Unix == Files

Benjamin Brewster

Adapted from slides by Jon Herlocker, OSU

Unix Paradigm

- Everything is a file
  - Even directories
  - Even hardware devices
  - Everything… except for processes

What is a File in UNIX?

- System Programmer View
  - Stream of bytes
    - An array
    - Newlines/carriage returns & tabs are all just bytes
  - Persistent
  - Usually stored on magnetic disk

Accessing a File in UNIX

```
# Open the file
file_descriptor = open(…)

# Call read(...) or write(...) any
# number of times

# Close the file
close(file_descriptor)
```

We're talking about C programming now, not shell scripts

Opening a File

- File can be open for
  - read only - O_RDONLY
  - write only - O_WRONLY
  - read and write - O_RDWR
- When you open a file for writing…
  - Should you delete an existing file with the same name?
  - If not, where do you want to start writing
    - Beginning? End? Somewhere else?
  - If the file doesn't exist, should you create it?
  - If you create it, what should the initial access permissions be?

Open for Read

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
int main(void)
{
  char file[] = "cs344/grades.txt";
  int file_descriptor;
  file_descriptor = open(file, O_RDONLY);
  if (file_descriptor < 0)
  {
    fprintf(stderr, "Could not open %s\n", file);
    exit(1);
  }
  exit(0);
}
```

Man pages again?

- In some cases there are two different man pages with the same name
  - Example :
    - “man read” will bring up the Bash manual page
    - “man -S 2 read” will bring up the read() system call page
- Make sure that you are looking at the right man page
  - Will say “Linux” at the bottom or top of *system call* pages (used in C). If it says Bash, it’s for the *shell* itself!
  - Man pages for *user programs* may say “GNU” or “FSF”
  - They **shouldn’t** say SunOS, Solaris, or Tcl!

Can’t find the right man page?

- Use “man -k”, which does a keyword search of all man page titles
  - Example: “man -k pthread” returns all man pages with “pthread” in their title
  - “man -k split” shows you man pages related to “split”
- The results will show you which section of the manual a man page is found:

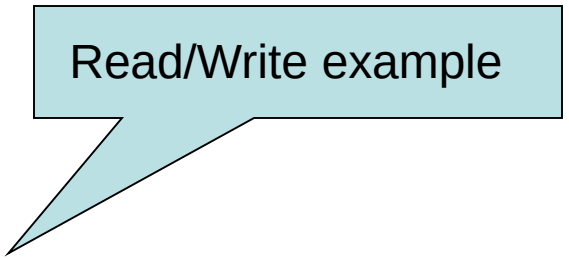
```
split          (1)  - split a file into pieces
split          (n)  - Split a string into a proper Tcl list
splitdiff      (1)  - separate out incremental patches
tiffsplit      (1)  - split a multi-image TIFF into single-image TIFF files
…
```

- You can then specify the right man page section
  - Example: “man -S n split”

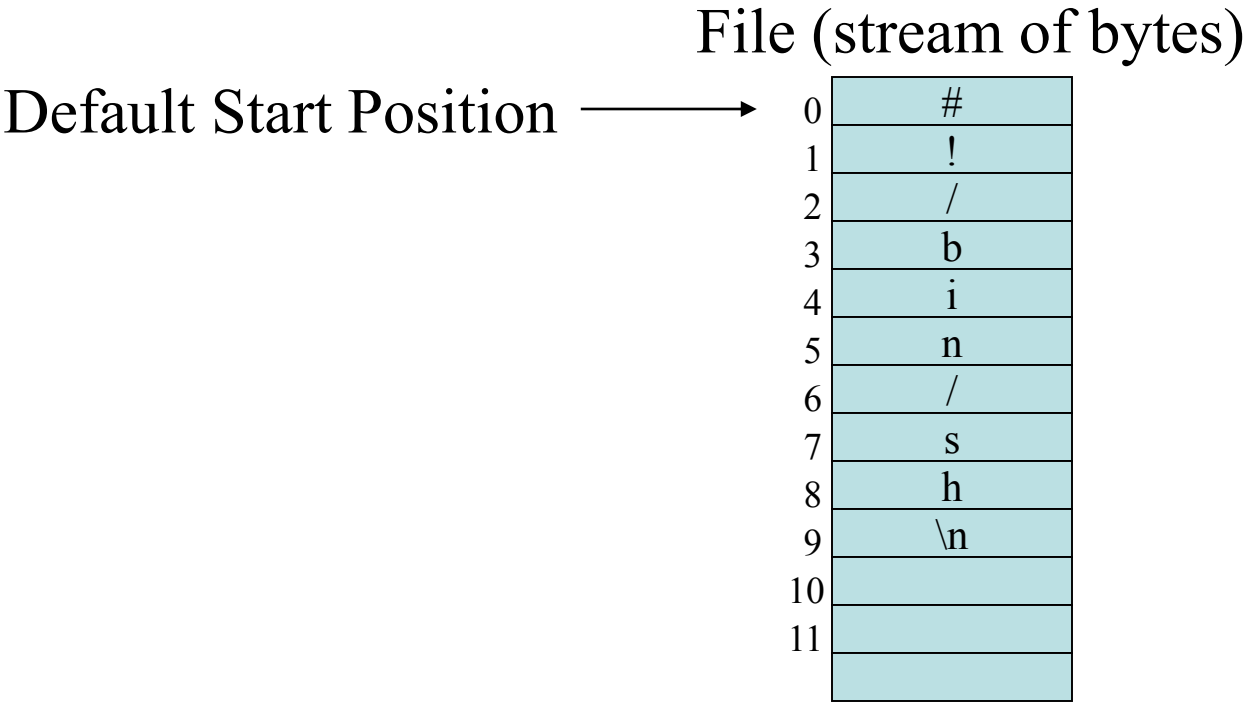
Open for Write

```
int main(void)
{
  char file[] = "cs344/grades.txt";
  int file_descriptor;
  file_descriptor = open(file, O_WRONLY);
  if (file_descriptor < 0)
  {
    fprintf(stderr, "Could not open %s\n", file);
    perror("in main");
    exit(1);
  }
  exit(0);
}
```

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int main(void)
{
   int file_descriptor;
   char *file = "test.txt";
   ssize_t nread;
   ssize_t nwritten;
   char buffer[512];
   file_descriptor = open(file, O_RDWR);
   if (file_descriptor == -1)
      exit(1);
   nread = read(file_descriptor, buffer, 512);
   nwritten = write(file_descriptor, buffer, nread);
   exit(0);
}
```



The file pointer



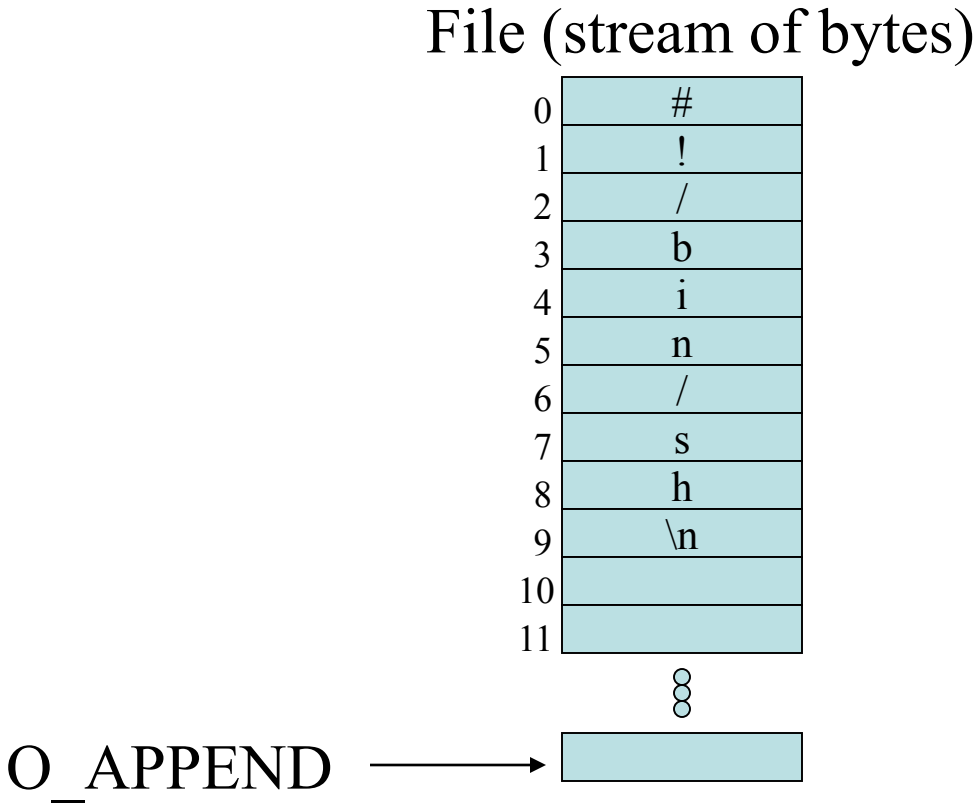
Truncating an Existing File

- When you open a file for writing
  - Should you delete an existing file with the same name?
    - To delete it: O_TRUNC
  - Example:
    - `file_descriptor = open(file, O_WRONLY | O_TRUNC);`
    - Deletes all data in existing file
    - Sets write pointer to position 0

Appending to an Existing File

- O_APPEND
- Example
  - `file_descriptor = open(file, O_WRONLY | O_APPEND);`
- Before every write, the file pointer will be reset to the end of the file

The file pointer



Creating a New File

- O_CREAT
- Flag indicates that the file should be created if it doesn't already exist
- Example
  - `file_descriptor = open(file, O_WRONLY | O_CREAT);`

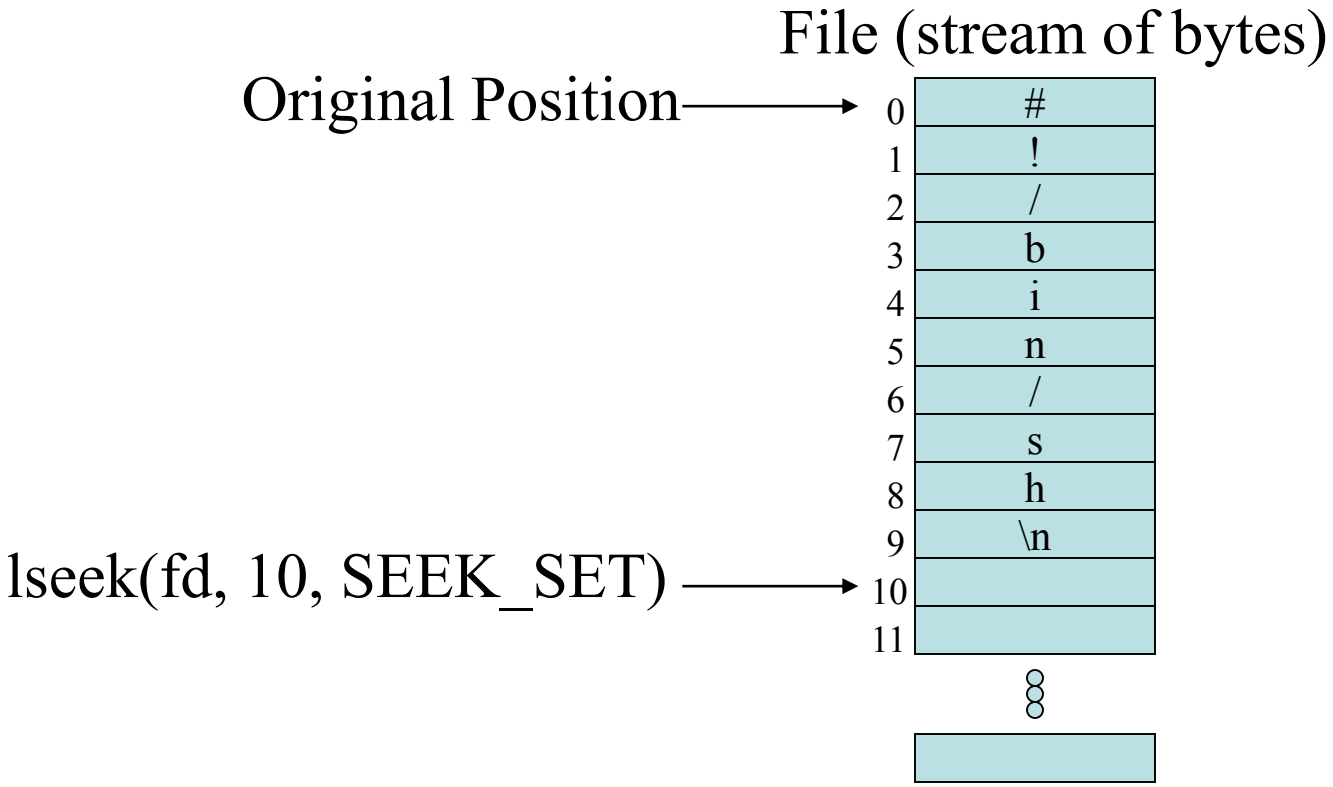
Access Permissions

- Third parameter only applies when new file is created (i.e. using O_CREAT)
- Third parameter is octal number
  - bits of the octal number signify the permissions
- If you don't specify it, there are reasonable defaults
- Example:
  - `file_descriptor = open(file, O_WRONLY|O_CREAT, 0644);`
- See "man chmod"

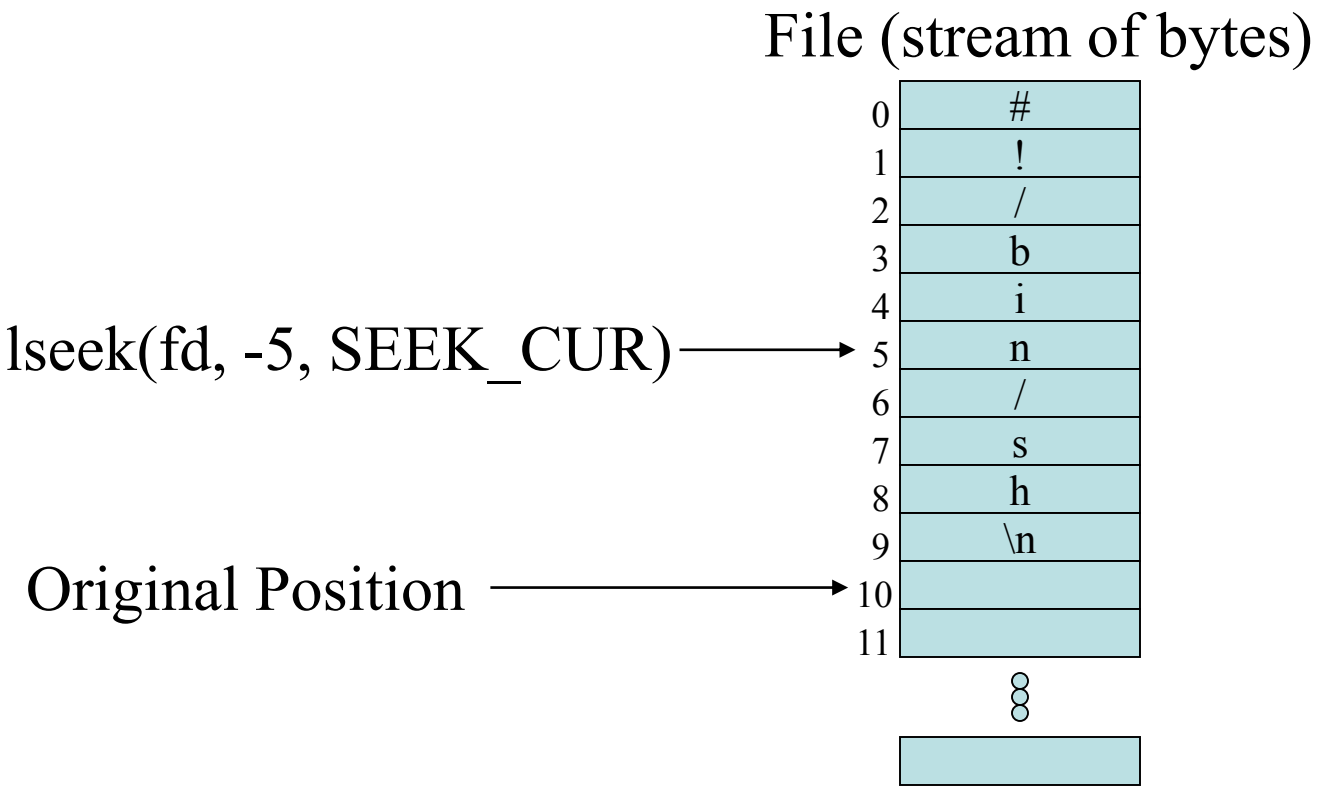
lseek

- Random access
  - Jump to any byte in a file
- Move to byte #16
  - `newpos = lseek(file_descriptor, 16, SEEK_SET);`
- Move forward 4 bytes
  - `newpos = lseek(file_descriptor, 4, SEEK_CUR);`
- Move to 8 bytes from the end
  - `newpos = lseek(file_descriptor, -8, SEEK_END);`

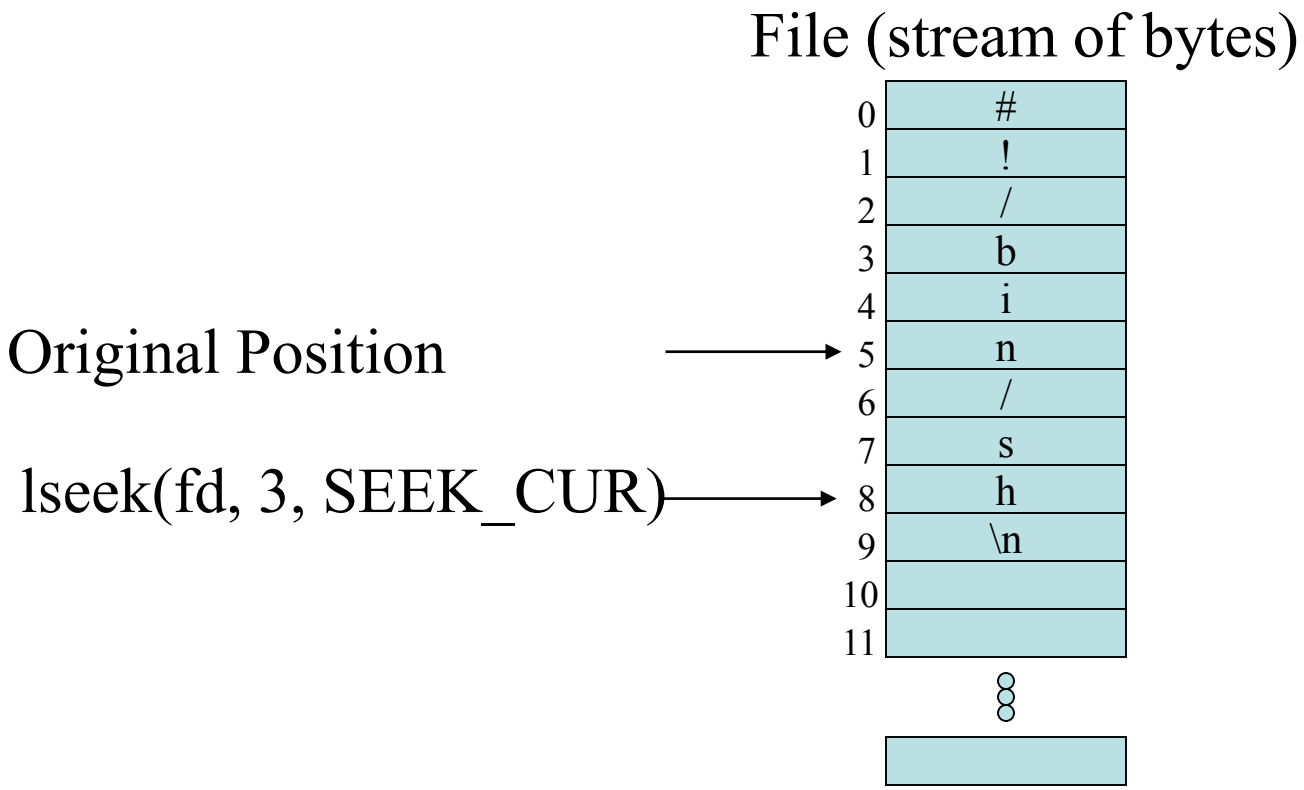
lseek - SEEK_SET



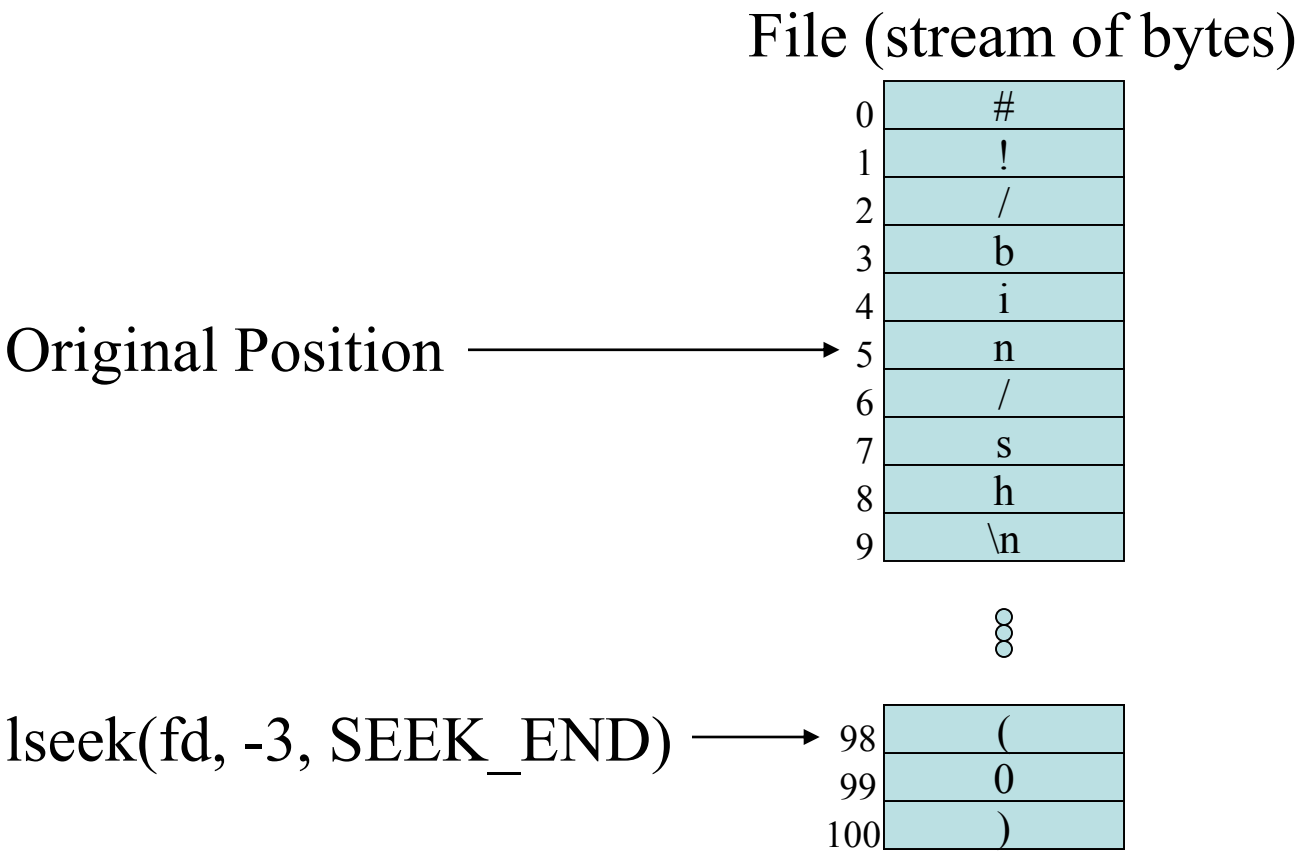
lseek - SEEK_CUR



lseek - SEEK_CUR



lseek - SEEK_CUR

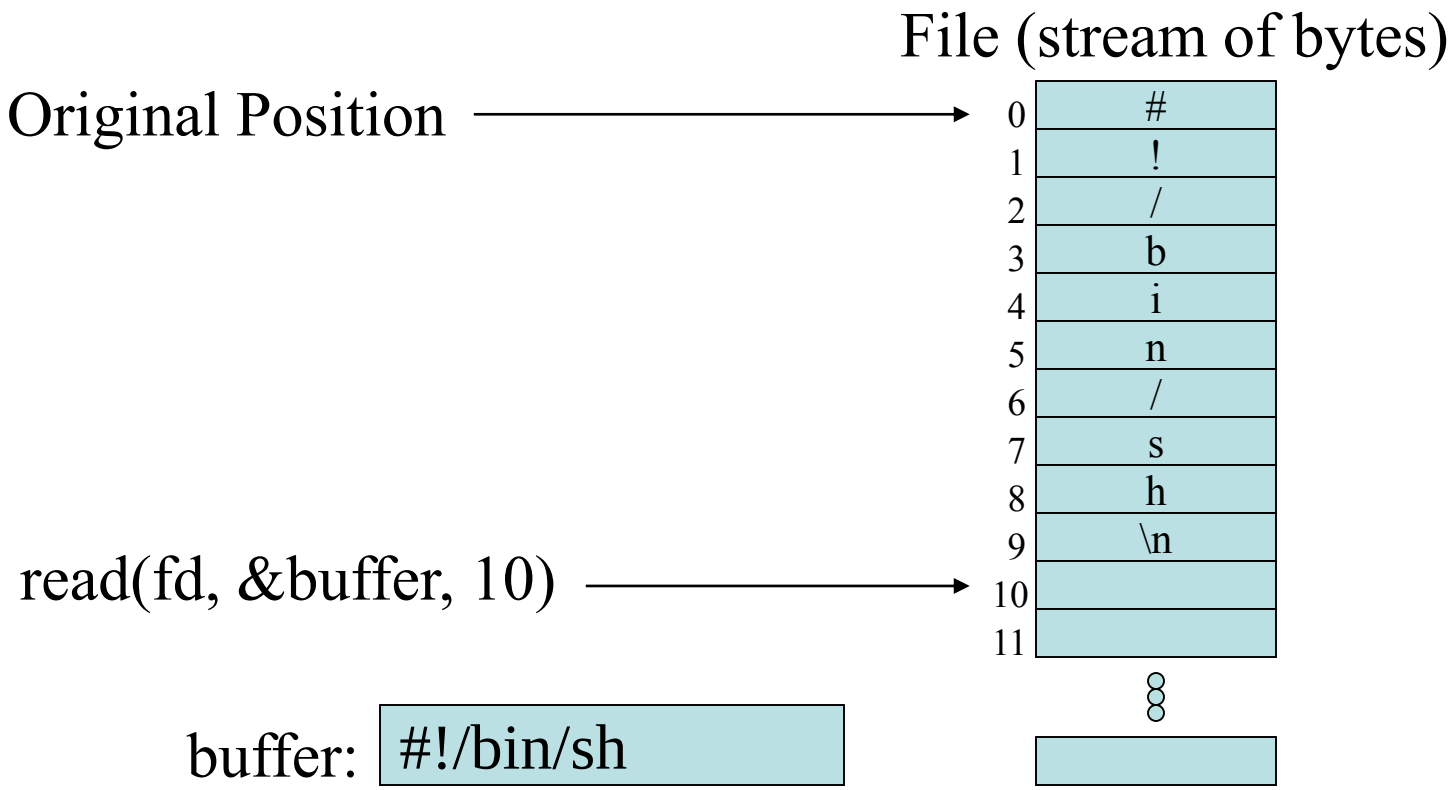


Read/Write and the File Pointer

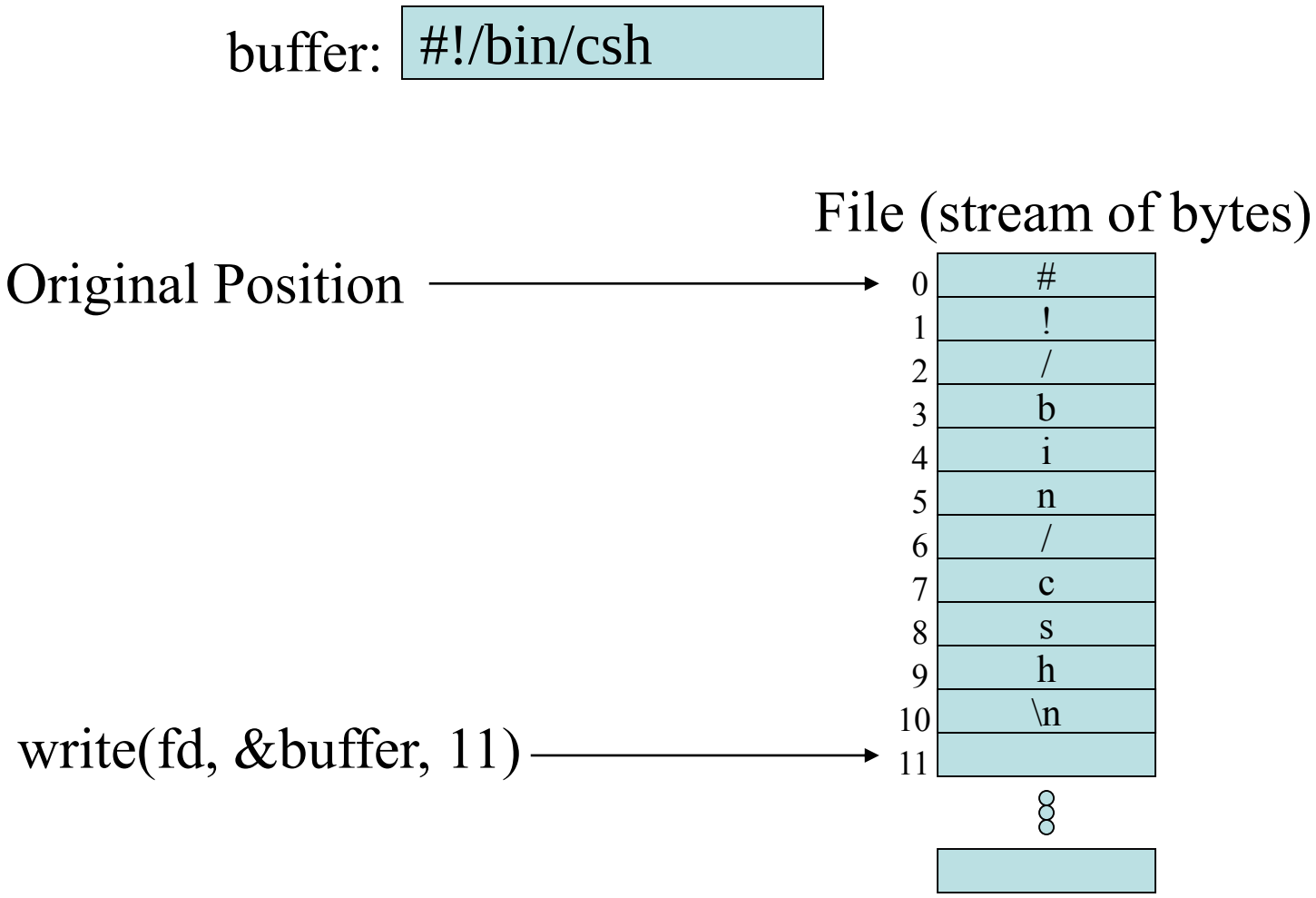
- Both read() and write() will change the file pointer.

- The pointer will be incremented by exactly the number of bytes read or written.

read


File (stream of bytes)
Original Position
0
1
2
3
4
5
6
7
8
9
10
11
#
!
/
b
i
n
/
s
h
\n
read(fd, &buffer, 10)
buffer:
#!/bin/sh

write


buffer:
#!/bin/csh
File (stream of bytes)
Original Position
write(fd, &buffer, 11)
0
1
2
3
4
5
6
7
8
9
10
11
#
!
/
b
i
n
/
c
s
h
\n

The Standard IO Library

- fopen, fclose, printf, fprintf, sprintf, scanf, fscanf, getc, putc, gets, fgets, etc.
- #include <stdio.h>

Why use stdio library

- Automatically buffers input and output intelligently
- Easy to work in line mode
  - ie read one line at a time
  - write one line at a time
- Powerful string and number formatting

Why use read()/write()

As opposed to stdio

- Maximal performance
  - IF you know exactly what you are doing
  - No additional hidden overhead from stdio
- Control exactly what is written/read at what times

Some stdio functions

- `fclose` Close a stream.
- `feof` Check if End Of File has been reached.
- `fgetc` Get next character from a stream.
- `fgetpos` Get position in a stream.
- `fopen` Open a file.
- `fprintf` Print formatted data to a stream.
- `fputc` Write character to a stream.
- `fread` Read block of data from a stream.
- `fseek` Reposition stream's position indicator.
- `getc` Get the next character.
- `getchar` Get the next character from stdin.

Some more stdio functions

- `gets` Get a string from stdin.
- `printf` Print formatted data to stdout.
- `putc` Write character to a stream.
- `putw` Write an integer to a stream.
- `remove` Delete a file.
- `rename` Rename a file or directory.
- `rewind` Reposition file pointer to the beg. of a stream.
- `scanf` Read formatted data from stdin.
- `sprintf` Format data to a string.
- `sscanf` Read formatted data from a string.
- `ungetc` Push a character back into stream.

Obtaining File Information

- stat() and fstat()
- Retrieve all sorts of information about a file
  - Which device it is stored on
  - Ownership/Permissions of that file
  - Number of links
  - Size of the file
  - Date/Time of last modification and access
  - Ideal block size for I/O to this file

```
struct stat statbuf;
int r;

r = stat(argv[1], &statbuf);
if (r == -1)
{
  fprintf(stderr, "Could not stat %s\n", argv[1]);
  perror(argv[0]);
  exit(1);
}

/* print out the size of the file */
printf("The logical size of %s is %ld bytes\n",
  argv[1], statbuf.st_size);
printf("The physical size of %s is %ld bytes\n",
  argv[1], statbuf.st_blocks * 512);
```

```
struct stat statbuf;
int r;
int fd;

fd = open(argv[1], O_RDONLY);
if (fd == -1) { /* handle error */  exit(1); }

r = fstat(fd, &statbuf);
if (r == -1)
{
  fprintf(stderr, "Could not fstat %s\n", argv[1]);
  perror(argv[0]);
  exit(1);
}

printf("The logical size of %s is %ld bytes\n", argv[1],
  statbuf.st_size);
printf("The phys    size of %s is %ld bytes\n", argv[1],
  statbuf.st_blocks * 512);
```

End