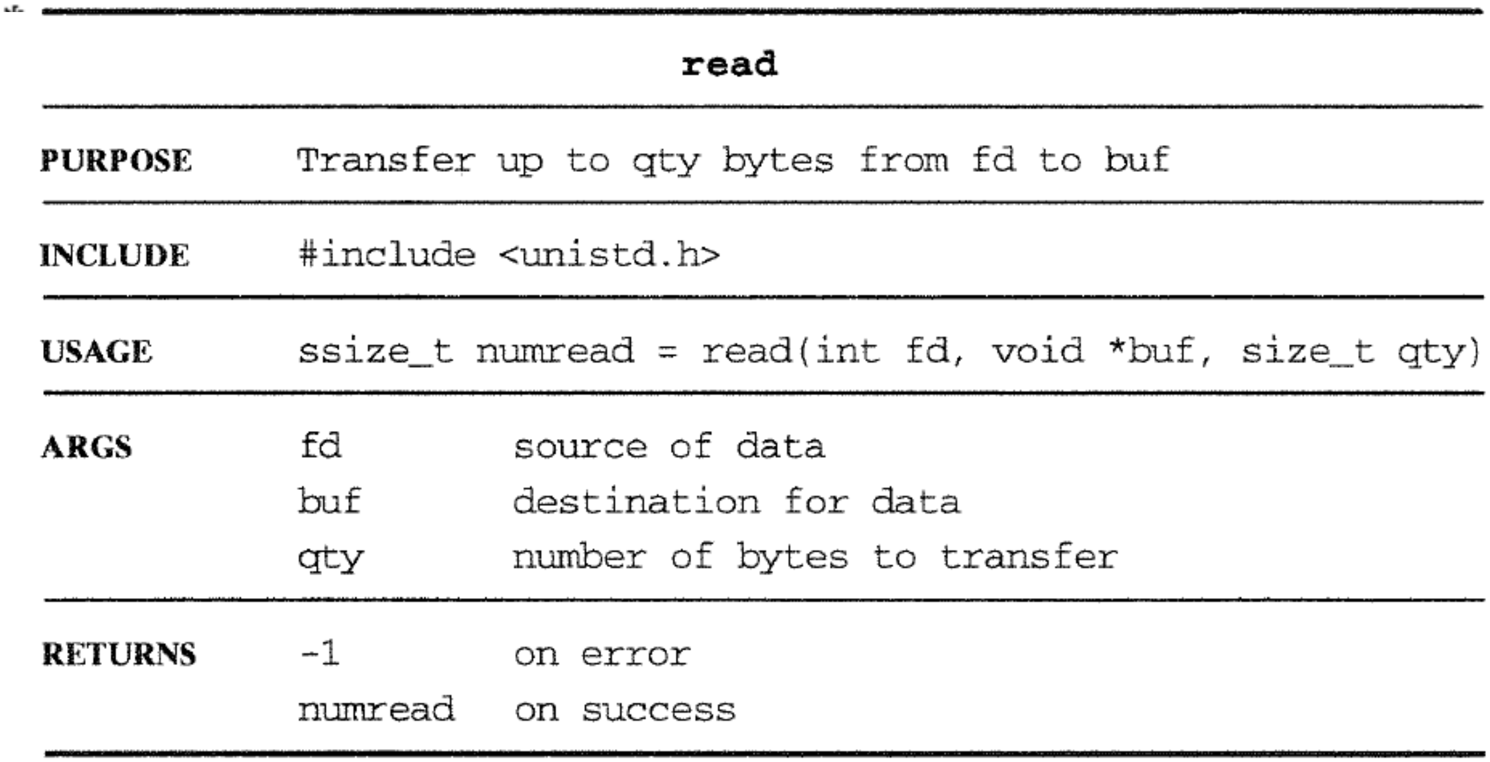
open		
PURPOSE	Creates a connection to a file	
INCLUDE	#include <fcntl.h>	
USAGE	int fd = open(char *name, int how)	
ARGS	name	name of file
	how	O_RDONLY, O_WRONLY, or O_RDWR
RETURNS	-1	on error
	int	on success

Can I Write who?

We use open, read, and close

- Reading Data from a File: **read()**

```
fd = open(name, mode)

n = read(fd, array, numchars)
```



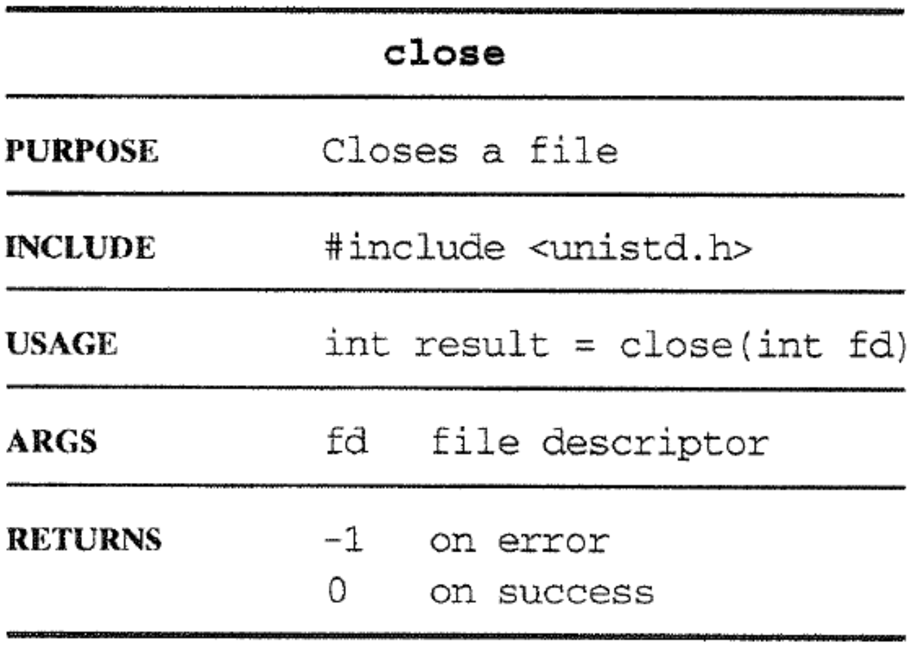
	read	
PURPOSE	Transfer up to qty bytes from fd to buf	
INCLUDE	#include <unistd.h>	
USAGE	ssize_t numread = read(int fd, void *buf, size_t qty)	
ARGS	fd	source of data
	buf	destination for data
	qty	number of bytes to transfer
RETURNS	-1	on error
	numread	on success

We use open, read, and close

- Closing a File: close()

```
fd = open(name, mode)

n = read(fd, array, numchars)
```

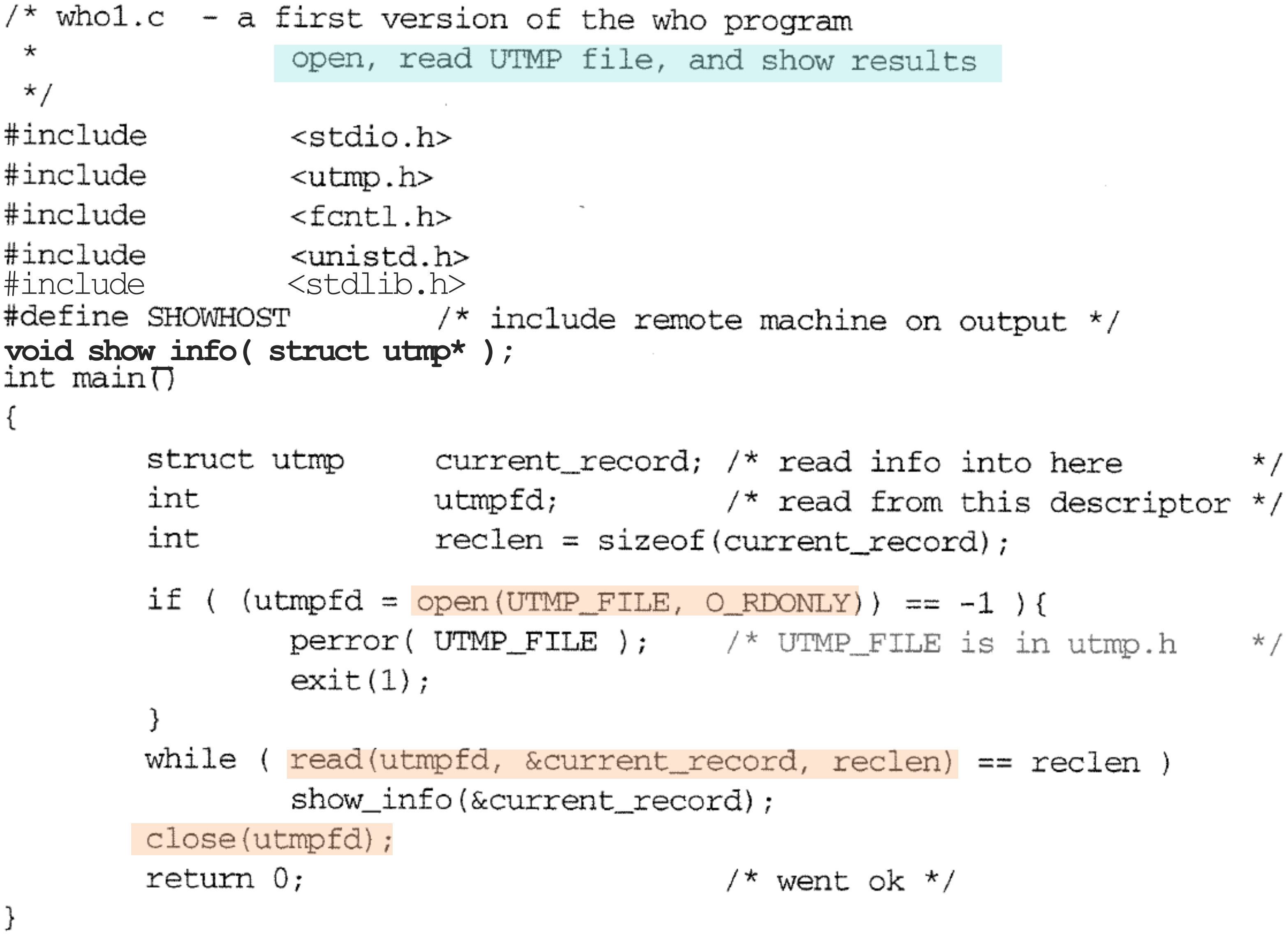
```
close(fd)
```



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

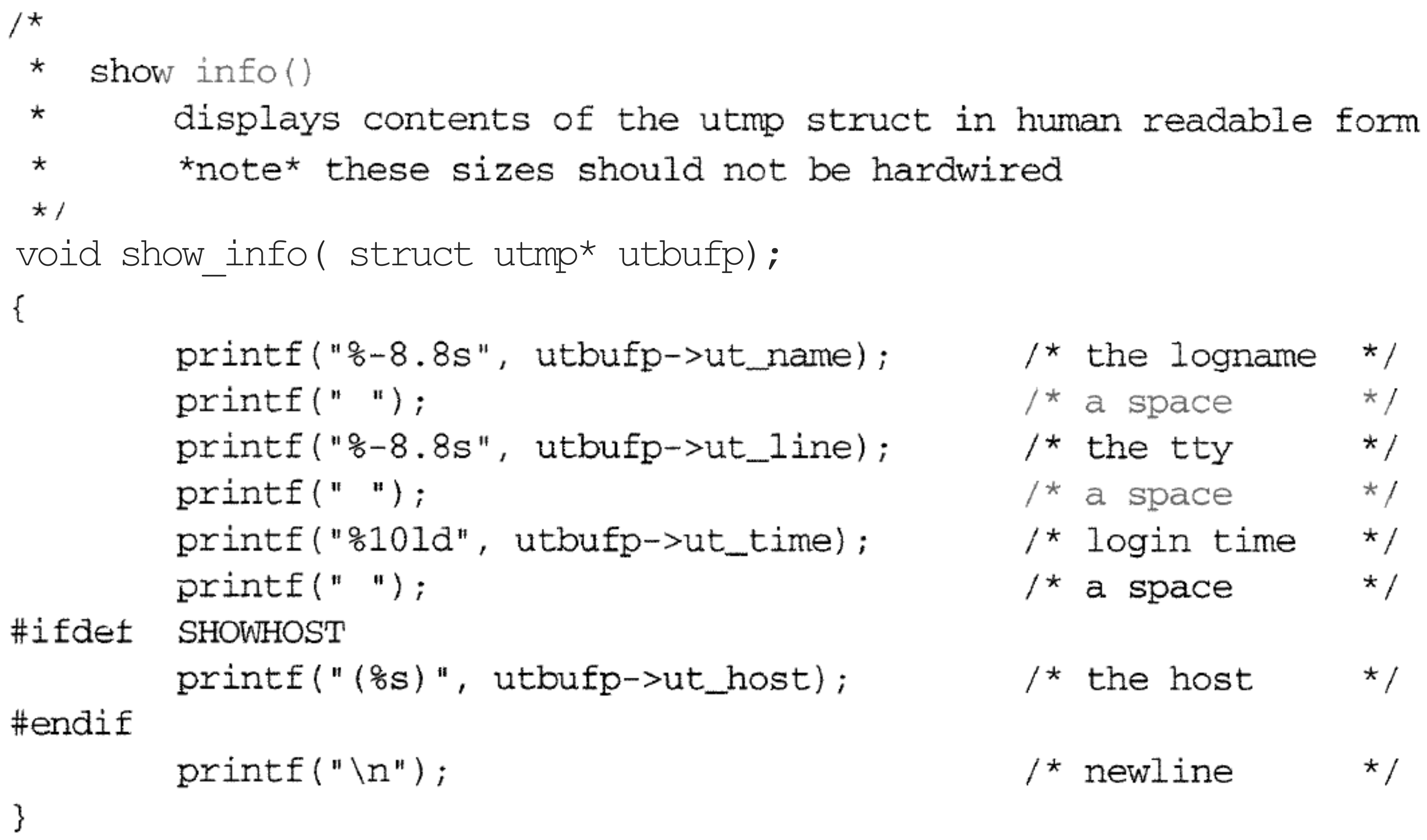
Why we need to close a file?

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>
#include        <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 */
}
```

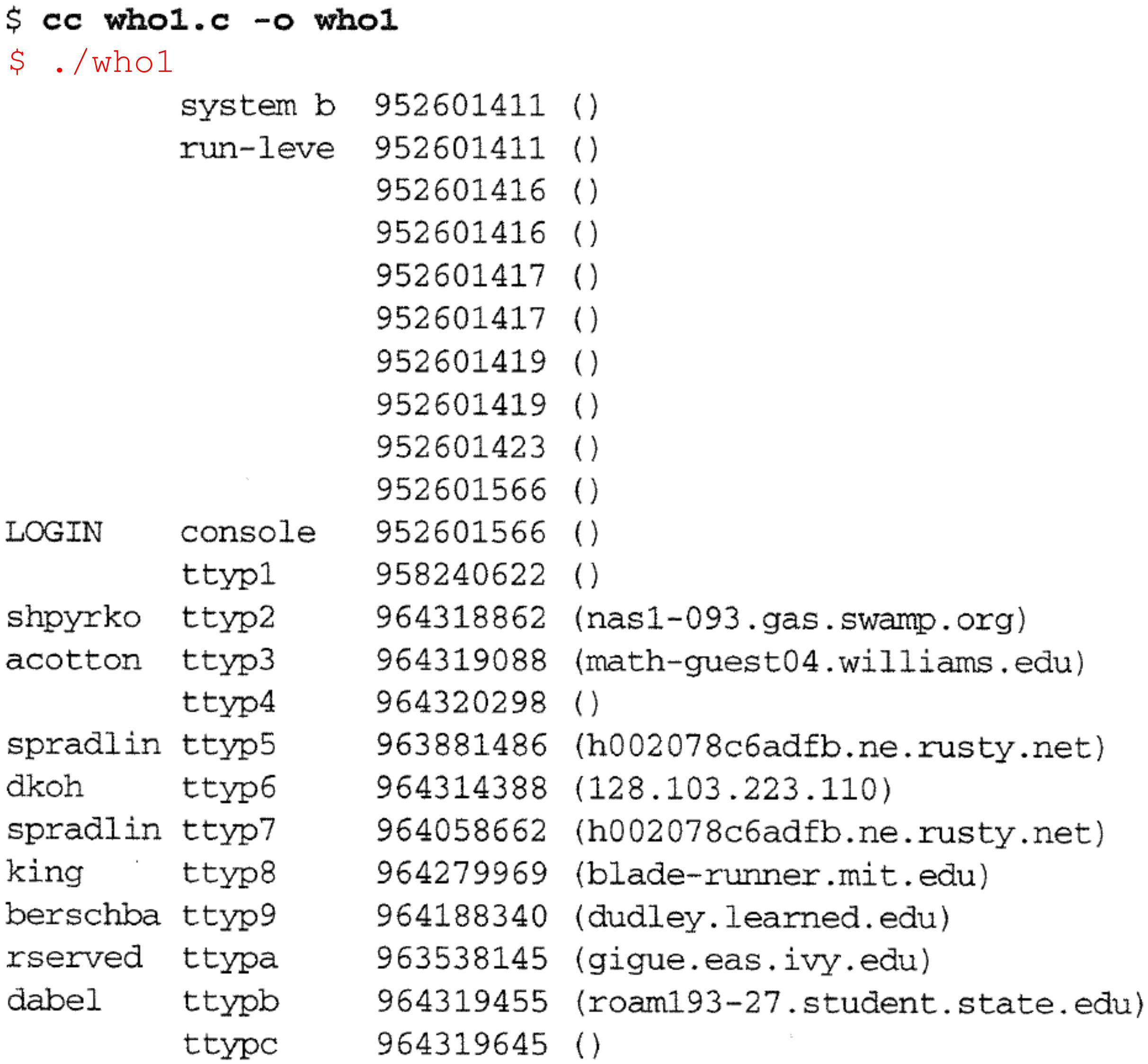
Displaying Log-In Records



```
/*
 *  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      */
}
```

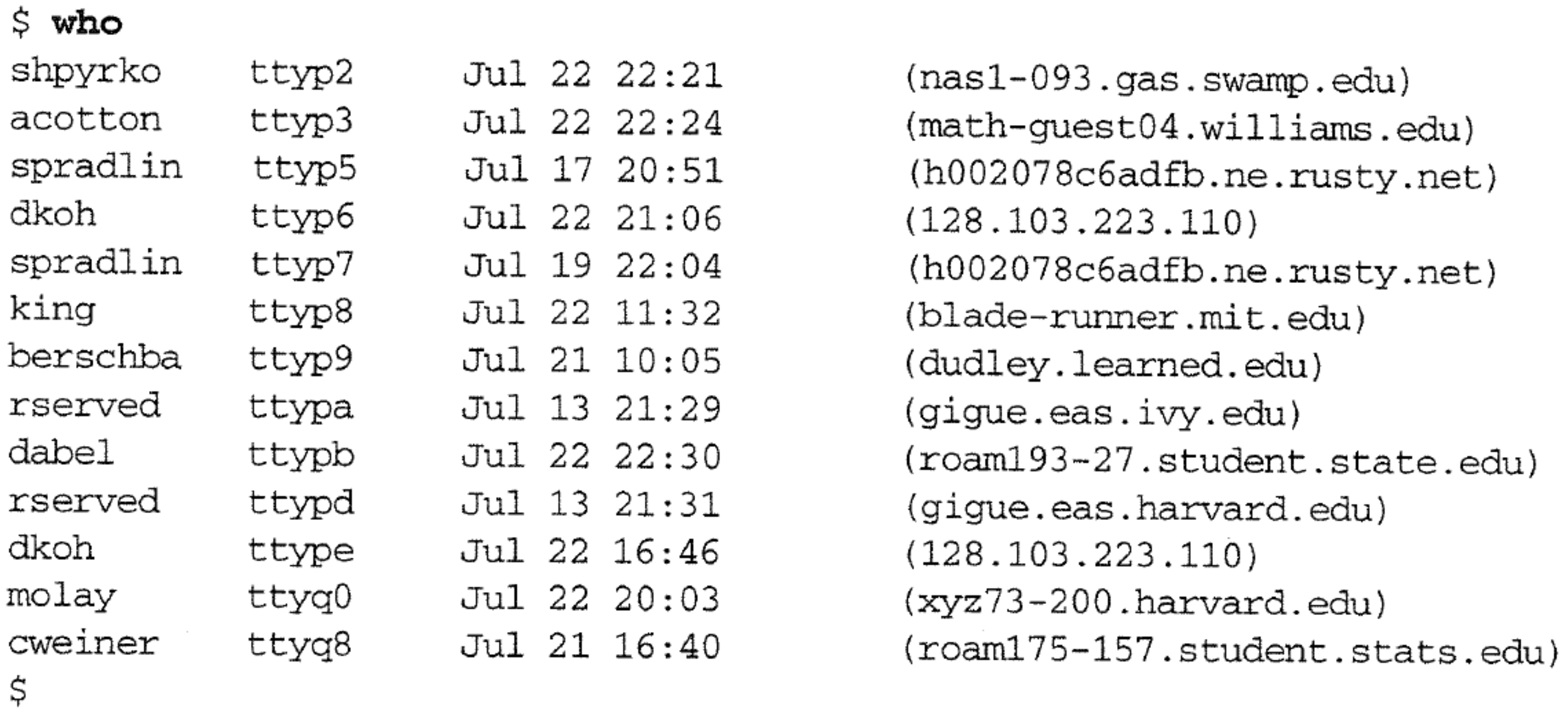
Can I Write who?

Compile and run it:



```
$ cc who1.c -o 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 ()
```

Let’s compare our program with the system version

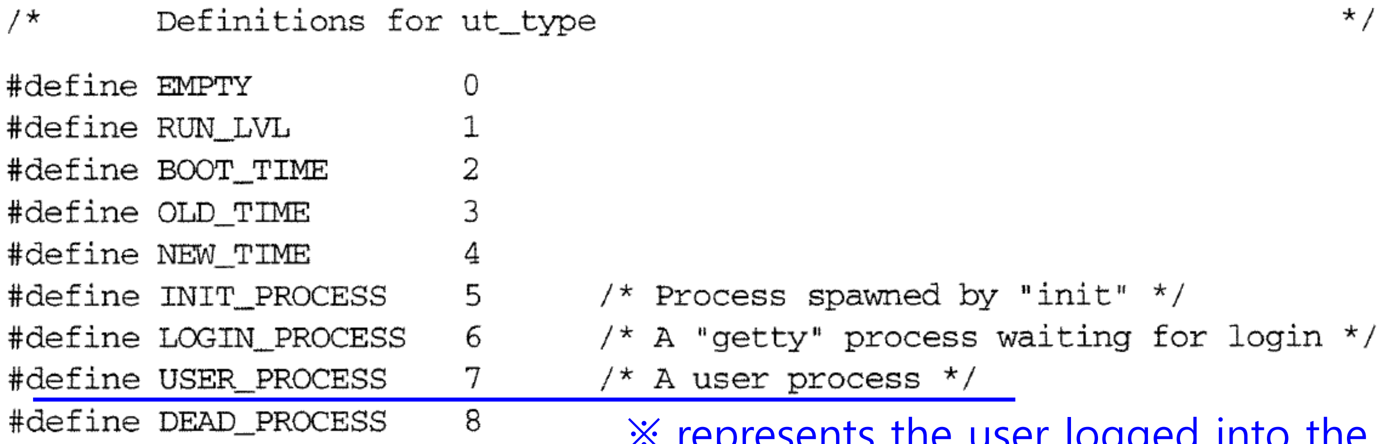


```
$ who
shpyrko     ttyp2     Jul 22 22:21         (nas1-093.gas.swamp.edu)
acotton     ttyp3     Jul 22 22:24         (math-guest04.williams.edu)
spradlin    ttyp5     Jul 17 20:51         (h002078c6adfb.ne.rusty.net)
dkoh        ttyp6     Jul 22 21:06         (128.103.223.110)
spradlin    ttyp7     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

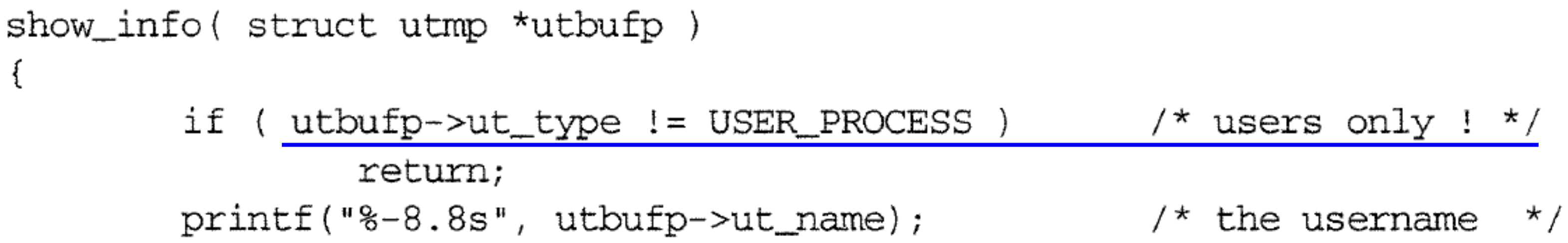
- Suppressing blank records
  - /usr/include/utmp.h



```
/*      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



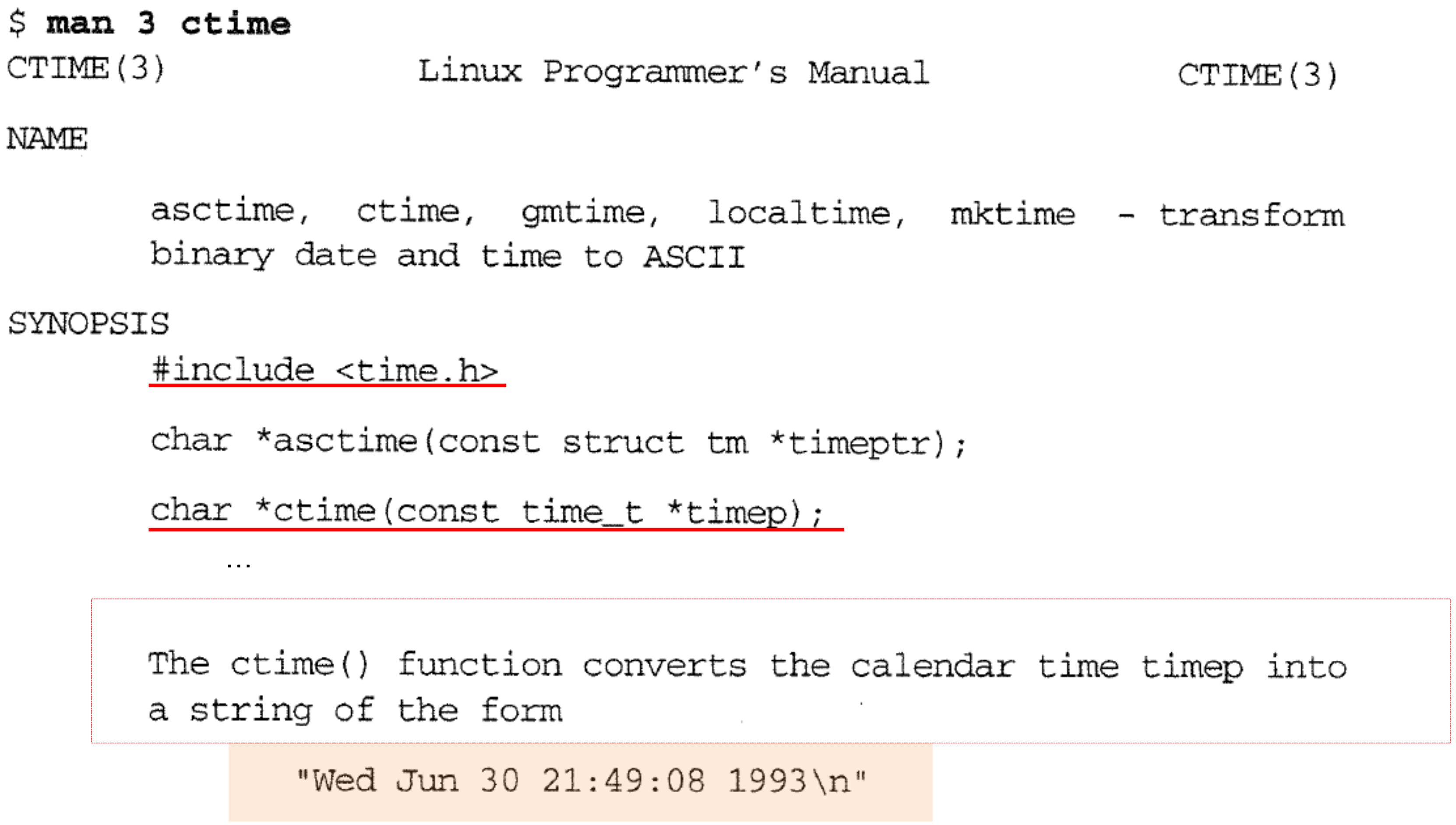
```
show_info( struct utmp *utbufp )
{
        if ( utbufp->ut_type != USER_PROCESS )      /* users only ! */
                return;
        printf("%-8.8s", utbufp->ut_name);          /* the username  */
```

Writing who2.c (cont.)

- Displaying Log-in Time in Human-Readable Form
  - How unix stores times: time_t
    - Unix stores times as the number of seconds since midnight, Jan 1, 1970, G.M.T. The time_t data type is an integer that stores a number of seconds.
  - Making a time_t readable: **ctime**

Writing who2.c (cont.)

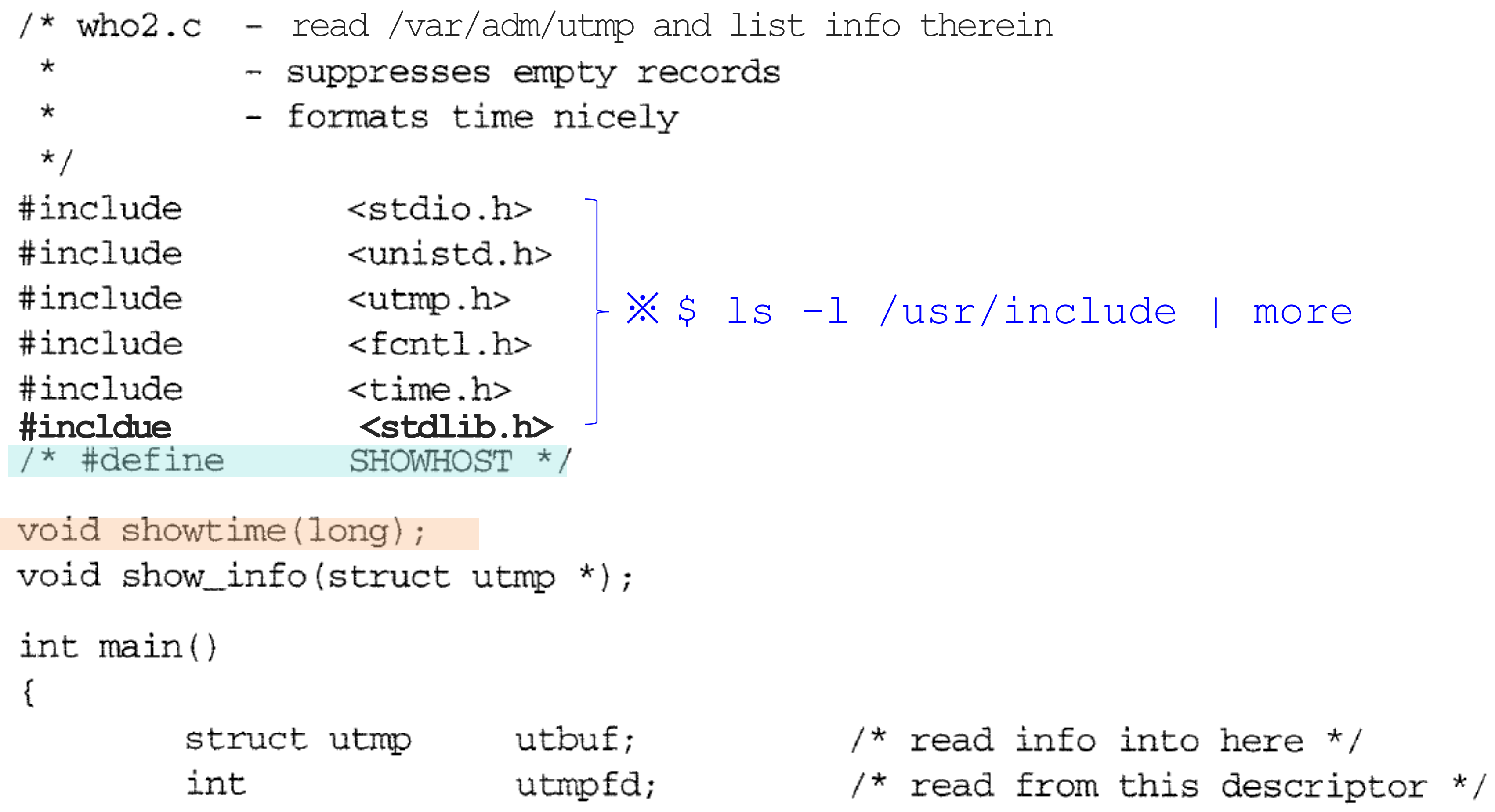
- Displaying Log-in Time in Human-Readable Form



```
$ 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"
```

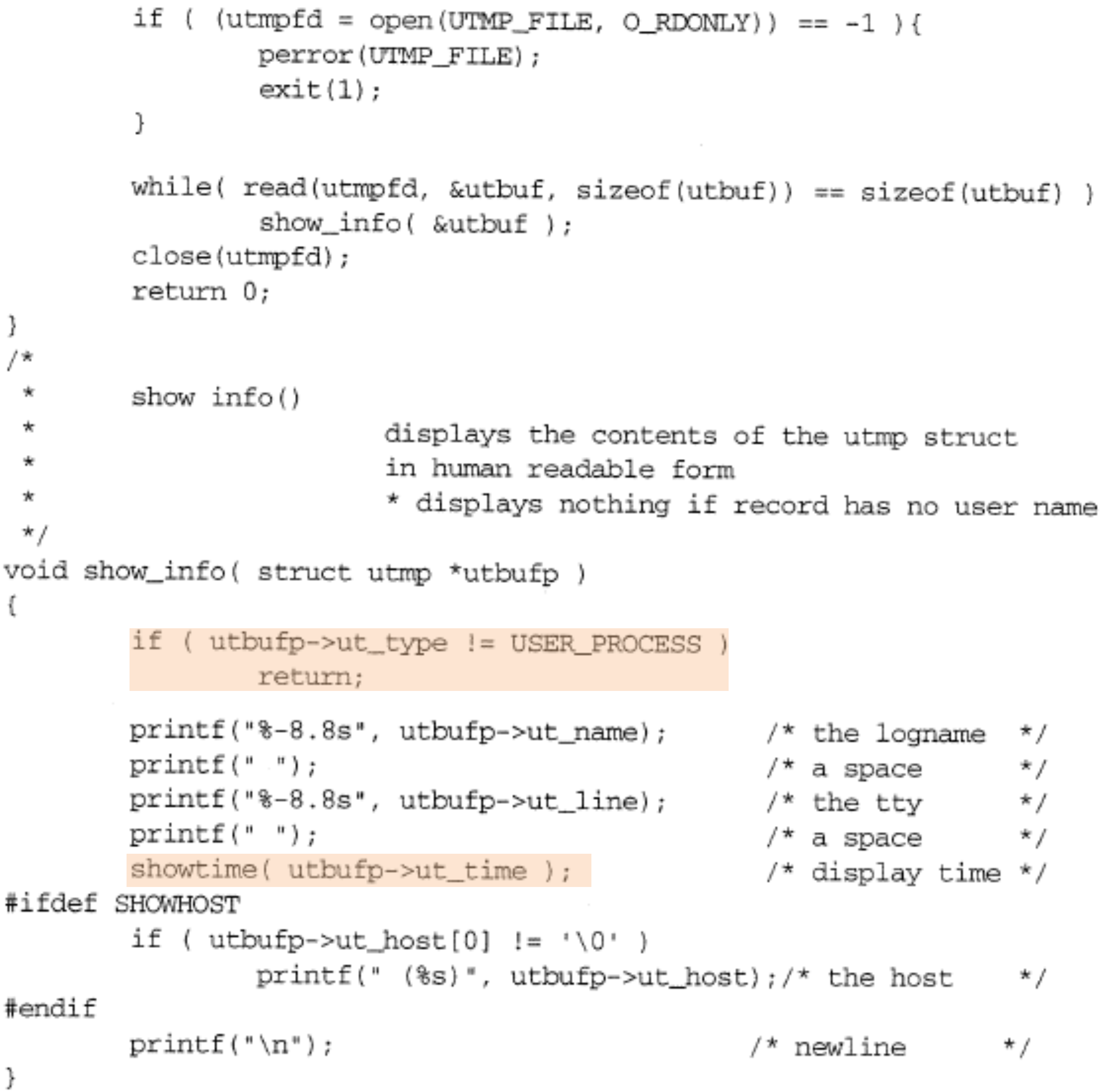
Writing who2.c (cont.)

- Putting it All together



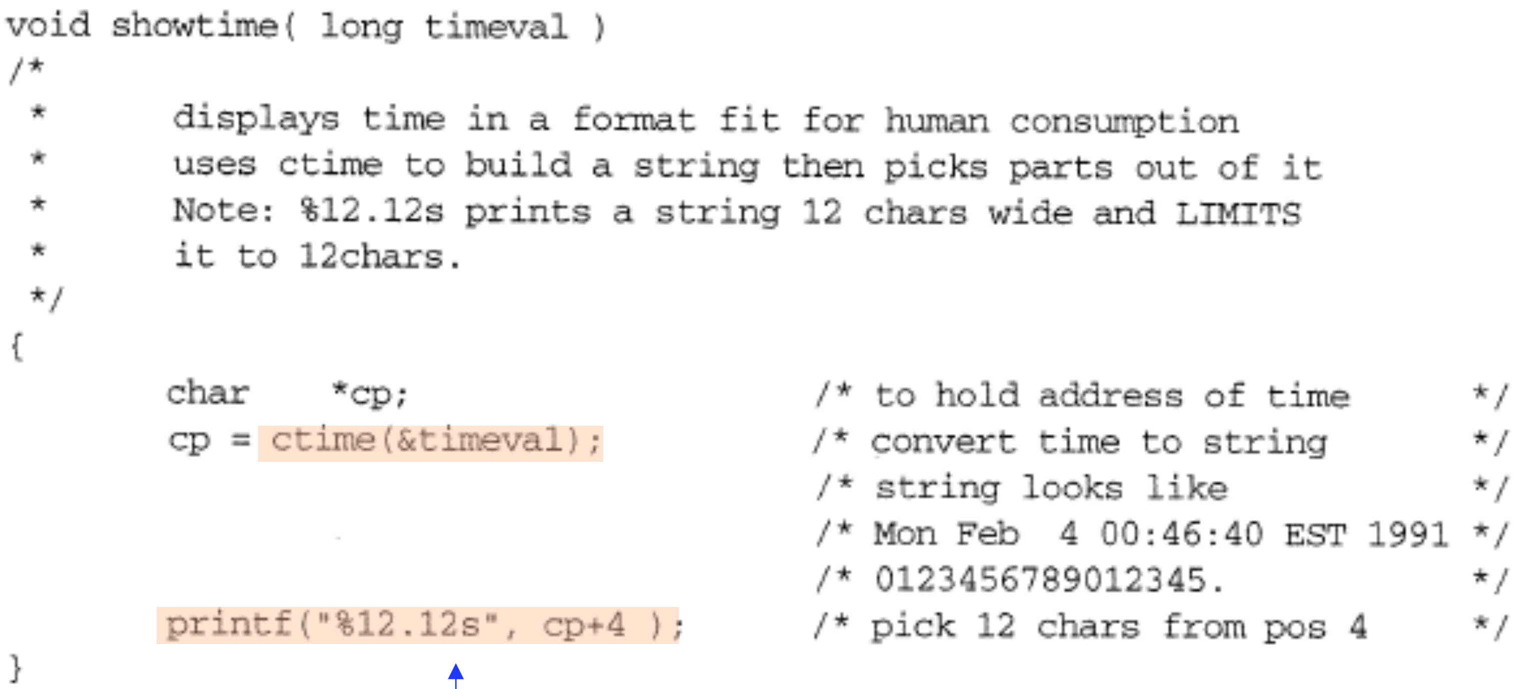
```
/* who2.c   - read /var/adm/utmp and list info therein
 *          - suppresses empty records
 *          - formats time nicely
 */
#include        <stdio.h>
#include        <unistd.h>
#include        <utmp.h>        ※ $ ls -l /usr/include | more
#include        <fcntl.h>
#include        <time.h>
#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 */
```

Can I Write who?



```
        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      */
}
```

Can I Write who?

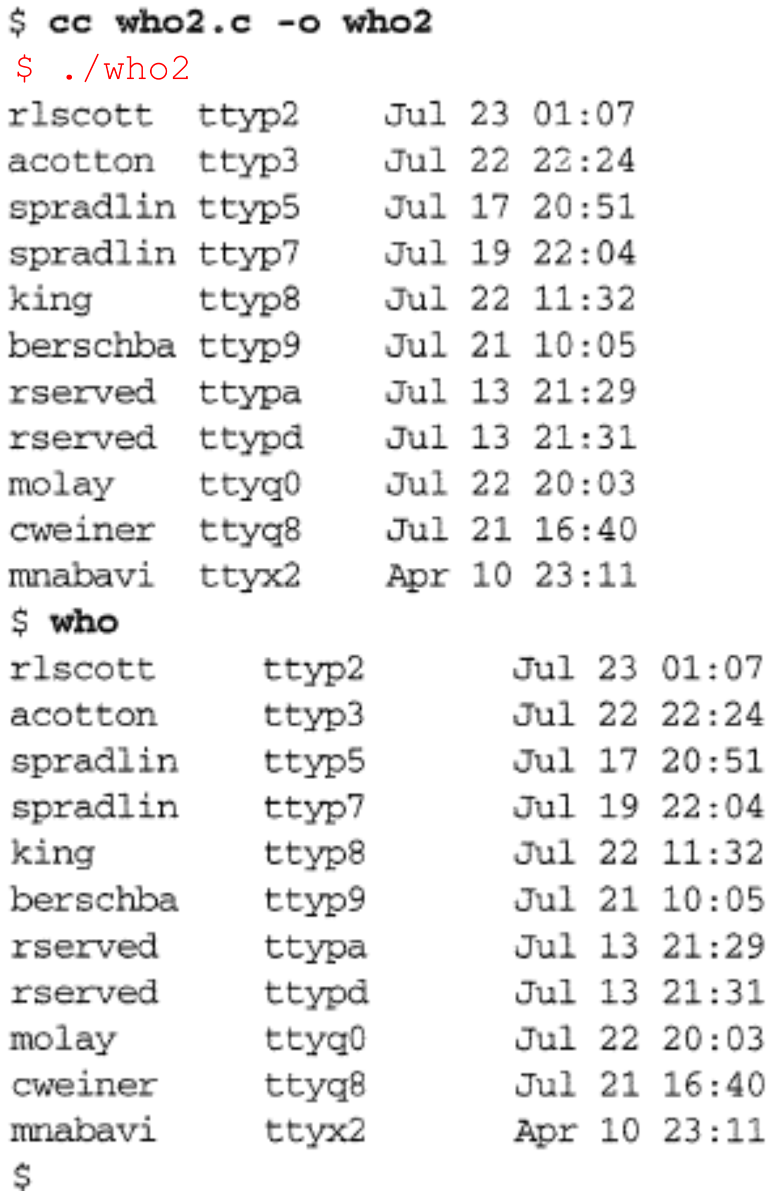


```
void showtime( long timeval )
/*
 *      displays time in a format fit for human consumption
 *      uses ctime to build a string then picks parts out of it
 *      Note: %12.12s prints a string 12 chars wide and LIMITS
 *      it to 12chars.
 */
{
        char    *cp;                    /* to hold 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     */
}
```

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

partial string!

Testing who2.c



```
$ cc who2.c -o 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
$
```

Agenda

5 min break!

- 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
- 2.9 Reading and Writing a File
- 2.10 What to Do with System-Call Errors

Q1: What does cp do?

- cp makes a copy of a file

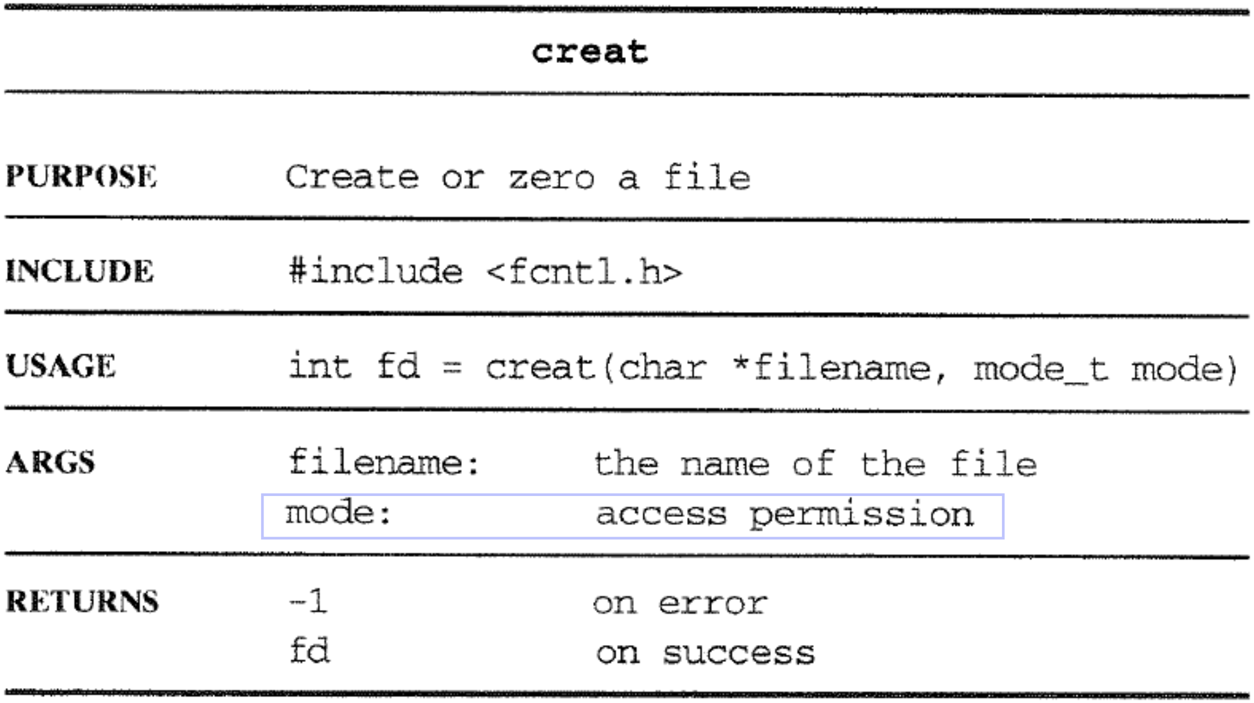
```
$ cp source-file target-file
```

  - 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, 644);



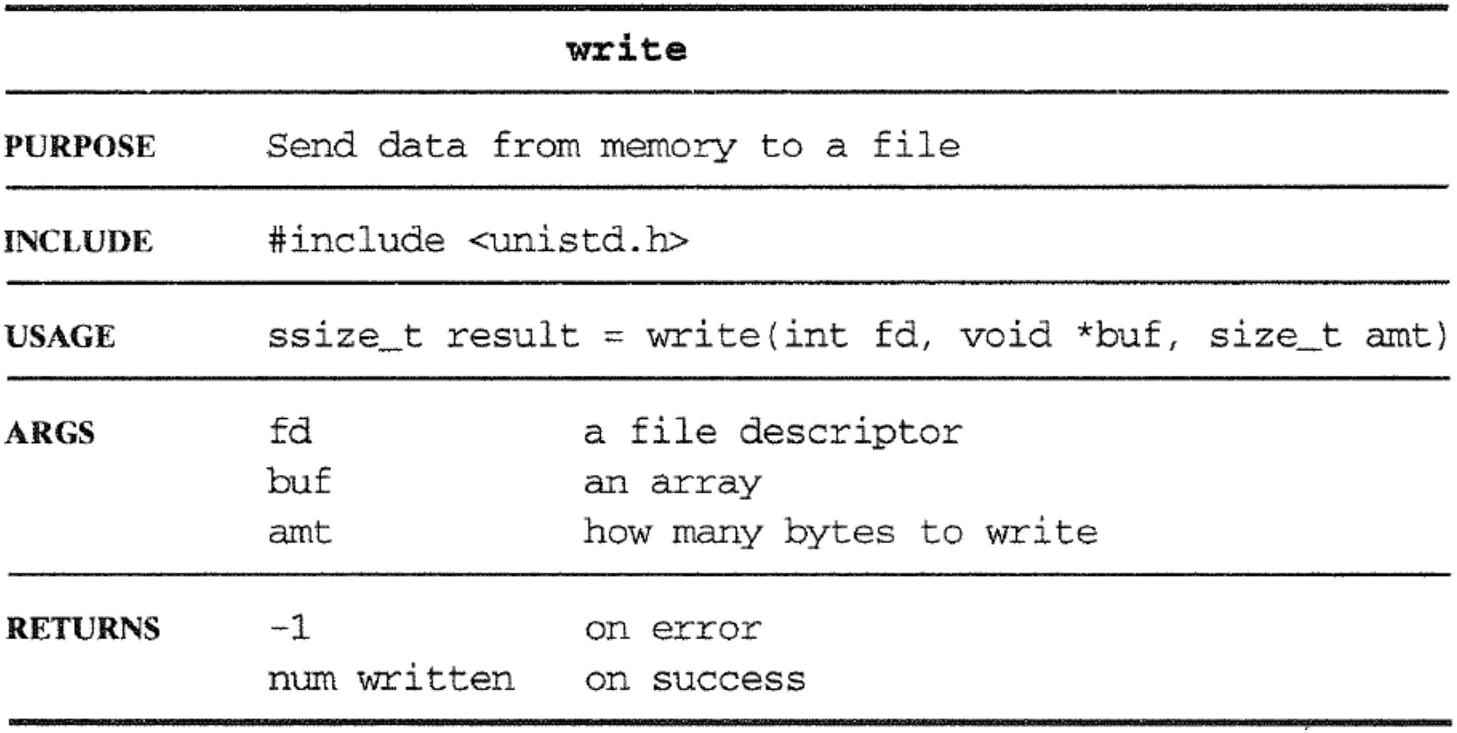
creat

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

Q2: How Does cp Create and Write?

- Writing to a File

n = write(fd, buffer, num);

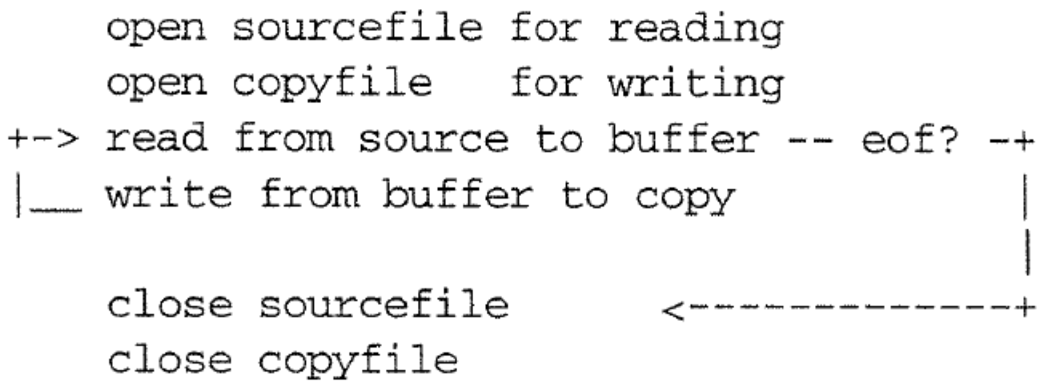


write

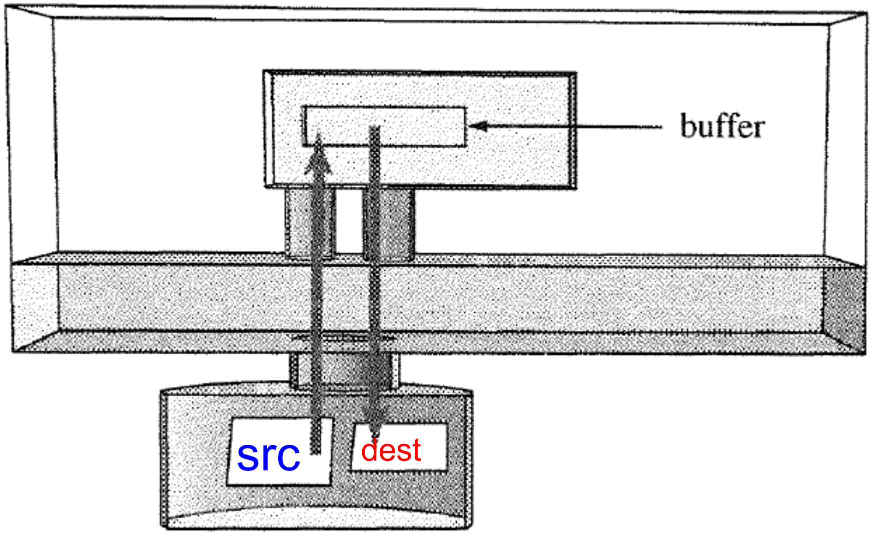
PURPOSE	Send data from memory to a file	
INCLUDE	#include <unistd.h>	
USAGE	ssize_t result = write(int fd, void *buf, size_t amt)	
ARGS	fd	a file descriptor
	buf	an array
	amt	how many bytes to write
RETURNS	-1	on error
	num written	on success

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
```

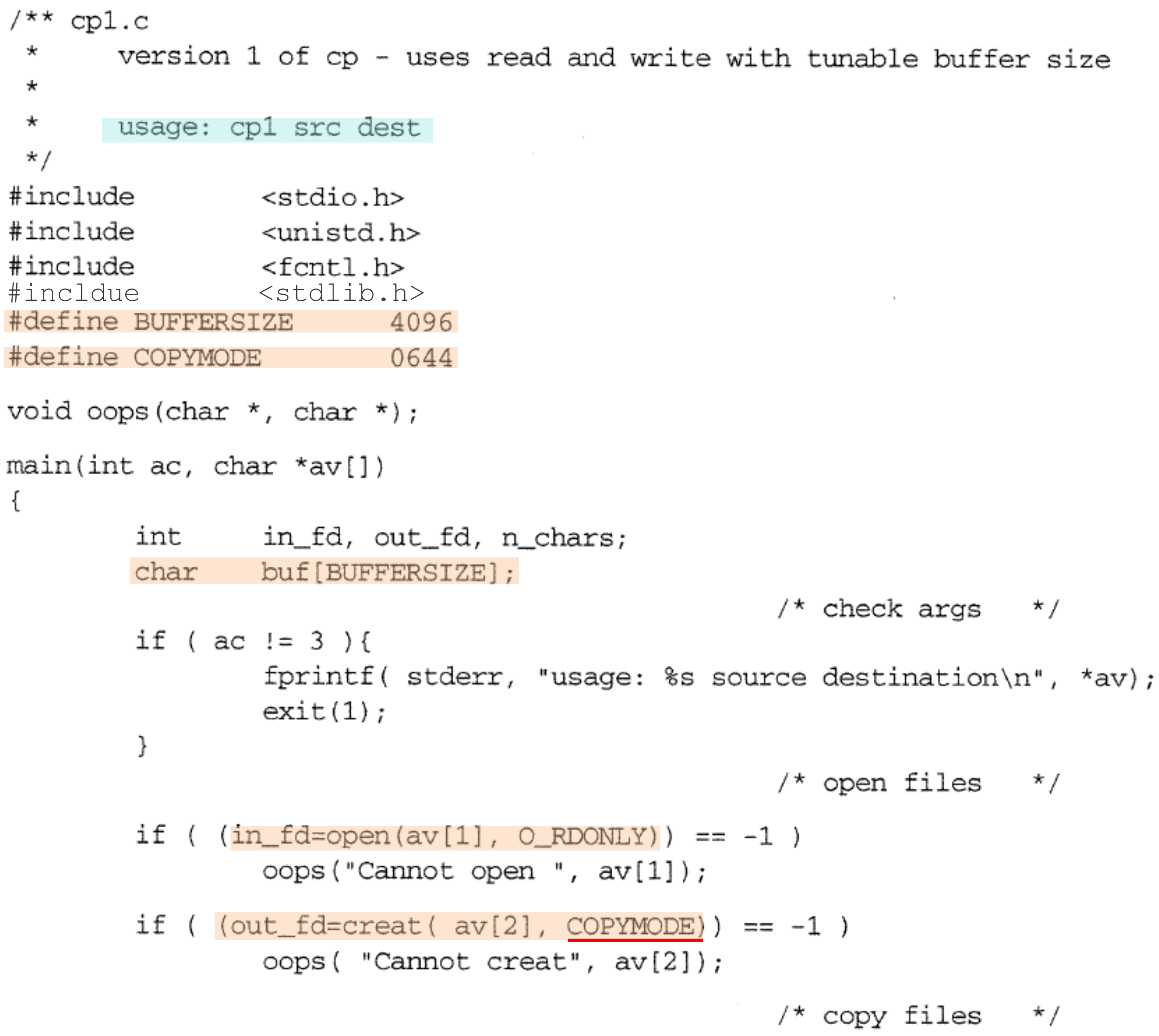


Why do we need to close a file?

FIGURE 2.4

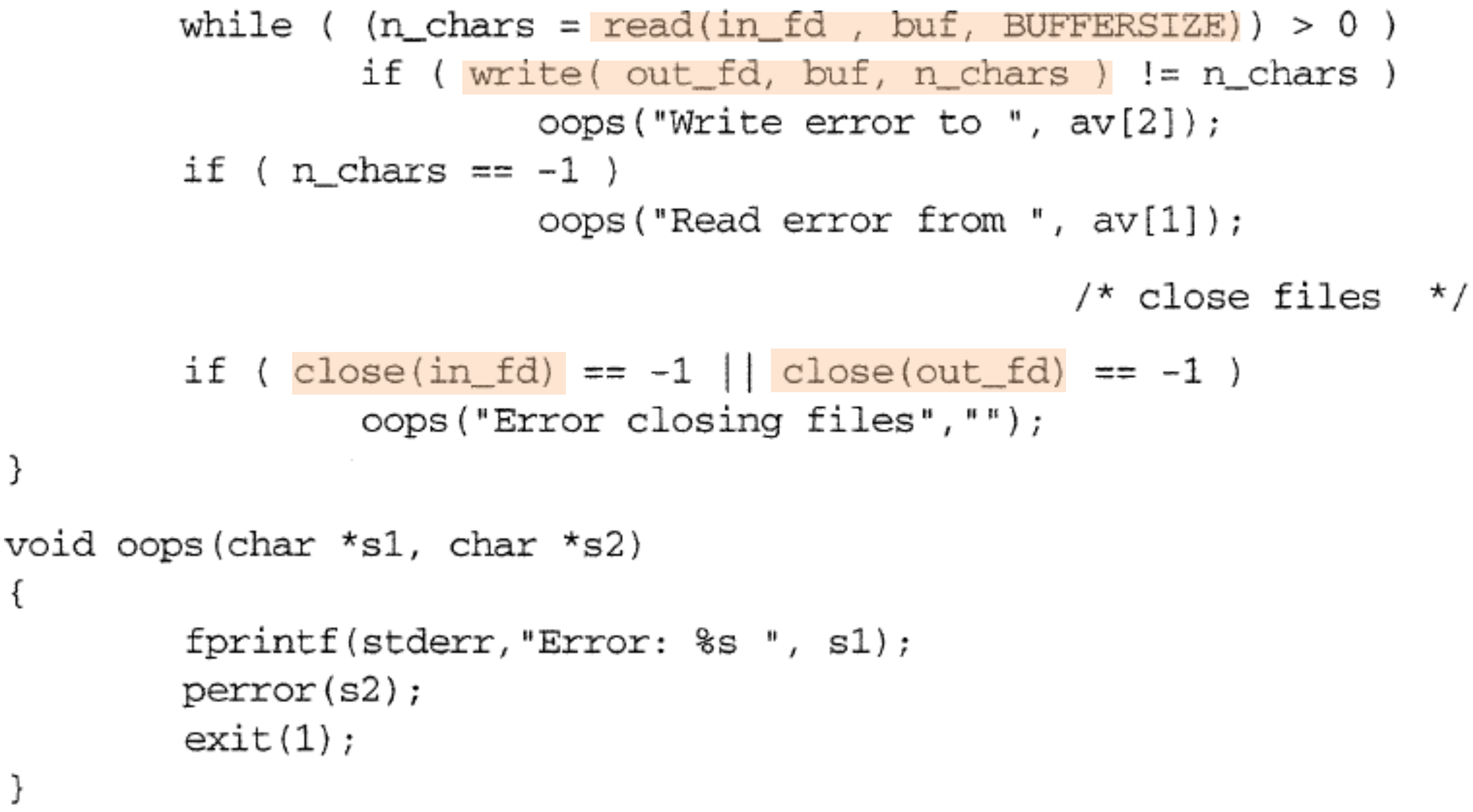
Copying a file by reading and writing.

Can I Write cp



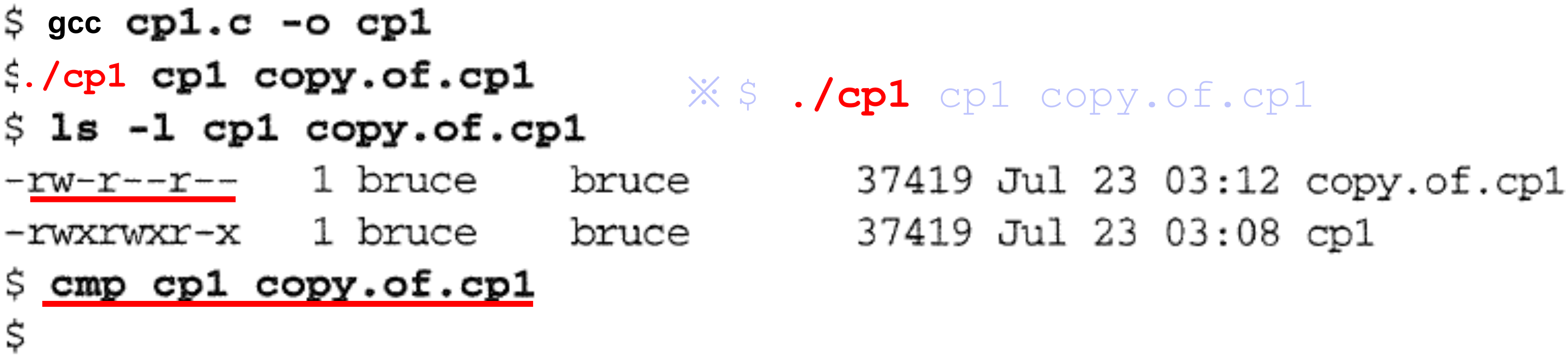
```
/** cp1.c
 *      version 1 of cp - uses read and write with tunable buffer size
 *
 *      usage: cp1 src dest
 */
#include        <stdio.h>
#include        <unistd.h>
#include        <fcntl.h>
#incldue        <stdlib.h>
#define BUFFERSIZE      4096
#define COPYMODE        0644

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   */
```

Can I Write **cp**



```
        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);
}
```

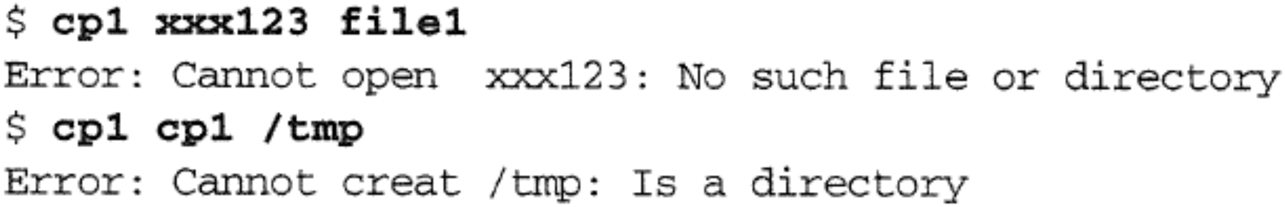
Compile and run it:



```
$ gcc cp1.c -o 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
$
```

※ $ ./cp1 cp1 copy.of.cp1

How well does our program respond to errors?



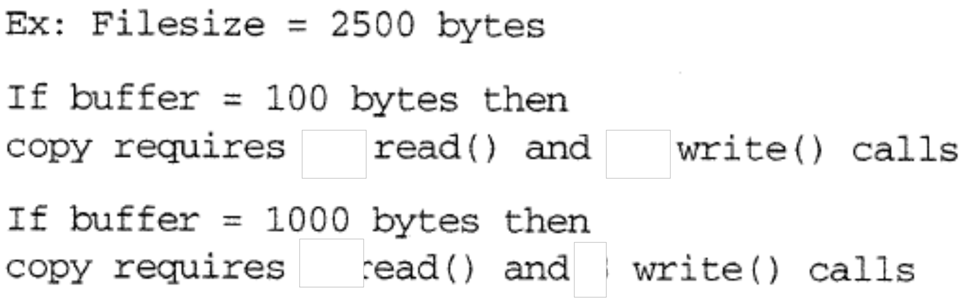
```
$ cp1 xxx123 file1
Error: Cannot open  xxx123: No such file or directory
$ cp1 cp1 /tmp
Error: Cannot creat /tmp: Is a directory
```

Agenda

- 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
- 2.9 Reading and Writing a File
- 2.10 What to Do with System-Call Errors

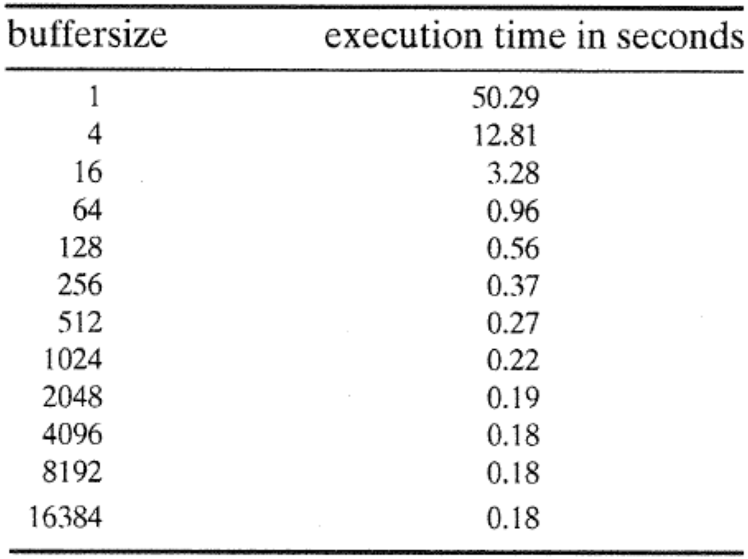
Does the Size of the Buffer Matter?

- Example:



```
Ex: Filesize = 2500 bytes

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

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

- System calls consume time:



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

`cp1` copying a 5MB file

Why System Calls are Time Consuming?

- It runs various kernel functions, and it also requires a shift from USER MODE to KERNEL MODE and back;
- Not only does transferring data take time but mode change takes time
- **Thus, try to minimize system calls**

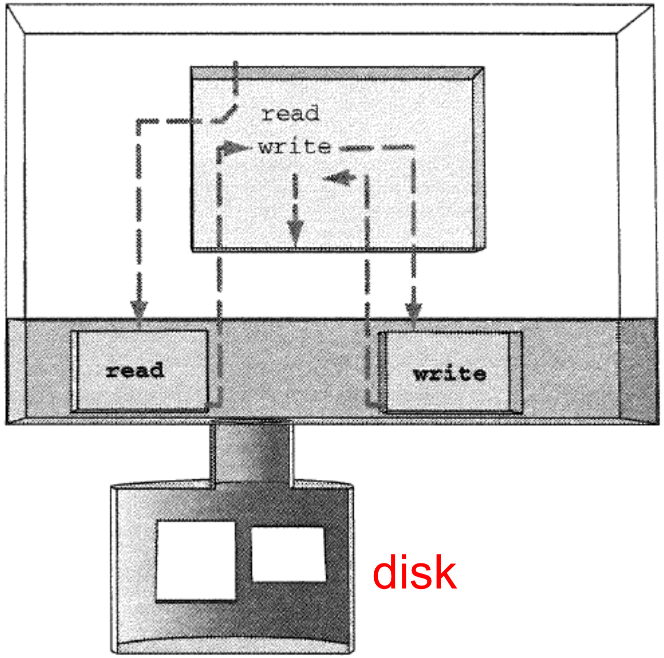


FIGURE 2.5

Control flow during system calls.

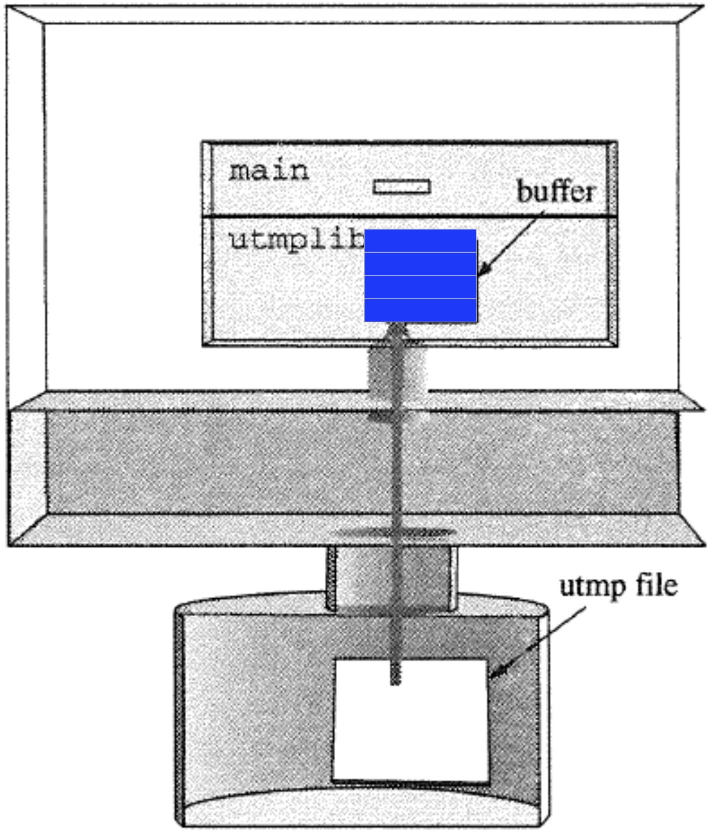
Does This Mean That who2.c is Inefficient?

- **Yes!**
  - Making one system call for each line of output makes as much sense as buying pizza by the slice or eggs one at a time
  - who2.c use one system call for each utmp record

- A better idea:
  - Read in a bunch of records at once
  - Then, process the ones in your local storage one by one

Adding Buffering to who2.c

- We make who2.c much more efficient by using buffering to reduce system calls



File buffering with utmplib

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

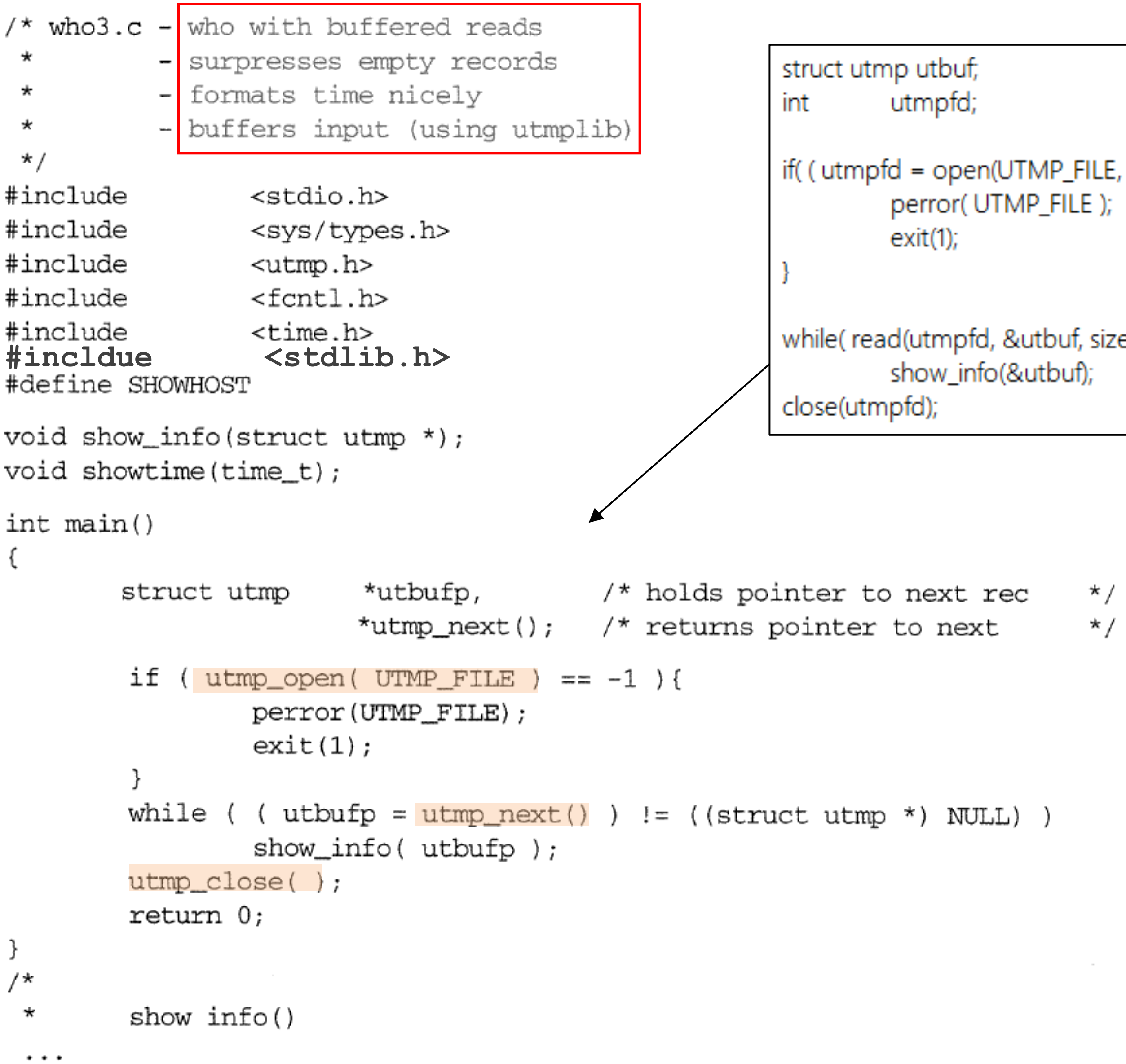
Functions in utmplib.c read structs 16 at a time from the disk into an array.

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

FIGURE 2.6

Buffering disk data in user space.

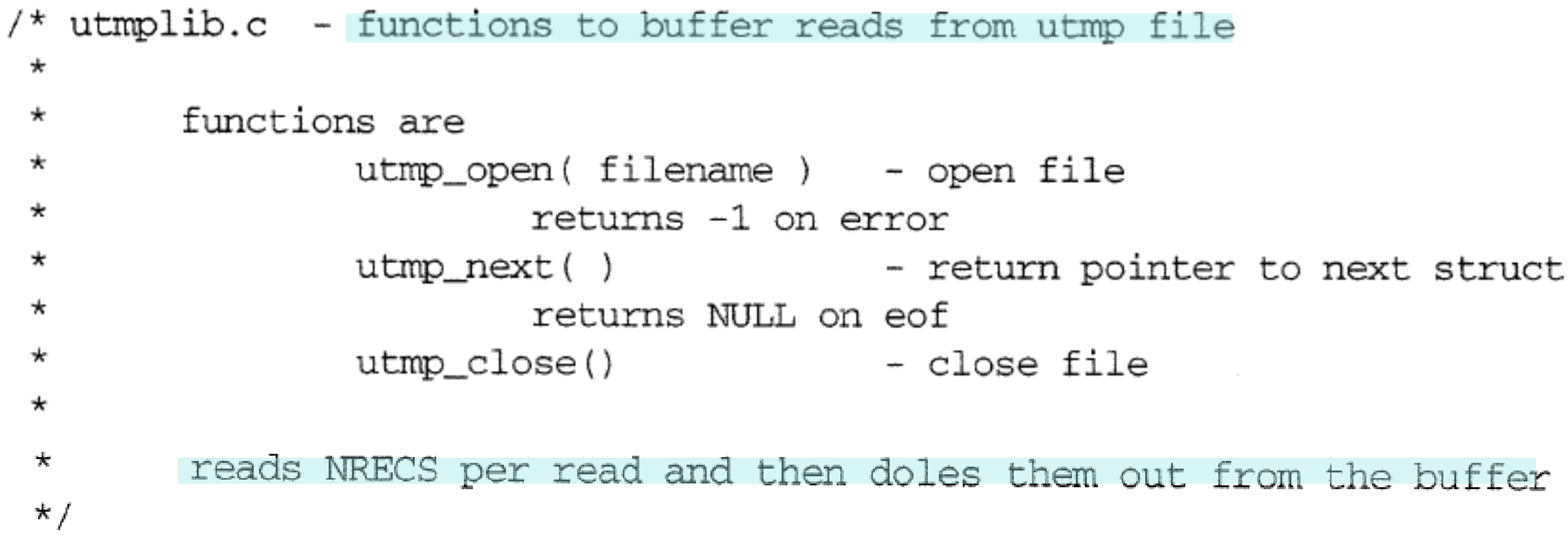
Revised Version: `who3.c`



```
/* 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);
```

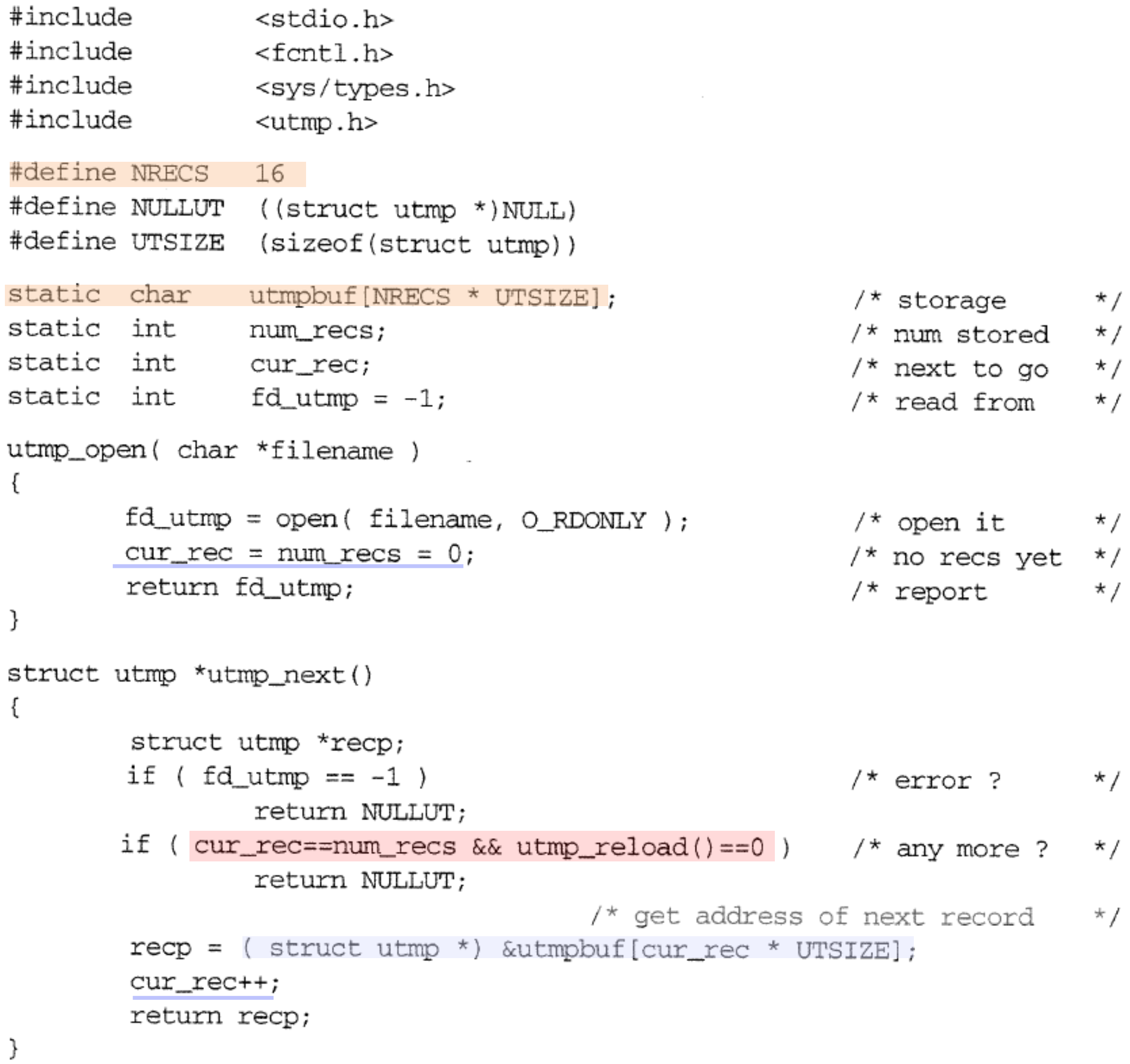
- The code for `utmplib.c`



```
/* 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
 */
```

More Efficient File I/O: Buffering

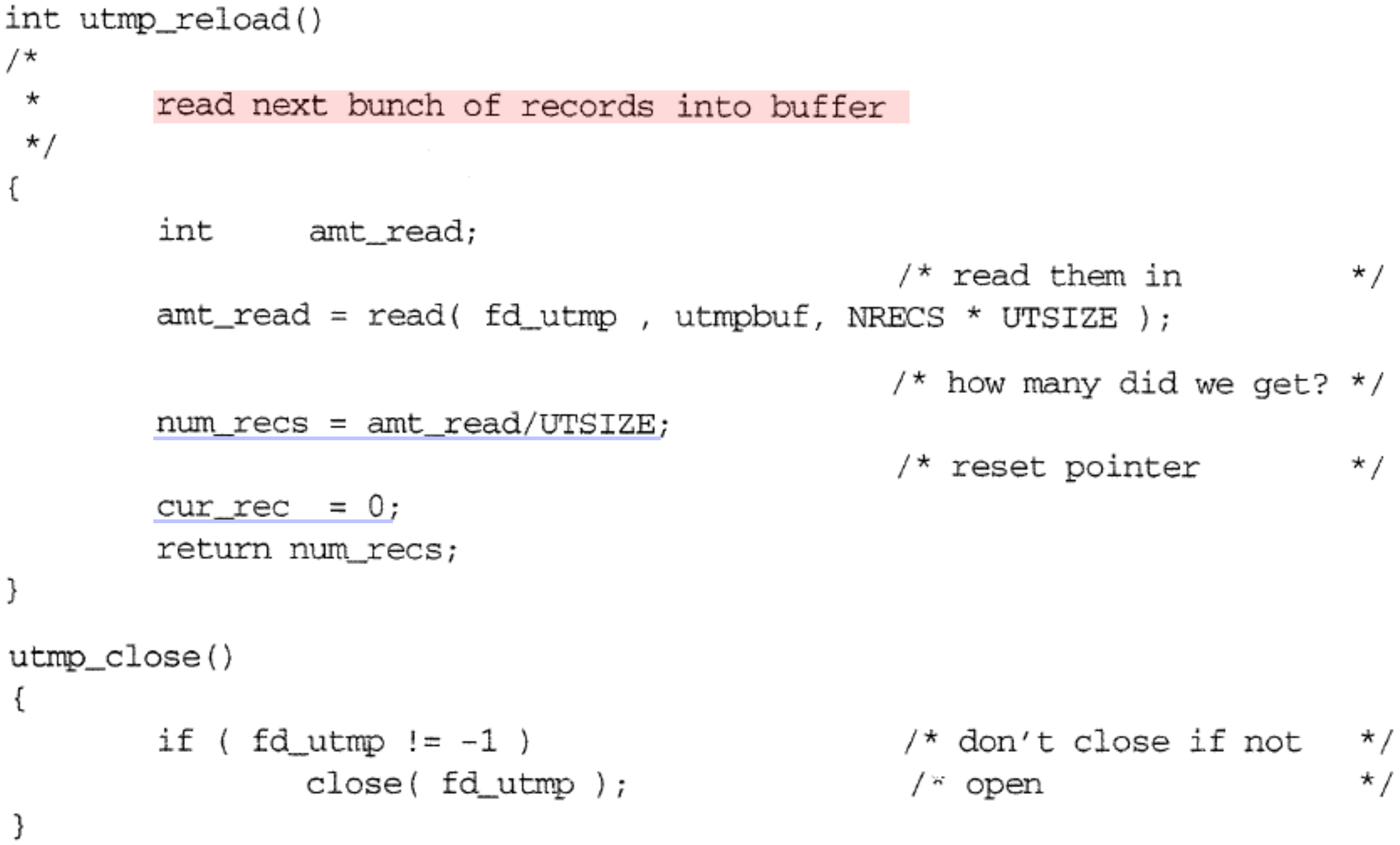
- The code for `utmplib.c (cont.)`



```
#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;
}
```

More Efficient File I/O: Buffering

- The code for `utmplib.c (cont.)`



```
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                 */
}
```

- Compile and Run

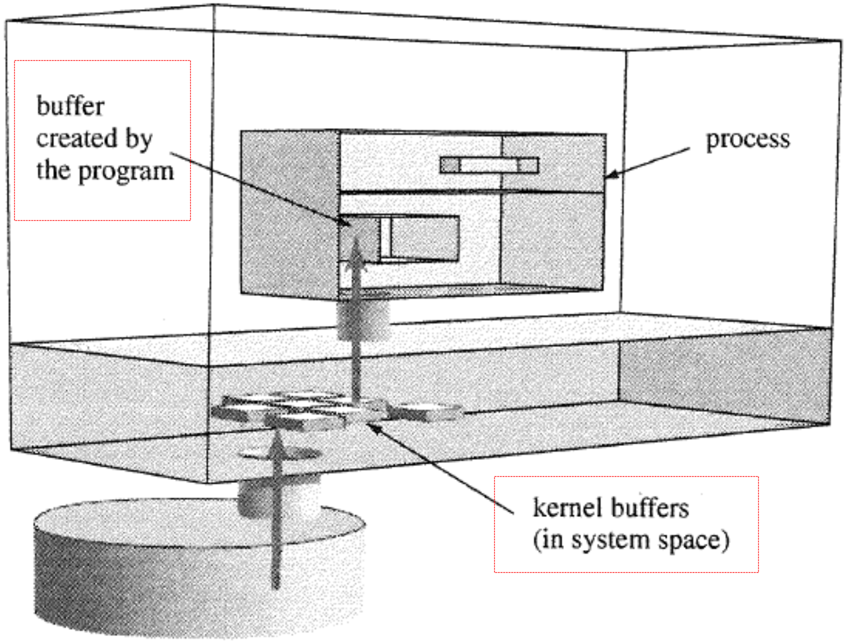
```
$ gcc who3.c umtplib.c –o who3
$ ./who3
```

Agenda

- 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
- 2.9 Reading and Writing a File
- 2.10 What to Do with System-Call Errors

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

- It does!
  - To save time, the kernel keeps copies of disk blocks in memory
  - The read() copies data into a process buffer from a kernel buffer, and write() copies data from the process buffer to a kernel buffer



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.

Agenda

- 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
- 2.9 Reading and Writing a File
- 2.10 What to Do with System-Call Errors

Logging Out: What It Does

- The system changes a record in the utmp file.
  - Specifically what changes?

- Experiment to see how it works:
  - 1. Log in to one machine.
  - 2. Use the who1 program we wrote to see the contents of `utmp`.
  - 3. Log out of one of your sessions.
  - 4. Repeat 1-3 to see what happened to the `utmp` record.

Logging Out: How It Works

- The program that removes your name from the log has to do the following:
  - **1. Open the utmp file.**

```
fd = open(UTMP_FILE, O_RDWR);
```

  - **2. Read the utmp file until it finds the record for your terminal.**

```
While( read(fd, rec, utmplen) == utmplen )    /* get next record */
        If( strcmp(rec.ut_line, myline) == 0)     /* what, my line?  */
                revise_entiry();                            /* remove my name  */
```

Logging Out: How It Works (cont.)

- The program that removes your name from the log has to do the following:
  - **3. Write a revised utmp record in its place.**
    - How do we write the revised record back to the file?
      - If we just call write, our code updates the next record

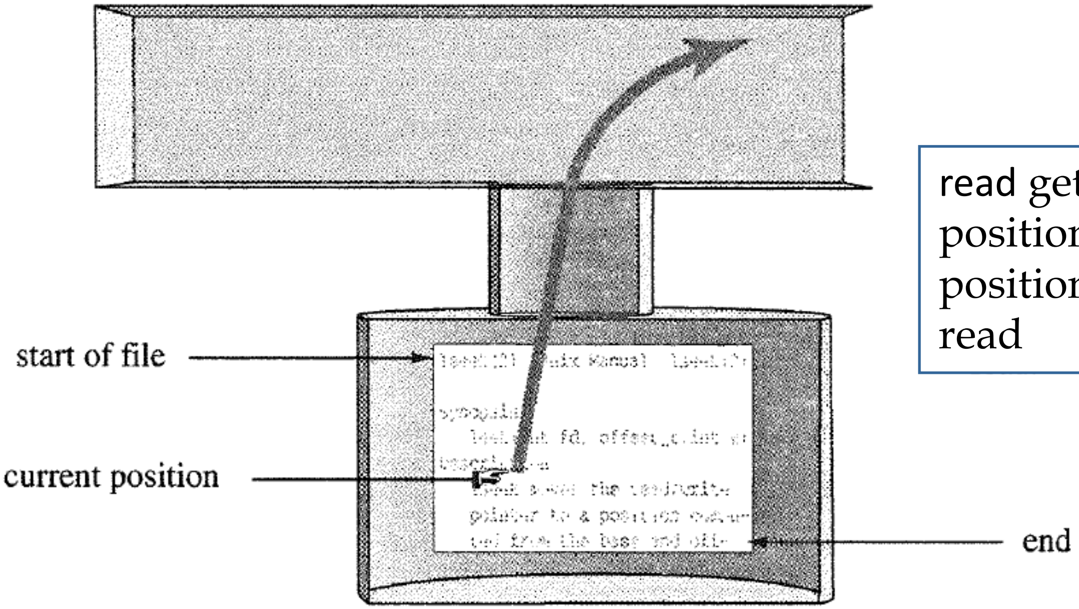
Q: How can a program change the current read-write position in a file?
A: The lseek system call.

- **4. Close the utmp file.**

```
close(fd);
```

Moving the Current Position: lseek()

- Every open file has a current position
  - The current position belongs to the connection to the file, not to the file



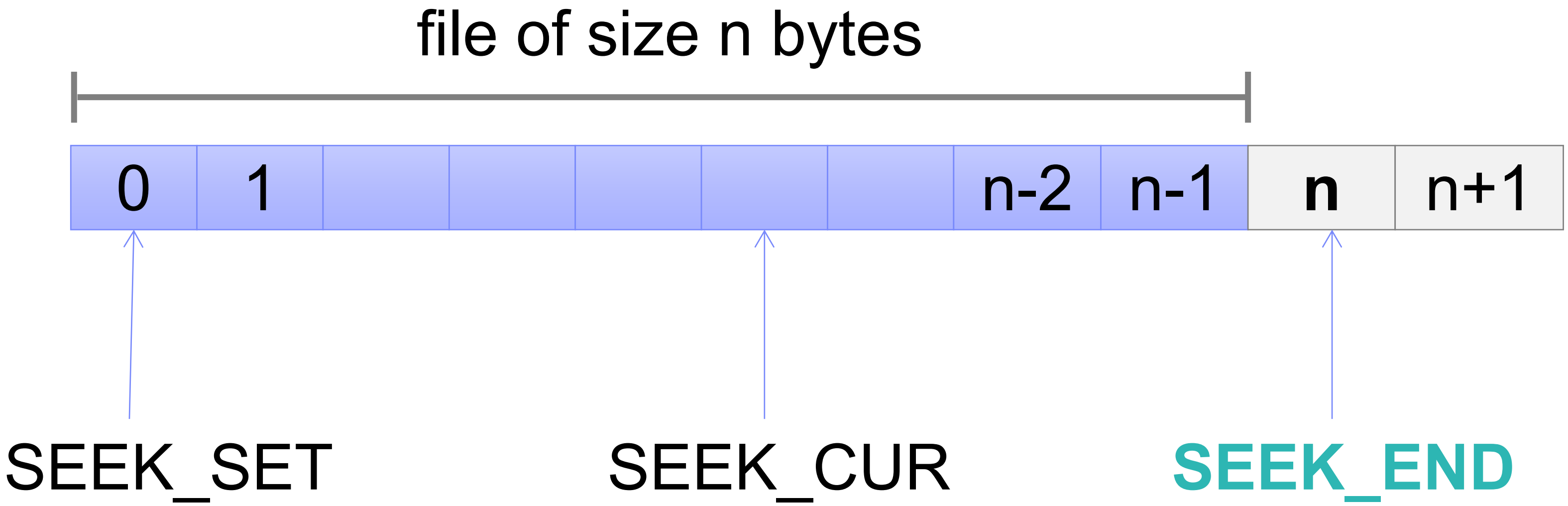
read gets data at the current position and advances the current position by the number of chars read

FIGURE 2.8

Every open file has a current position.

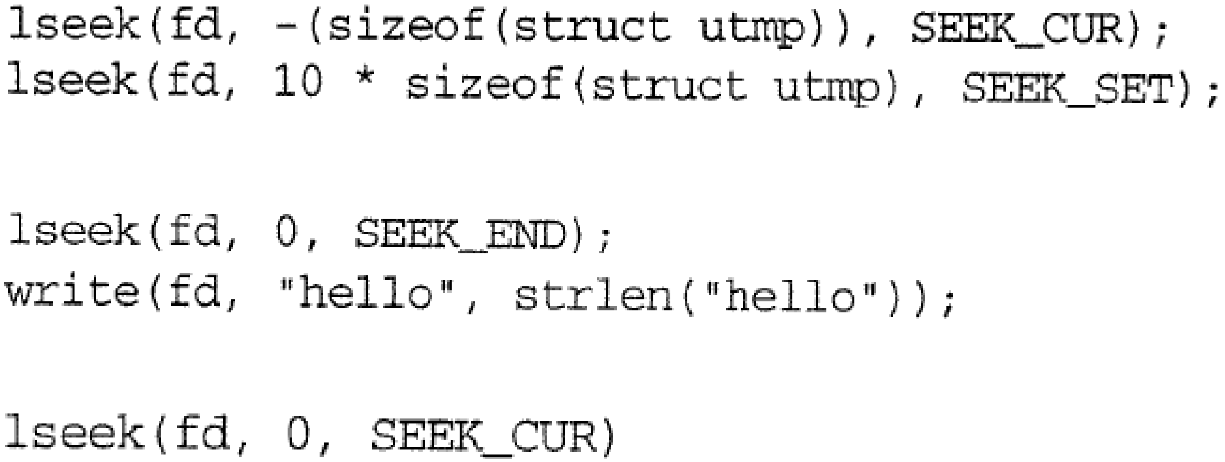
Moving the Current Position : lseek()

- offset: location in bytes
- whence: SEEK_SET, SEEK_CUR, SEEK_END



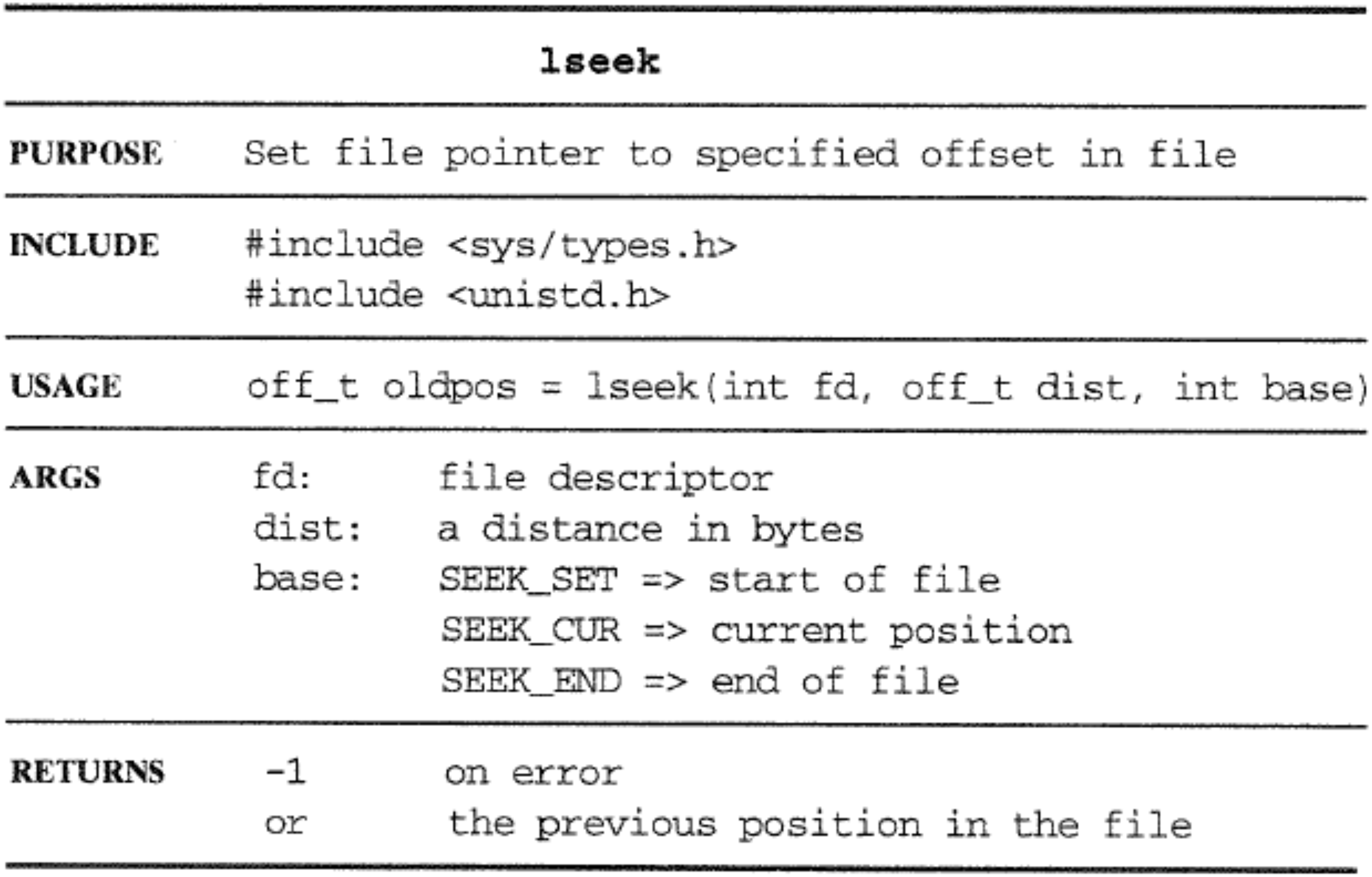
Moving the Current Position: lseek()

- The lseek() system call lets you change the current position of an open file:



```
lseek(fd, -(sizeof(struct utmp)), SEEK_CUR);
lseek(fd, 10 * sizeof(struct utmp), SEEK_SET);


lseek(fd, 0, SEEK_END);
write(fd, "hello", strlen("hello"));


lseek(fd, 0, SEEK_CUR)
```

Moving the Current Position: lseek()

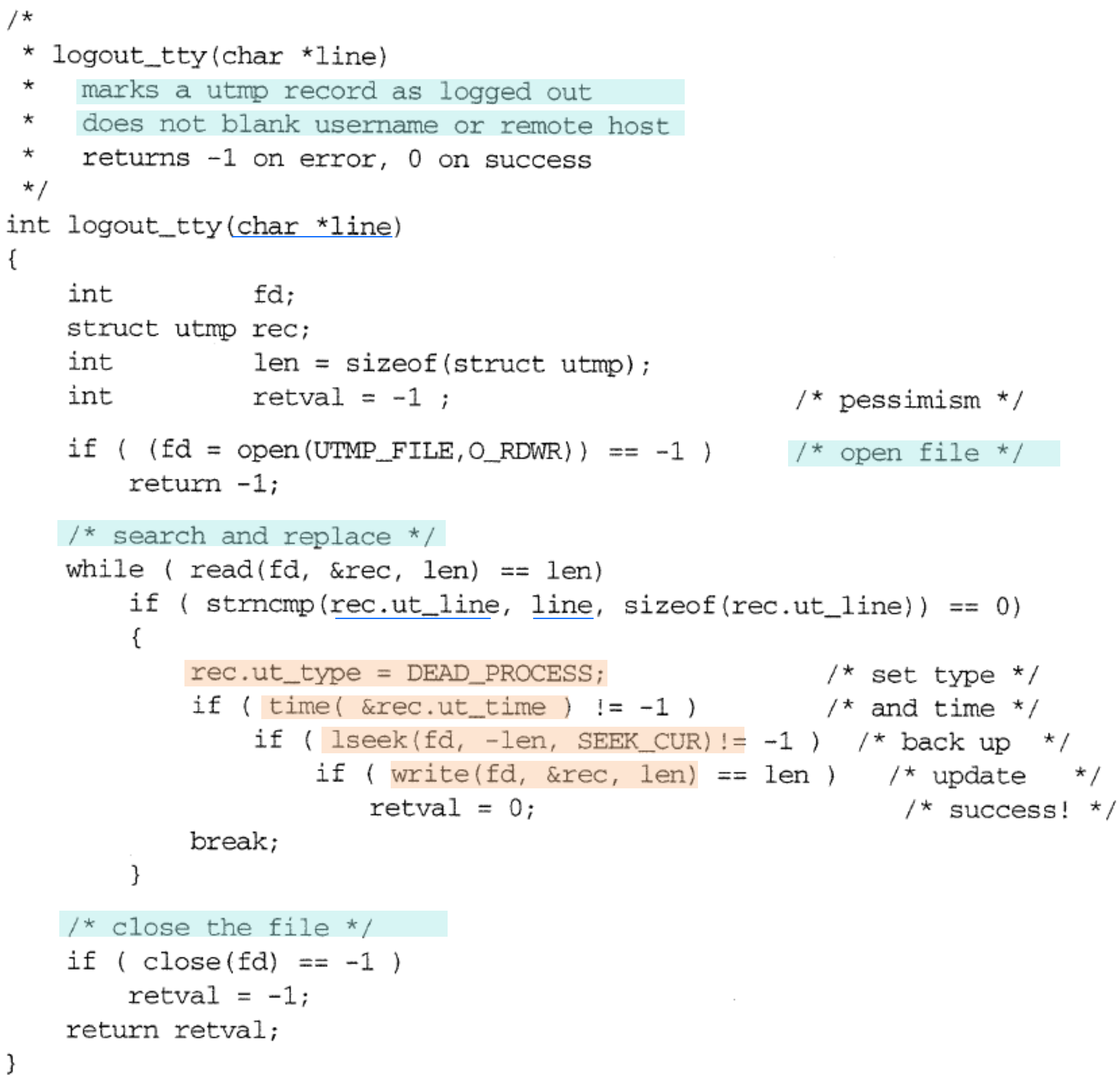


lseek	
PURPOSE	Set file pointer to specified offset in file

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

USAGE	off_t oldpos = lseek(int fd, off_t dist, int base)
ARGS	fd: file descriptor dist: a distance in bytes base: SEEK_SET => start of file SEEK_CUR => current position SEEK_END => end of file
RETURNS	-1 on error or the previous position in the file

Code to Log Out from a Terminal



```
/*
 * logout_tty(char *line)
 *   marks a utmp record as logged out
 *   does not blank username or remote host
 *   returns -1 on error, 0 on success
 */
int logout_tty(char *line)
{
    int          fd;
    struct utmp rec;
    int          len = sizeof(struct utmp);
    int          retval = -1 ;                      /* pessimism */

    if ( (fd = open(UTMP_FILE,O_RDWR)) == -1 )      /* open file */
        return -1;

    /* search and replace */
    while ( read(fd, &rec, len) == len)
        if ( strncmp(rec.ut_line, line, sizeof(rec.ut_line)) == 0)
        {
            rec.ut_type = DEAD_PROCESS;                    /* set type */
            if ( time( &rec.ut_time ) != -1 )              /* and time */
                if ( lseek(fd, -len, SEEK_CUR) != -1 )    /* back up  */
                    if ( write(fd, &rec, len) == len )    /* update   */
                        retval = 0;                        /* success! */
            break;
        }

    /* close the file */
    if ( close(fd) == -1 )
        retval = -1;
    return retval;
}
```

Agenda

- 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
- 2.9 Reading and Writing a File
- 2.10 What to Do with System-Call Errors

What to Do with System-Call Errors

- When open cannot open a file, it returns -1
- When read cannot read data, it returns -1
- When lseek cannot seek, it returns -1

- System calls return -1 when something goes wrong

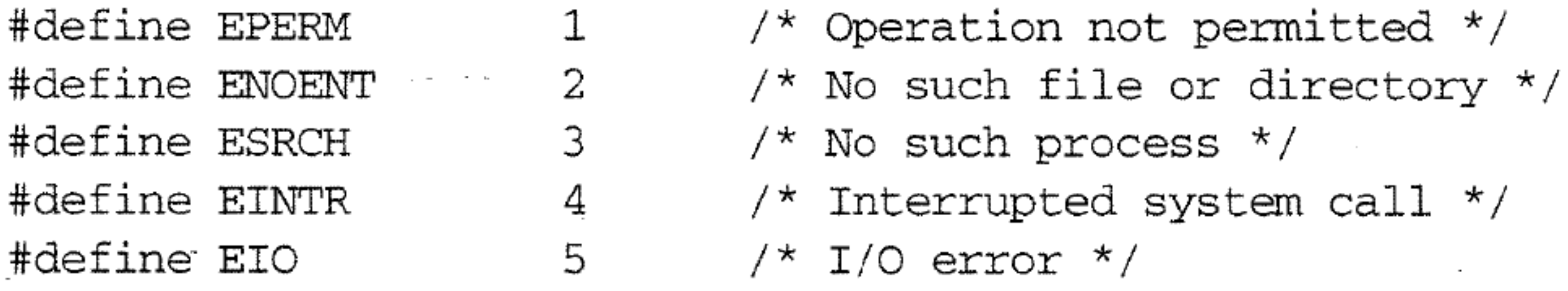
- Your programs should test the return value of every system call they make and take intelligent action when errors occur

How to Identify What Went Wrong: errno

- The kernel tells your program the cause of the error by storing an error code in a **global variable** called **errno**

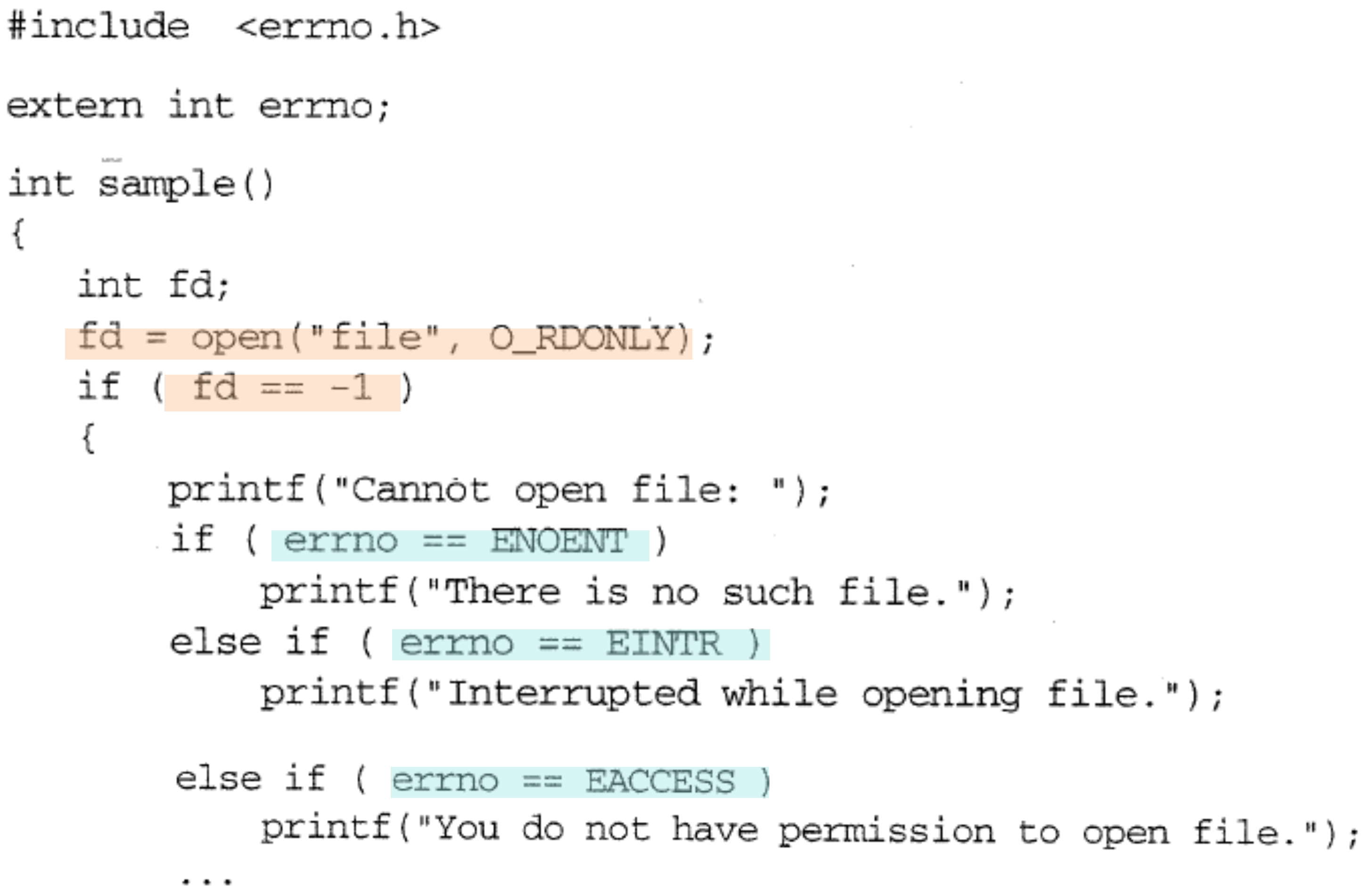
- The manpage for `errno(3)` and the file `<errno.h>` include the error-code symbols and numeric codes

```
$ 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 */
```

Different Responses to Different Errors

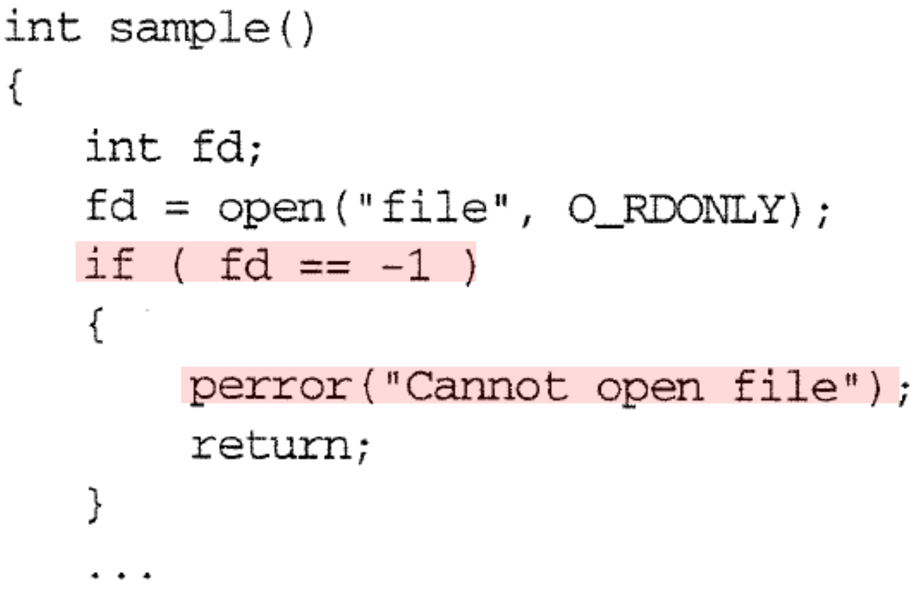


```
#include  <errno.h>

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.");
        ...
```

Reporting Errors: perror(3)

- If you want to print a message describing the error, you could test the value of errno and print different messages for different values.

- Print a system error message



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

Objectives

- **Ideas and Skills**
  - The role and use of on-line documentation (man)
  - 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
- **UNIX/LINUX Commands**
  - man, who, cp, login

Next Time

- Chapter 3 : Directories and File Properties

Homework#1

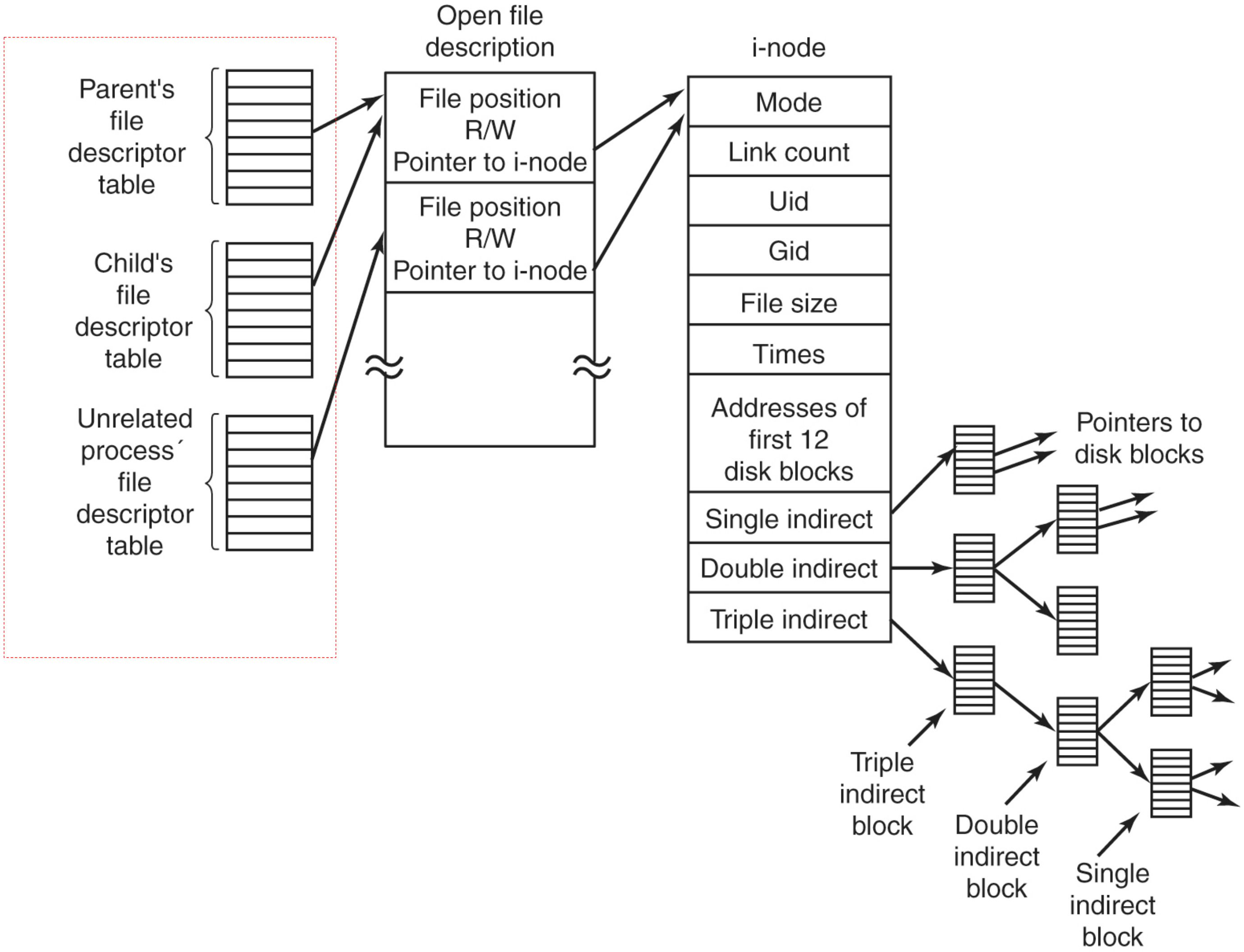
- Complete who3 and cp1 programs
  - who3: who3.c
  - cp1 : cp1.c
- Compressing the source code of the programs to LMS
  - Compressed file name : student id_hw1
- We will just check if the programs are working
- Due: 11:59:59 pm, September 11, 2019

Appendix: File descriptor

- All system calls use file descriptor to refer to open files
  - regular files, pipes, FIFO, sockets, terminals, devices
- Each process has its own list of open fd
- Three standard fd opened by default
  - inherited from the shell

FD	Purpose	POSIX name	stdio stream
0	standard input	STDIN_FILENO	stdin
1	standard output	STDOUT_FILENO	stdout
2	standard error	STDERR_FILENO	stderr

Appendix: File descriptor


Open file description
i-node
Parent's file descriptor table
Child's file descriptor table
Unrelated process´ file descriptor table
File position
R/W
Pointer to i-node
File position
R/W
Pointer to i-node
Mode
Link count
Uid
Gid
File size
Times
Addresses of first 12 disk blocks
Single indirect
Double indirect
Triple indirect
Pointers to disk blocks
Triple indirect block
Double indirect block
Single indirect block