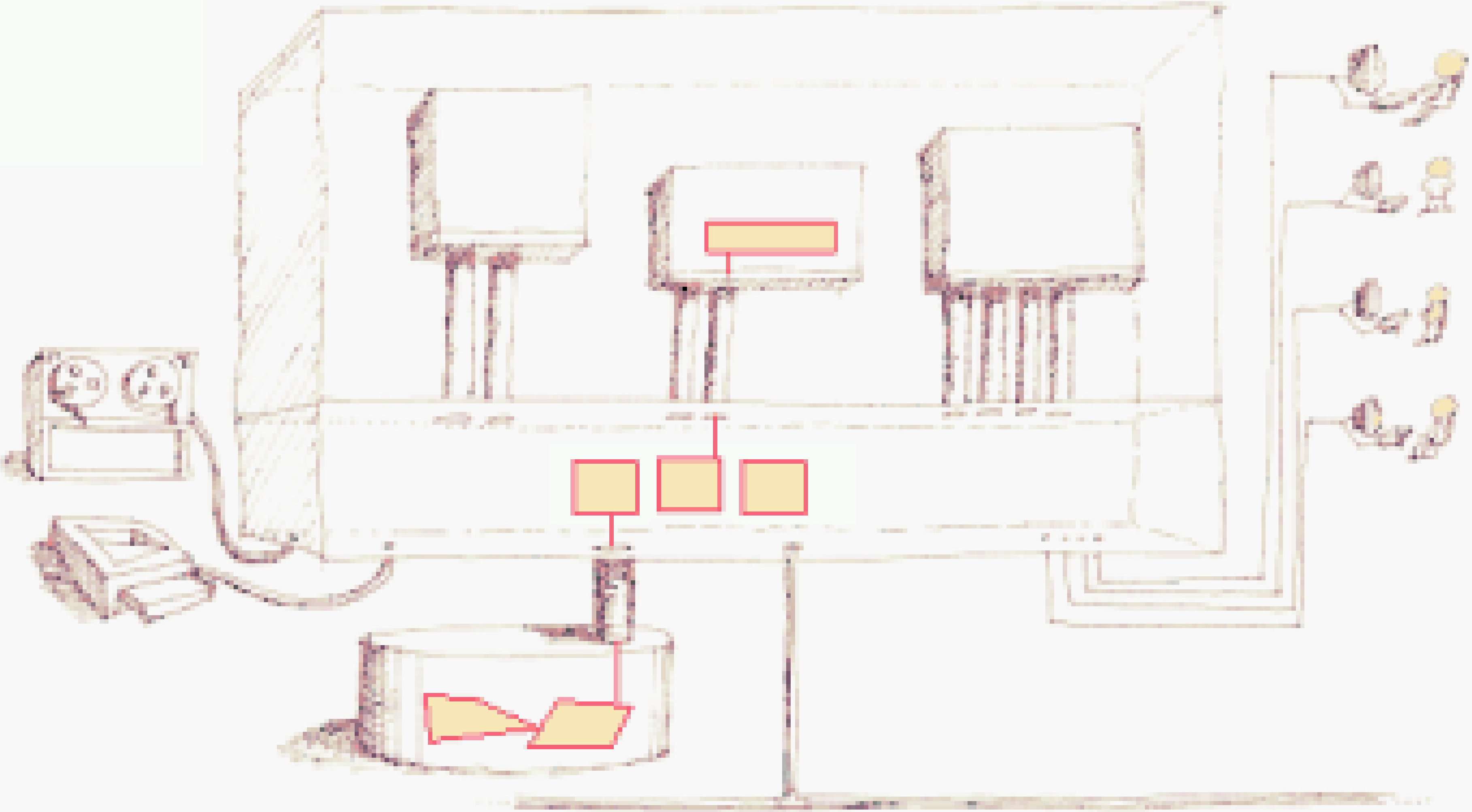
Lecture 2: Users, Files, On–Line Documents

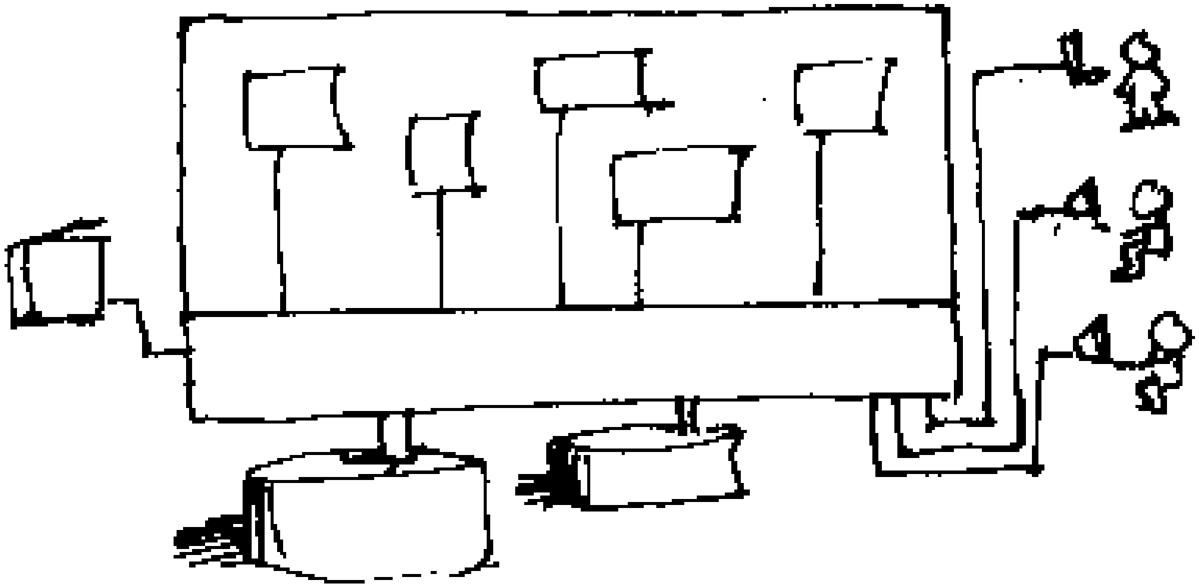


Class 2: Files, Users, On-line Documentation

Summary: Today we shall write a version of the who utility. In the process, we shall learn about:

☆ files, users, time, buffering, AND the fact that Unix is self-documenting. ☆

Recall:



II How Does who Work?

A. What does it do?

purpose:

output:

B. How does it do it?

C. How can I learn about the details?

II How Does who Work?

A. What does it do?

purpose: list users currently logged on

output: logname, terminal, time, from where

* By running the command we see what it does.

* By consulting the on-line manual..

```
man who
```

we learn even more.

II How Does who Work?

B. How does who do it?

To learn more about Unix commands:

* read the manual

```
man who
```

* search the manual

```
man -k user | more
```

* read the .h files in /usr/include

```
more /usr/include/foo.h
```

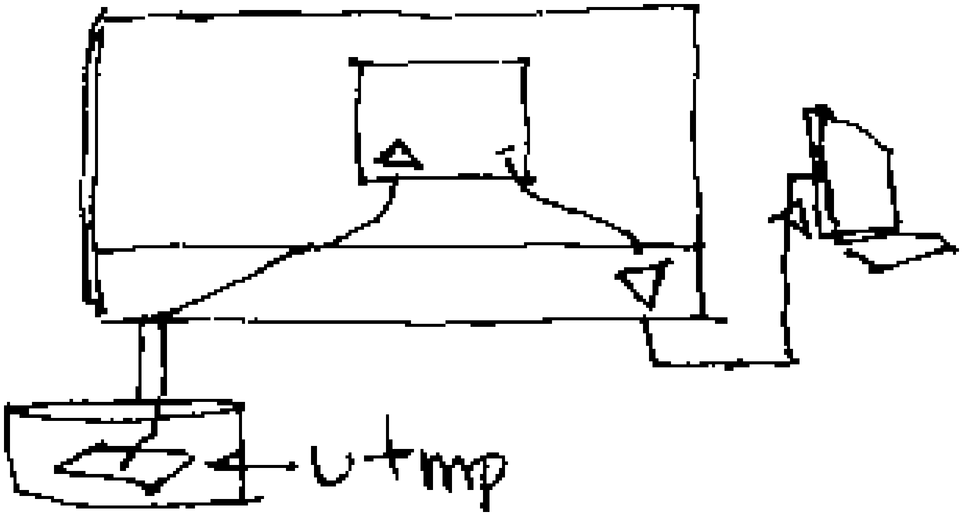
* follow the "See Also" links

Answer

who works by:

```
open wtmp
read record
display info
close file
```

eof



Q: How Do I Read 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?

What Does the Manual Say?

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

```
man 2 read
```

← *topic*

↑ *section*

Answer:

We use open(), read(), and close().

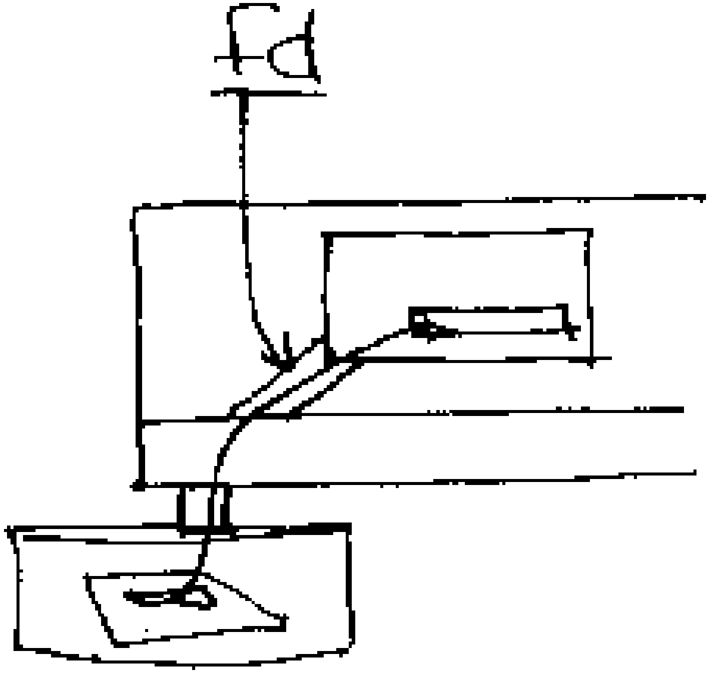
```
fd = open(name, mode)
```

char *

```
O_RDONLY
O_WRONLY
O_RDWR
```

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

creates a connection to a file.



Answer:

We use open(), read(), and close().

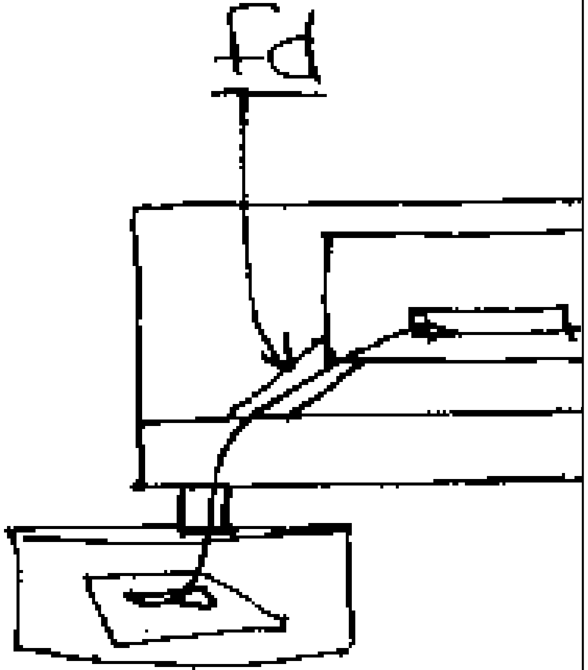
```
fd = open(name, mode)

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

- transfers numchars from file to the array

- returns number of chars actually transferred, OR -1 on error

```
close(fd)
```

- destroys the connection



Now we can write who1.c

It is getting pretty close,
but it needs some work..

IV. How Do We Get It to Look Good?

A. Suppress blank entries

B. Format time

Let's Check the Manual

(And the header files)

A. Suppress blank entries

check ut_type (see utmp.h)

B. Format time

ctime() converts to string
Unix stores time as seconds since the beginning of the Epoch

Result: who2.c

Moral: Unix provides complete documentation on system structures and programs

V. Project 2: cp (read and write)

In who, we read from a file.
How do we write to a file?
Let's explore a real example:

```
cp  sourcefile  copyfile
```

(1) What does cp do?

ANS: creates or truncates a file,
then writes data into it

(2) How does cp create and write?

ANS: search the manual for
the answers

Creating/Truncating A File

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

First: creates or truncates a file
Then: opens the file for writing
If create, set mode to, 0644

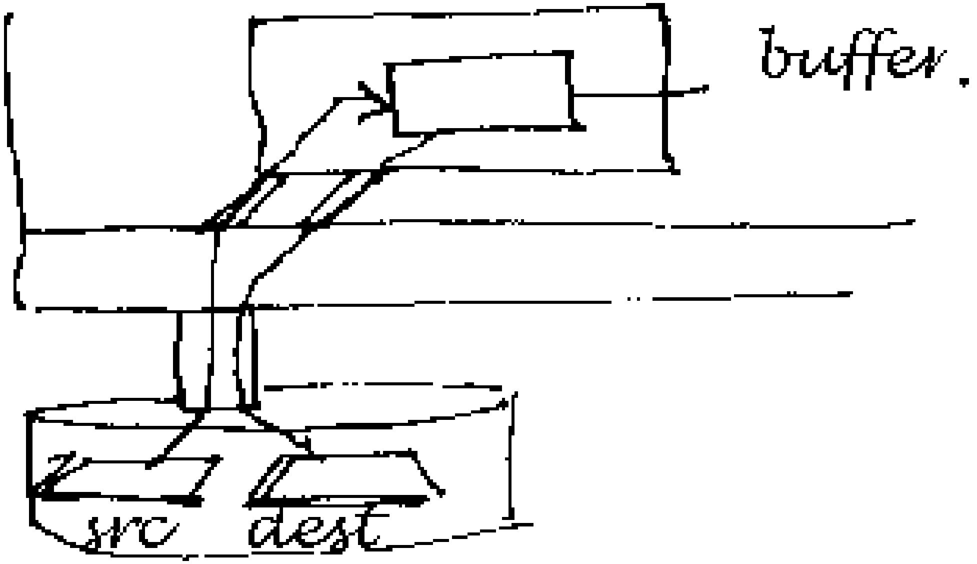
returns -1 on error,
otherwise a file descriptor

Writing to A File

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

* sends 'num' chars to the file
* returns actual number sent
OR returns -1 on error

Writing Ucopy.c



Logic:

```
      open sourcefile
      creat copyfile

 -->  read source → buffer ---+  eof?
|                             |
 ---  write buffer → copy     |
                              |
                              |
      close <-----------------+
      close
```

VI. Does Buffer Size Matter?

Ex: Filesize = 2500 bytes
if buffer = 100 bytes,
=> 25 read() and 25 write() calls

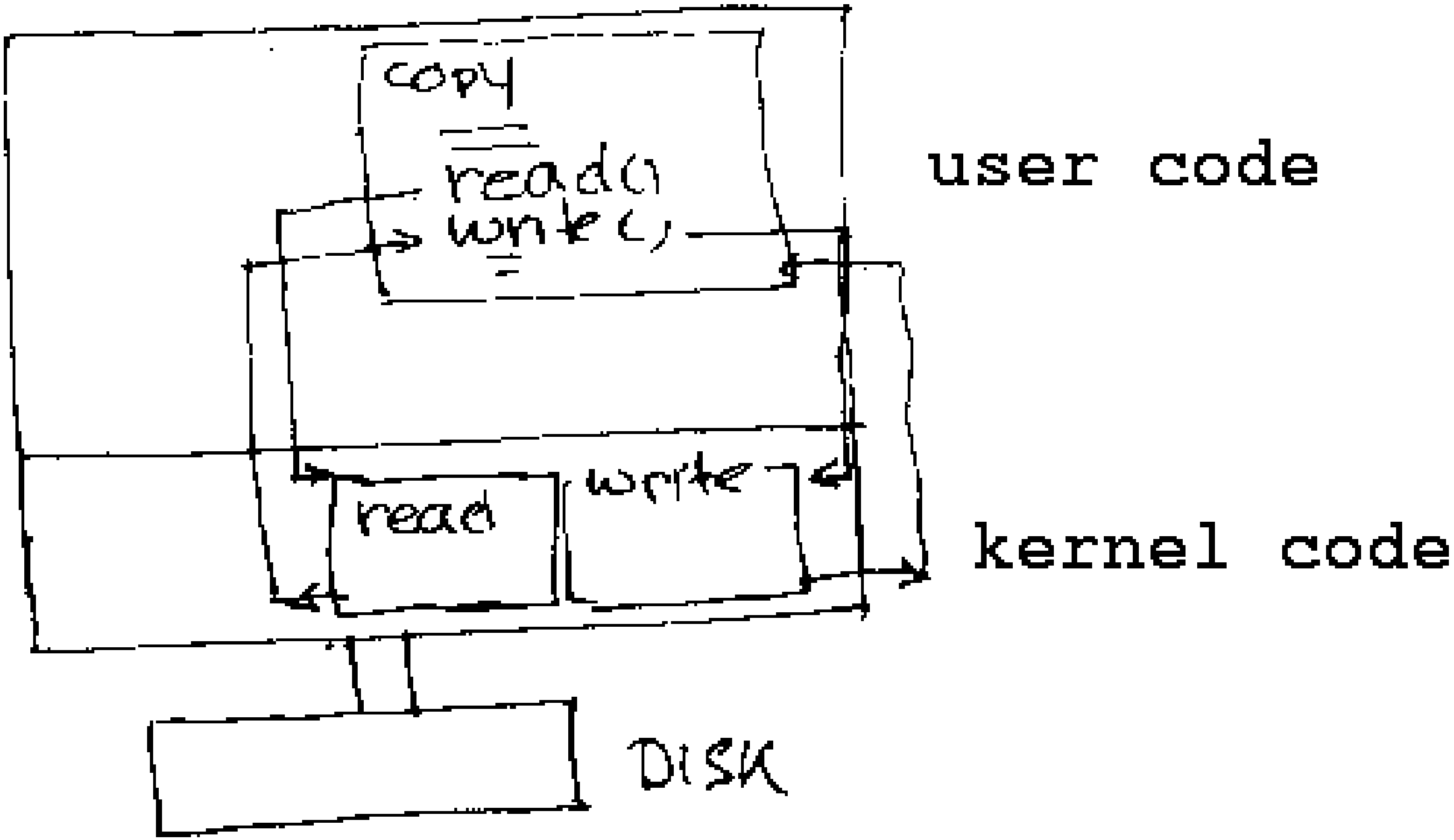
if buffer = 1000 bytes
=> 3 read() and 3 write() calls

Important Idea:

A system call is resource `expensive' (i.e. takes time). It runs various kernel functions, and it also requires a shift from USER MODE to KERNEL MODE and back.

** Thus, try to minimize system calls **

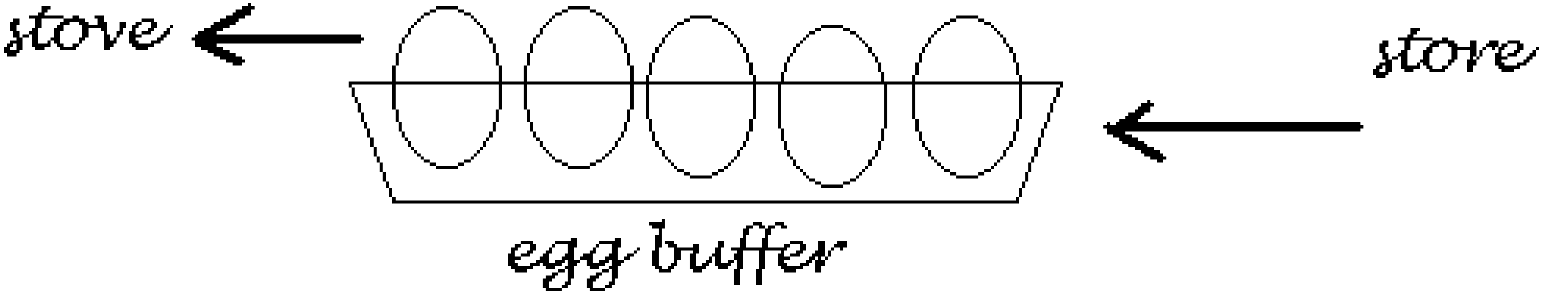
Control flow in copying a file:



VII. Does This Mean who2.c Is Inefficient?

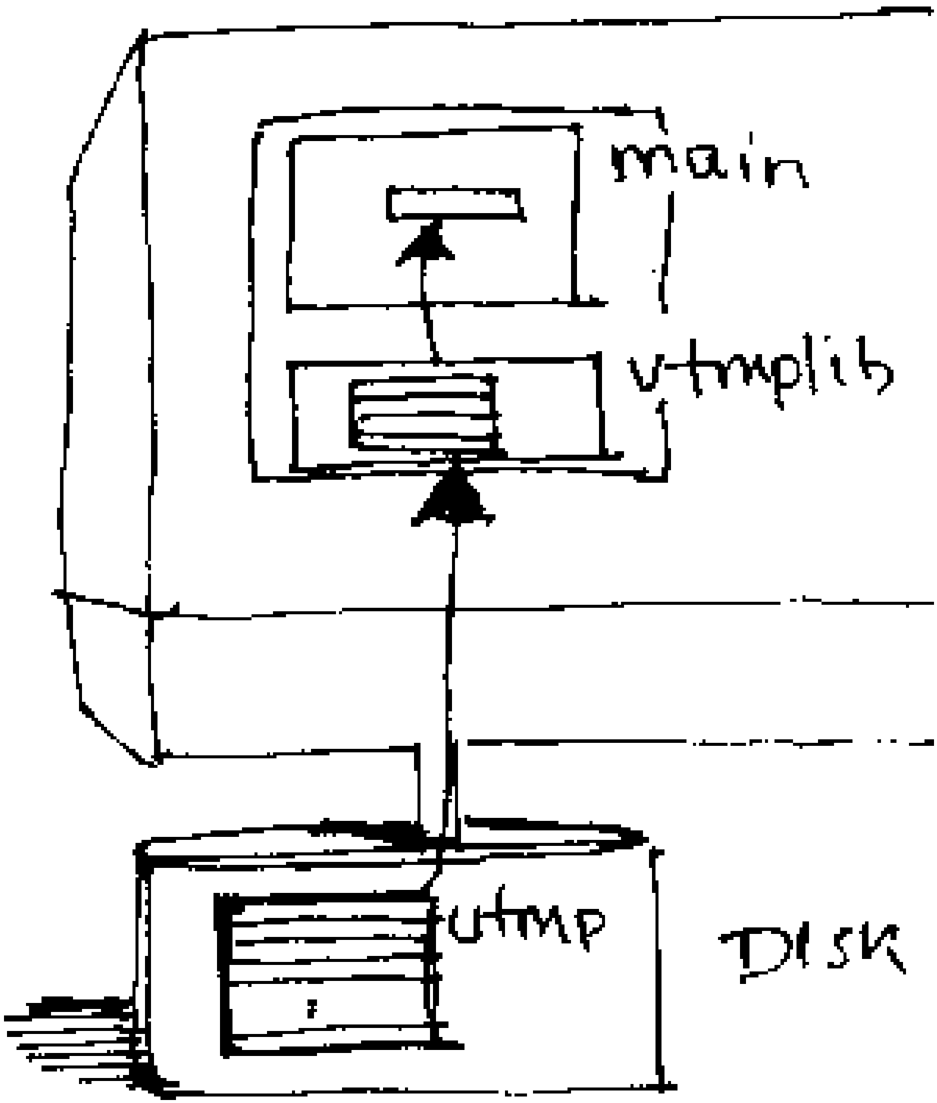
Yes! Making one system call for each line of output makes as much sense as buying pizza by the slice or eggs one at a time.

Better Idea: Read in a bunch of records at a time and then, as with eggs in a carton, take them one by one.



Adding Buffering to who2.c

Picture:



main calls functions in utmplib.c

utmplib.c keeps a utmp buffer with space for 16 utmp structs.

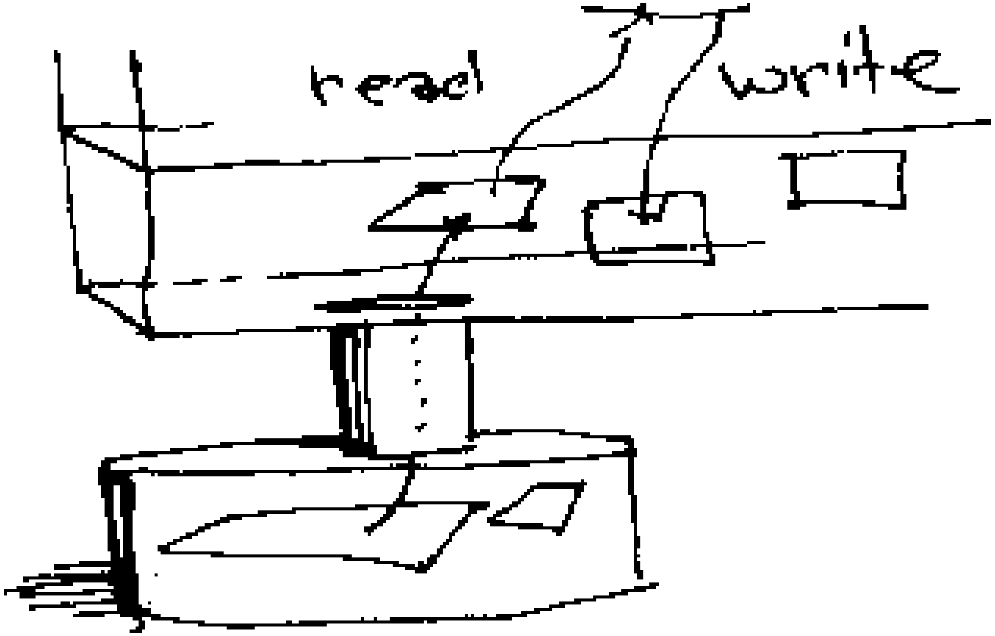
utmplib.c reads only when the buffer empties.

Result:
who3.c

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

It does!

The kernel keeps copies of disk blocks in memory. It writes those blocks to disk now and then. The read() call actually copies data from kernel buffers, not from the disk.



Consequences of Buffering

1) Faster "disk" I/O
2) Optimal disk reads and writes
3) Need to sync disks before shutdown

IX. Final Example: tail

1) What does tail do?

2) How does tail work?

...

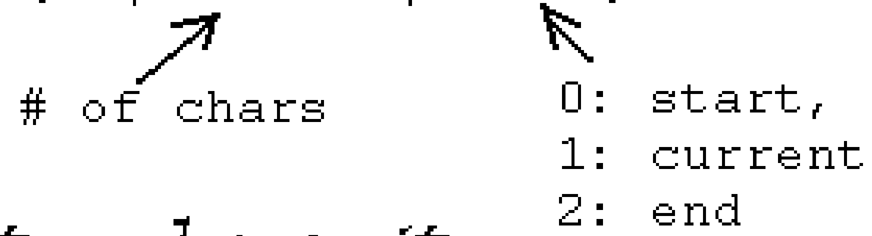
IX. Final Example: tail

1) What does tail do?

2) How does tail work?

Q: How can a program jump to the end (or to any arbitrary location) of a file?

A: `lseek(fd, offset, base)`



```
# of chars          0: start,
                    1: current
                    2: end
```

The next read or write occurs there.