System Programming
(ELEC462)

Users, Files, and the Manual

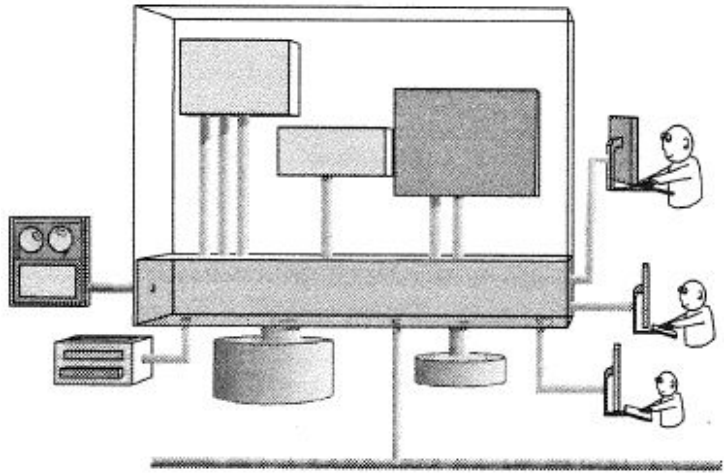
Dukyun Nam

KNU

Contents

Introduction

- Every multiuser computer system has a `who` command
  - The command tells you who else is using the computing
  - How does it work?



< Users, files, processes, devices, and kernel >

- Recall: Unix system
  - The large box is **computer memory**
    - It is divided into user space and system space
  - Various **programs** are running in user space
  - These programs communicate to the outside world through the **kernel**

Introduction (cont.)

- All Unix commands are programs
  - Written by a variety of people, usually in C
    - e.g., `who` and `ls`
  - When you type `ls`, you are asking your shell to run the program named `ls`
  - The `ls` program lists the names of files in the directory
- To add new commands to Unix
  - You write a new program and have the executable file stored in one of standard system directories
    - e.g., `/bin`, `/usr/bin`, `/usr/local/bin`

Project 1: `who` command

```
$ 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)
$
```

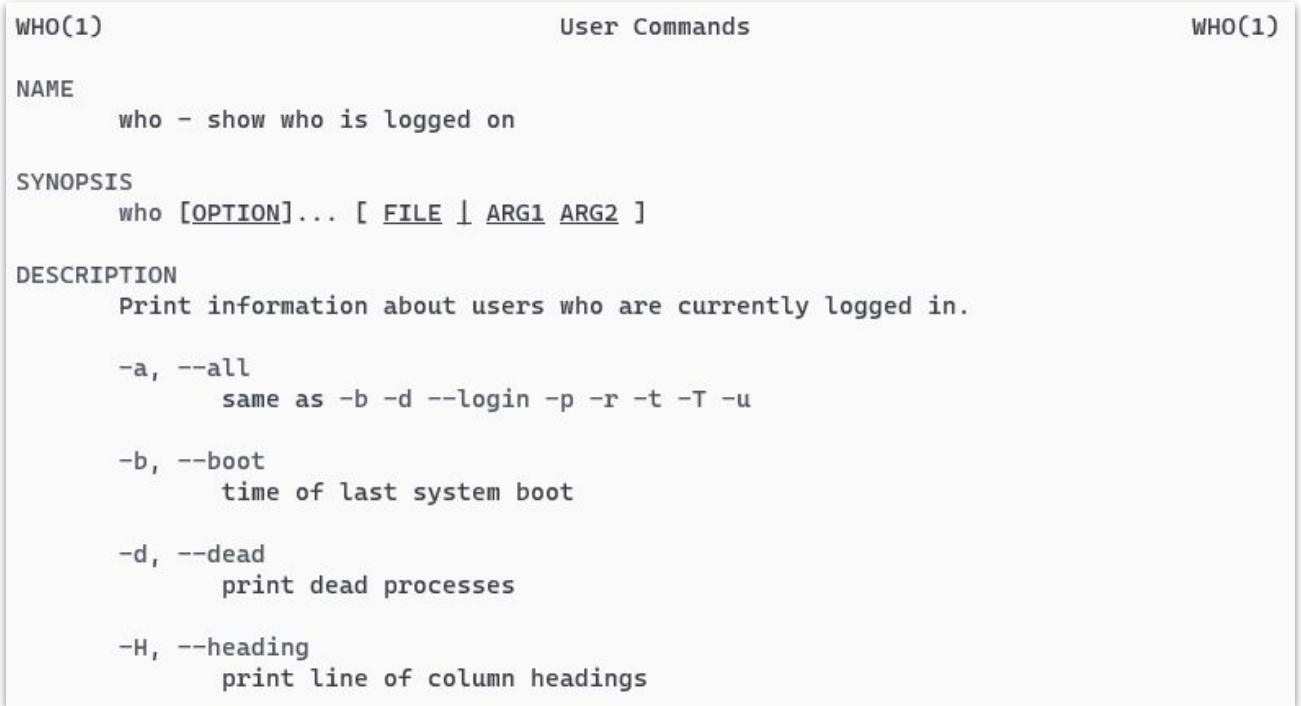
< Example: `who` command >

- Purpose
  - List users currently logged on
- Output
  - logname, terminal, time, from where

How Does `who` Do It?

- Read the manual

```
$ man who
...
       If FILE is not specified, use  /var/run/utmp.  /var/log/wtmp as FILE is
common.   If  ARG1 ARG2 given, -m presumed: 'am i' or 'mom likes' are usual.
...
```



```
WHO(1)                          User Commands                          WHO(1)

NAME
       who - show who is logged on

SYNOPSIS
       who [OPTION]... [ FILE | ARG1 ARG2 ]

DESCRIPTION
       Print information about users who are currently logged in.

       -a, --all
              same as -b -d --login -p -r -t -T -u

       -b, --boot
              time of last system boot

       -d, --dead
              print dead processes

       -H, --heading
              print line of column headings
```

How Does `who` Do It? (cont.)

- Search the manual

```
$ man -k utmp
```

```
endutent (3)         - access utmp file entries
endutxent (3)        - access utmp file entries
getutent (3)         - access utmp file entries
getutent_r (3)       - access utmp file entries
getutid (3)          - access utmp file entries
getutid_r (3)        - access utmp file entries
getutline (3)        - access utmp file entries
getutline_r (3)      - access utmp file entries
getutmp (3)          - copy utmp structure to utmpx, and vice versa
getutmpx (3)         - copy utmp structure to utmpx, and vice versa
getutxent (3)        - access utmp file entries
getutxid (3)         - access utmp file entries
getutxline (3)       - access utmp file entries
login (3)            - write utmp and wtmp entries
logout (3)           - write utmp and wtmp entries
pututline (3)        - access utmp file entries
pututxline (3)       - access utmp file entries
sessreg (1)          - manage utmpx/wtmpx entries for non-init clients
setutent (3)         - access utmp file entries
setutxent (3)        - access utmp file entries
systemd-update-utmp (8) - Write audit and utmp updates at bootup, runlevel changes and...
systemd-update-utmp-runlevel.service (8) - Write audit and utmp updates at bootup, run...
systemd-update-utmp.service (8) - Write audit and utmp updates at bootup, runlevel cha...
utmp (5)             - login records
utmpdump (1)         - dump UTMP and WTMP files in raw format
utmpname (3)         - access utmp file entries
utmpx (5)            - login records
utmpxname (3)        - access utmp file entries
```

How Does `who` Do It? (cont.)

- Read the `.h` files (The manpage of `utmp`)

```
$ man 5 utmp
$ less /usr/include/utmp.h
```

  - We can see the structure of the records in the `utmp` file

```
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 */
};
```

- Follow SEE ALSO links

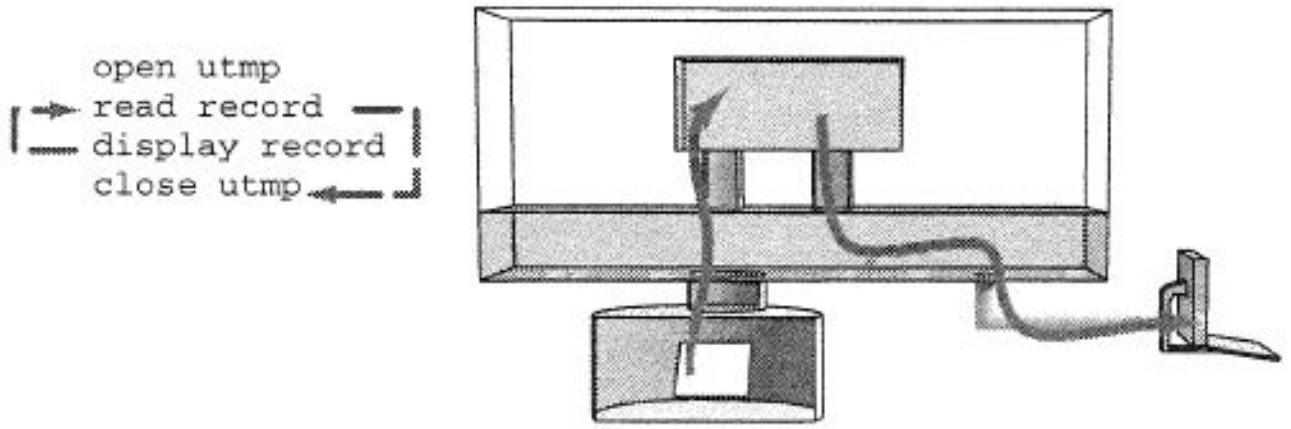
utmp, wtmp, and btmp

- utmp, wtmp, and btmp are files on Unix-like systems
  - Keep track of all logins and logouts to the system
  - **utmp** maintains a full accounting of the current status of the system, system boot time (used by uptime), recording user logins at which terminals, logouts, system events, etc.
  - **wtmp** acts as a historical utmp
  - **btmp** records failed login attempts
- Location
  - /var/run/utmp, /var/log/wtmp, /var/log/btmp

How Does `who` Work?

- We learned how `who` works
  - 1) By reading the on-line manual on the topics of `who` and `utmp`
  - 2) By reading the header file `utmp.h`
- `who` reads structures from a file
  - The file contains one structure for each login session
    - A login session: *the period of activity* between a user logging in and logging out of a system (from wikipedia)
  - We know the exact layout of the structure

How Does `who` Work? (cont.)



< Data flow in the `who` command >

- The file is an array
  - `who` must read the records and print out the information

Can I Write `who`?

- Two tasks we need to program
  - Read structs from a file
  - Display the information stored in a struct
- How to read the structs from a file?
  - If you have used `getc()` and `fgets()`, you know how to read characters and lines
  - But what about structs of *raw data*?

Can I Write `who`? (cont.)

- Read the manual

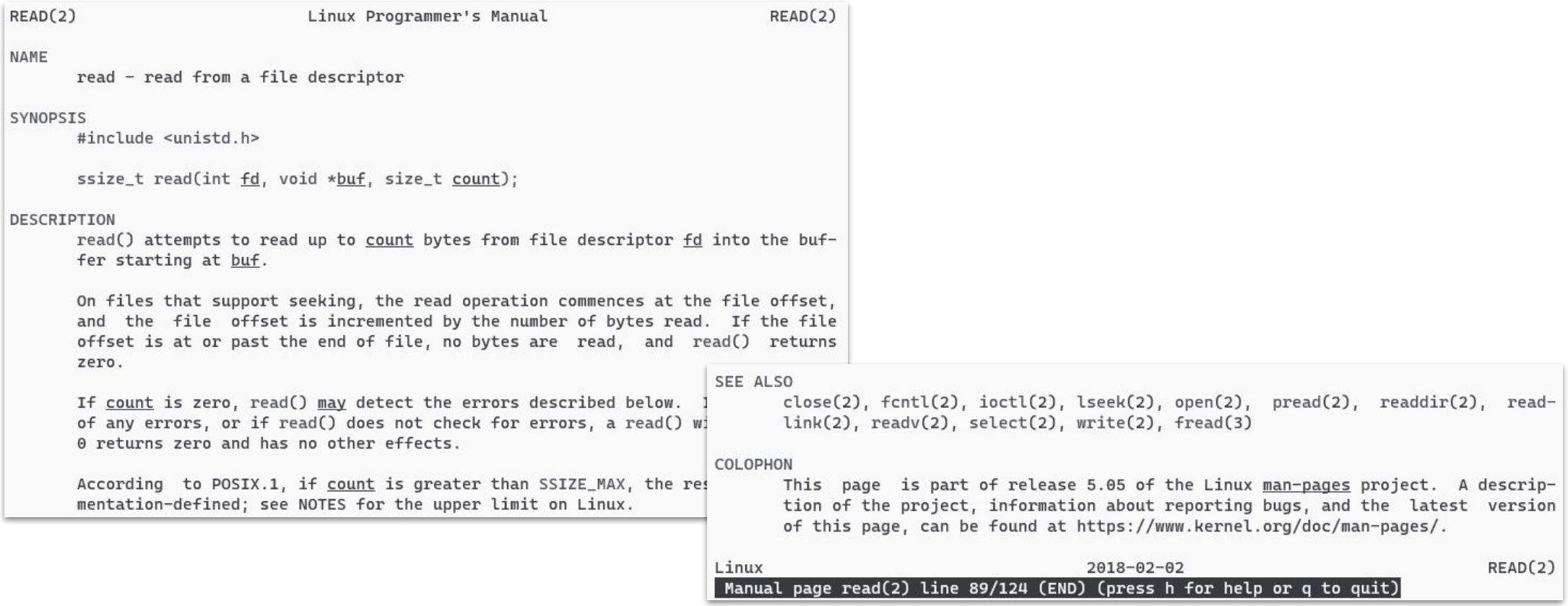
```
$ man -k file | grep read
```

```
__freadable (3)      - interfaces to stdio FILE structure
__freading (3)       - interfaces to stdio FILE structure
_llseek (2)          - reposition read/write file offset
eventfd_read (3)     - create a file descriptor for event notification
fc-cat (1)           - read font information cache files
fgetwc (3)           - read a wide character from a FILE stream
fgetws (3)           - read a wide-character string from a FILE stream
file2brl (1)         - Translate an xml or a text file into an embosser-ready braille ...
fts_read (3)         - traverse a file hierarchy
getwc (3)            - read a wide character from a FILE stream
git-prune-packed (1) - Remove extra objects that are already in pack files
llseek (2)           - reposition read/write file offset
lseek (2)            - reposition read/write file offset
lseek64 (3)          - reposition 64-bit read/write file offset
pppdump (8)          - convert PPP record file to readable format
pread (2)            - read from or write to a file descriptor at a given offset
pread64 (2)          - read from or write to a file descriptor at a given offset
pwrite (2)           - read from or write to a file descriptor at a given offset
pwrite64 (2)         - read from or write to a file descriptor at a given offset
read (2)             - read from a file descriptor
readahead (2)        - initiate file readahead into page cache
readelf (1)          - display information about ELF files
readfile (3am)       - return the entire contents of a file as a string
readlink (1)         - print resolved symbolic links or canonical file names
readprofile (8)      - read kernel profiling information
rwarray (3am)        - write and read gawk arrays to/from files
tee (1)              - read from standard input and write to standard output and files
X11::Auth (3pm)      - Perl module to read X11 authority files
x86_64-linux-gnu-readelf (1) - display information about ELF files
```

Can I Write `who`? (cont.)

- Look at the manpage in section 2 about `read`

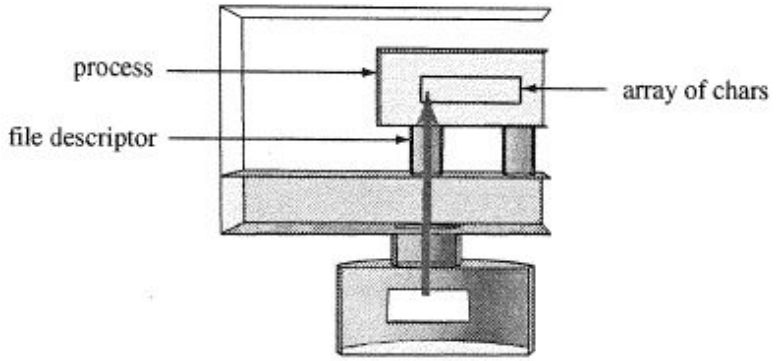
```
$ man 2 read
```



```
READ(2)                    Linux Programmer's Manual                    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 buf-
       fer starting at buf.

       On files that support seeking, the read operation commences at the file offset,
       and  the  file  offset is incremented by the number of bytes read.  If the file
       offset is at or past the end of file, no bytes are  read,  and  read()  returns
       zero.

       If count is zero, read() may detect the errors described below.  I
       of any errors, or if read() does not check for errors, a read() wi
       0 returns zero and has no other effects.

       According  to POSIX.1, if count is greater than SSIZE_MAX, the res
       mentation-defined; see NOTES for the upper limit on Linux.

SEE ALSO
       close(2), fcntl(2), ioctl(2), lseek(2), open(2),  pread(2),  readdir(2),  read-
       link(2), readv(2), select(2), write(2), fread(3)

COLOPHON
       This  page  is part of release 5.05 of the Linux man-pages project.  A descrip-
       tion of the project, information about reporting bugs, and the  latest  version
       of this page, can be found at https://www.kernel.org/doc/man-pages/.

Linux                              2018-02-02                           READ(2)
 Manual page read(2) line 89/124 (END) (press h for help or q to quit)
```

Can I Write `who`? (cont.)

- Answer: Use `open`, `read` and `close`
  - Use three system calls to extract log-in records from the `utmp` file
- Opening a file: `open`



< A file descriptor is a connection to a file >

  - The `open` system call creates a connection between a process and a file
  - That connection is called a *file descriptor* as a tunnel from the process to the kernel

Can I Write `who`? (cont.)

- Basic usage: open
  - Specify the name of file and the type of connection you want
  - The three types are a connection for *reading*, a connection for *writing*, and a connection for *reading and writing*

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`? (cont.)

- Basic usage: read
  - Ask the kernel to transfer *qty* bytes of data from the file descriptor *fd* to the array *buf* in the memory space of the calling process

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

Can I Write `who`? (cont.)

- Basic usage: close
  - Destroys the connection specified by file descriptor *fd*

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`

- Top-level code

```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 */
}
```

Writing `who1.c` (cont.)

- Displaying log-in records
  - `show_info` displays the information in the `utmp` records

```c
/*
 *  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("%10d", utbufp->ut_time);        /* login time   */
        printf(" ");                            /* a space      */
#ifdef  SHOWHOST
        printf("(%s)", utbufp->ut_host);        /* the host     */
#endif
        printf("\n");                           /* newline      */
}
```

Writing `who1.c` (cont.)

- Compare `who1` (left) and `who` (right)

```
$ gcc -o who1 who1.c
```

```
dynam@DESKTOP-Q4IJBP7:~/lab3$ ./who1
reboot   ~         1663163303 (5.10.102.1-microsoft-standard-WSL2)
LOGIN    console   1663163845 ()
dynam    pts/2     1663163315 ()
runlevel ~         1663163727 (5.10.102.1-microsoft-standard-WSL2)
```

```
dynam@DESKTOP-Q4IJBP7:~$ who
dynam    pts/2        2022-09-14 22:28
```

  - (Note) “who command produces no output on WSL2”
    - https://askubuntu.com/questions/1365678/who-command-produces-no-output-on-wsl2

```
$ sudo -b unshare --pid --fork --mount-proc /lib/systemd/systemd
--system-unit=basic.target
$ sudo login -f user_name
```

How Do We Get It to Look Good?

- Suppress blank records
  - The `utmp` file seems to have records for all terminals, even unused ones
  - Change our program so it does not print records for unused terminal lines
    - Print out `utmp` records that represent users logged into the system

```
/* Values for ut_type field, below */

#define EMPTY         0 /* Record does not contain valid info
                           (formerly known as UT_UNKNOWN on Linux) */
#define RUN_LVL       1 /* Change in system run-level (see
                           init(8)) */
#define BOOT_TIME     2 /* Time of system boot (in ut_tv) */
#define NEW_TIME      3 /* Time after system clock change
                           (in ut_tv) */
#define OLD_TIME      4 /* Time before system clock change
                           (in ut_tv) */
#define INIT_PROCESS  5 /* Process spawned by init(8) */
#define LOGIN_PROCESS 6 /* Session leader process for user login */
#define USER_PROCESS  7 /* Normal process */
#define DEAD_PROCESS  8 /* Terminated process */
#define ACCOUNTING    9 /* Not implemented */

#define UT_LINESIZE      32
#define UT_NAMESIZE      32
#define UT_HOSTSIZE     256
```

```
void show_info( struct utmp *utbufp )
{
        if ( utbufp->ut_type != USER_PROCESS )
                return;

        printf("%-8.8s", utbufp->ut_name);      /* the logname  */
```

How Do We Get It to Look Good? (cont.)

- Display the log-in time in human-readable form
  - Unix stores time as seconds since the beginning of the Epoch
  - `ctime()` converts to string

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

NAME
       asctime, ctime, gmtime, localtime, mktime, asctime_r, ctime_r, gmtime_r, local-
       time_r - transform date and time to broken-down time or ASCII

SYNOPSIS
       #include <time.h>

       char *asctime(const struct tm *tm);
       char *asctime_r(const struct tm *tm, char *buf);

       char *ctime(const time_t *timep);
       char *ctime_r(const time_t *timep, char *buf);
```

```
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     */
}
```

Writing `who2.c`

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

#define      SHOWHOST

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

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;
}
```

Writing `who2.c` (cont.)

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

```c
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     */
}
```

Writing `who2.c` (cont.)

- Compare `who2` and `who`

```
$ gcc -o who2 who2.c
```

```
dynam@DESKTOP-Q4IJBP7:~/lab3$ ./who2
dynam    pts/2    Sep 14 22:48
```

```
dynam@DESKTOP-Q4IJBP7:~$ who
dynam    pts/2        2022-09-14 22:28
```

Project 2: `cp` command

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

- What does `cp` do?
  - Creates or truncated a file, then writes data into it
- How does `cp` create and write?
  - Search the manual for the answer

```
CP(1)                          User Commands                          CP(1)

NAME
       cp - copy files and directories

SYNOPSIS
       cp [OPTION]... [-T] SOURCE DEST
       cp [OPTION]... SOURCE... DIRECTORY
       cp [OPTION]... -t DIRECTORY SOURCE...

DESCRIPTION
       Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

       Mandatory arguments to long options are mandatory for short options too.
```

How Does `cp` Create and Write?

- Creating/Truncating a file
  - One method to create or rewrite a file is the `creat` system call

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

How Does `cp` Create and Write? (cont.)

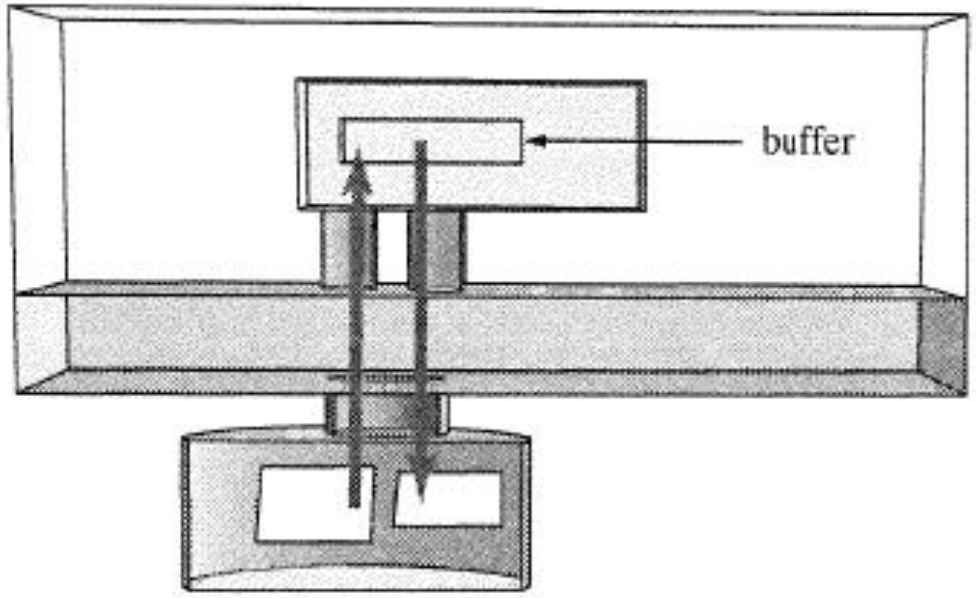
- Writing to a file
  - Data are sent to an open file with `write` system call

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

Can I Write `cp`?

- Program outline for `cp`

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



< Copying a file by reading and writing >

Can I Write `cp`? (cont.)

```c
/** 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>
#include        <stdlib.h>

#define BUFFERSIZE      4096
#define COPYMODE        0644

void oops(char *, char *);

int 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]);
```

```c
                                                /* 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","");

        return 0;
}

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

Can I Write `cp`? (cont.)

- Compile and test
  - The `cmp` utility compares two files and report differences
    - If there are no differences, the report shows nothing

```
dynam@DESKTOP-Q4IJBP7:~/lab3$ gcc -o cp1 cp1.c
dynam@DESKTOP-Q4IJBP7:~/lab3$ ./cp1 cp1 copy.of.cp1
dynam@DESKTOP-Q4IJBP7:~/lab3$ cmp cp1 copy.of.cp1
```

  - How does our program respond to errors?

```
dynam@DESKTOP-Q4IJBP7:~/lab3$ ./cp1 xxx123 file1
Error: Cannot open  xxx123: No such file or directory
dynam@DESKTOP-Q4IJBP7:~/lab3$ ./cp1 cp1 /tmp
Error: Cannot creat /tmp: Is a directory
```

More Efficient File I/O: Buffering

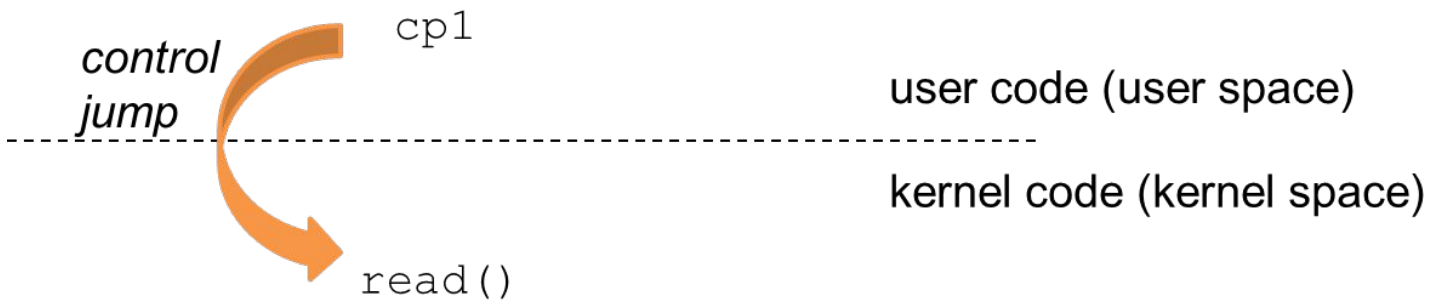
- Does buffer size matter?
  - If you use a ladle to transfer soup from one pot to another, a “larger” ladle requires fewer transfers and less time
  - e.g., Filesize = 2500 bytes
    - If buffer = 100 bytes, copy requires 25 read() and 25 write() calls
    - If buffer = 1000 bytes, how many?

More Efficient File I/O: Buffering (cont.)

- A system call is resource ‘expensive’ (i.e., 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

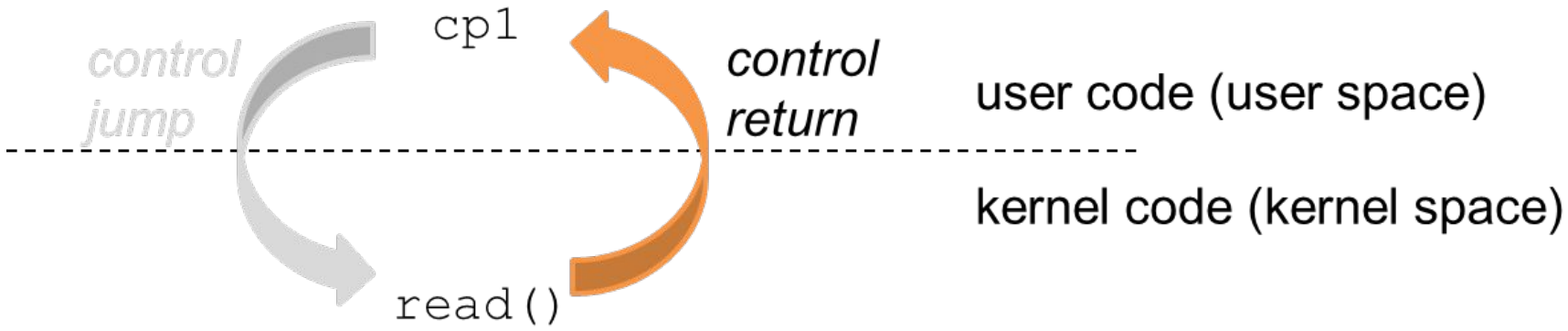
buffersize	execution time in seconds
1	50.29
4	12.81
16	3.28
64	0.96
128	0.56
256	0.37
512	0.27
1024	0.22
2048	0.19
4096	0.18
8192	0.18
16384	0.18

Why System Calls Are Time Consuming?



- Control flow in copying a file:
  - Suppose that our program `cp1` wants to read data
  - Then, `cp1` makes `read()` to ask the kernel for data
  - The code of actually transferring the data from the disk to the process is part of the kernel
  - Control, therefore, jumps from the code in user space to the kernel code in system space

Why System Calls Are Time Consuming? (cont.)



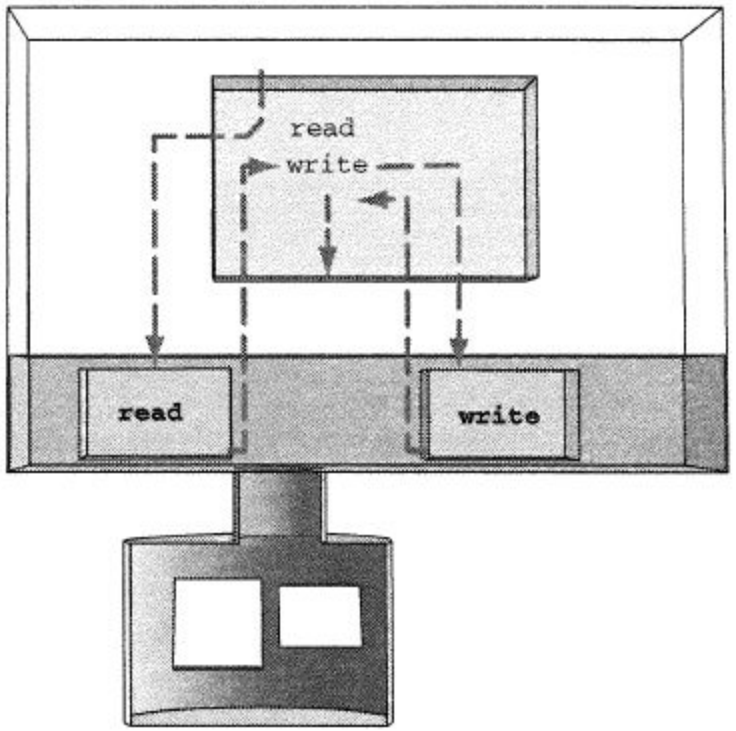
- The CPU then executes the kernel code to transfer data
- Time is spent on:
  - (1) Executing the code to transfer data, and
  - (2) Jumping into and out of the kernel

Why System Calls Are Time Consuming? (cont.)

- [Important] The CPU runs in kernel (or supervisor) mode with a special stack and memory environment when executing kernel code, and it runs in user mode when executing user code
  - The details of changing modes depend on the CPU
  - Linux must adapt to the modes supported by the CPU having its own notations of the modes; thus, causing time too

Why System Calls Are Time Consuming? (cont.)

- Control flow in copying a file



< Control flow during systems calls >

Does This Mean `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
- Better idea
  - Read a bunch of records at a time and then, as with eggs in a carton, take them one by one

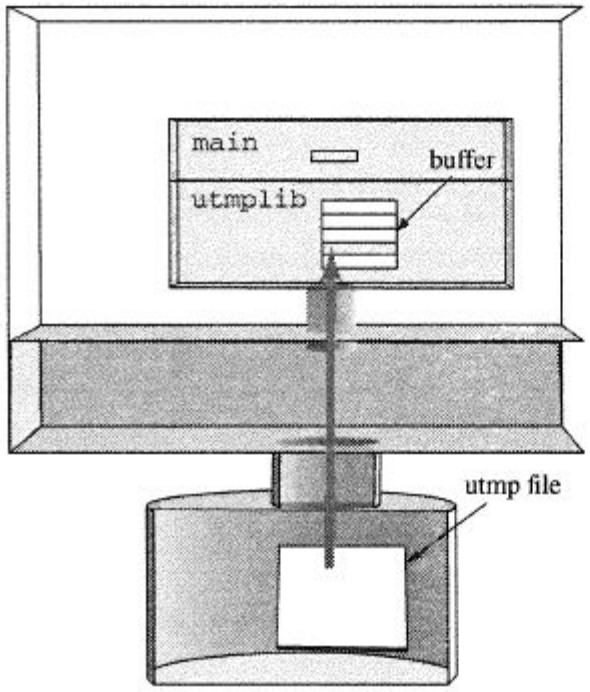
Does This Mean `who2.c` Is Inefficient?

- Revisit `who2.c`
  - Any problem?

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

#define      SHOWHOST

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

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;
}
```

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.

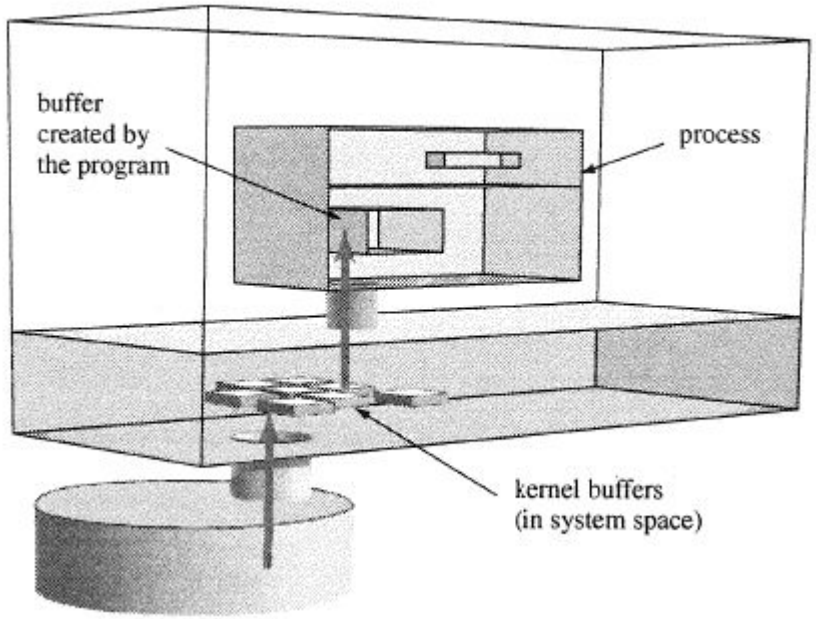
Functions in `utmplib.c` read structs 16 at a time from the disk into an array.

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

< Buffering disk data in user space >

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

- It DOES!
  - To save time (or, to reduce latency), the kernel keeps *copies of disk blocks* in memory
- Consequences of *Kernel Buffering*
  - (1) Faster "disk" I/O
  - (2) Optimized disk writes
  - (3) Need to write buffers to disk before shutdown



< Buffering disk data in the kernel >

Writing `who3`

- The file `utmplib.c` implements utmp record buffering:
  - `utmplib.c` contains the buffer and variables and functions to manage data flow through the buffer
  - The variables `num_recs` and `cur_rec` record how many structures are in the buffer and how many have been used

```
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    */
```

Writing `who3` (cont.)

- Each time a record is fetched, `utmp_next` checks if the `cur_rec` counter has reached the number of records in the buffer

```
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;
}
```

Writing `who3` (cont.)

- `utmplib.c`

```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
 */
#include        <stdio.h>
#include        <unistd.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    */

int utmp_reload();

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

```c
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;
}

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

Writing `who3` (cont.)

- Modify `main()` in `who2.c`

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

#define      SHOWHOST

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

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;
}
```

```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        <stdlib.h>
#include        <time.h>

#define SHOWHOST

int utmp_open( char *);
void utmp_close();
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;
}
```

Writing `who3` (cont.)

- Compiling `who3.c` and `utmplib.c` together

```
$ gcc -o who3 who3.c utmplib.c
```

Reading and Writing a File

- What happens when you log out?
  - One thing that happens is the system changes a record in the `utmp` file
  - A simple experiment to see how it works:
    - 1) Log in "twice," using two different terminals, to one machine
    - 2) Use `who1` you wrote to see the contents of `utmp`
      - Note what terminal lines you are using
    - 3) Log out of one of your sessions
    - 4) Run `who1` again to see what happened to those two `utmp` records
  - You will see that one of the records that contained your username has changed
  - Did this experiment make any other changes to the records?

Logging Out: What It Does

- The program that removes your name from the log has to do the following:
  - Step 1: Open the `utmp` file
  - Step 2: Read the `utmp` file until it finds the record for your terminal
  - Step 3: Write a revised `utmp` record in its place
  - Step 4: Close the `utmp` file

Logging Out: What It Does (cont.)

- Step 1: Open the `utmp` file
  - The log-out program reads from utmp and also writes to `utmp`, so the log-out program must open the file for reading and writing

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

    - fd: a handle (file descriptor) to UTMP_FILE
    - O_RDWR: open mode for read and write

Logging Out: What It Does (cont.)

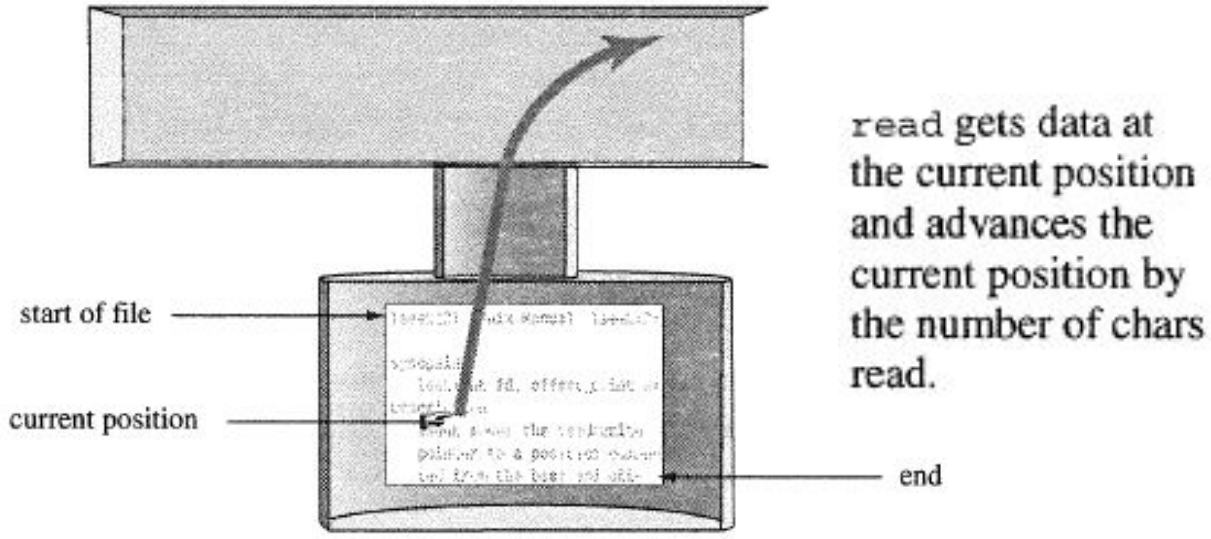
- Step 2: Read the `utmp` file until it finds the record for your terminal
  - A `while` loop can read one utmp record at a time and compare the `ut_line` member with the name of your terminal

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

Logging Out: What It Does (cont.)

- Step 3: Write a revised `utmp` record in its place
  - How to write the revised record back to the file?
    - What if we use `write()`?
    - Then, our code updates the ***next*** record
    - The kernel keeps track of our ***current position*** in the file and advances the current position each time bytes are read or written to the file
  - Q) If so, then how can a program change the *current read-write position* in a file?
    - A) The `lseek` system call
- Step 4: Close the `utmp` file
  - Call `close(fd)`

`lseek()`: Moving the Current Position



< Every open file has a current position >

- The kernel maintains a current position for each open file
- The current position belongs to the “connection” to the file, not to the file
  - If *two programs* open “the same file” at the same time, each connection has “its own” current position.

`lseek()`: Moving the Current Position (cont.)

- lseek sets the current position for open file *fd* to the position defined by the pair *dist* and *base*
  - The *base* may be (0) start of file, (1) current position, or (2) end of file

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

Sample Code to Log Out from a Terminal

- This code checks for errors from every system call it makes
  - Your system programs must always check every system call for errors
  - Leaving files inconsistent or incomplete can cause serious consequences

```
/*
        logout_tty(char* line)
        marks a utmp record as logged out
        does not blank username or remote host
        return -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;
}
```

What to Do with System-Call Errors

- System calls return -1 when something goes wrong
  - When `open` cannot open a file, it returns -1
  - When `read` cannot read data, it returns -1
  - When `lseek` cannot seek, it returns -1
- Your programs should
  - Test the return value of every system call they make
  - Take intelligent (or corresponding) 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`
  - errno - number of last error
- Here are a few examples

```
#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 */
```

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

Summary

- Chapter 2 explains the basics of Unix file input/output operations
- Every file can be seen as a sequence of characters, programs can interpret the contents of the file in any way they like
- The chapter explains the `who` command and shows how Unix keeps track of `who` is using the system by storing fixed size records in a file