COMP1521

TUTORIAL 8

FILES > FILE OPERATIONS

NOTE:

- Assignment 2 was just released!
- Try to start early as it takes a bit longer than the first assignment!

FILES

What is a file in UNIX systems

Memory in Unix systems is just contiguous data; our filesystem is an abstraction to sort this data into things like:

- Regular files
  - ascii text
  - C programs
  - Executables
- Directories
- A network connection (sockets)
- Peripherals (stream of input data from say a keyboard)
- Symbolic links (the data stream from another file)

- Basically, in Unix everything is a file

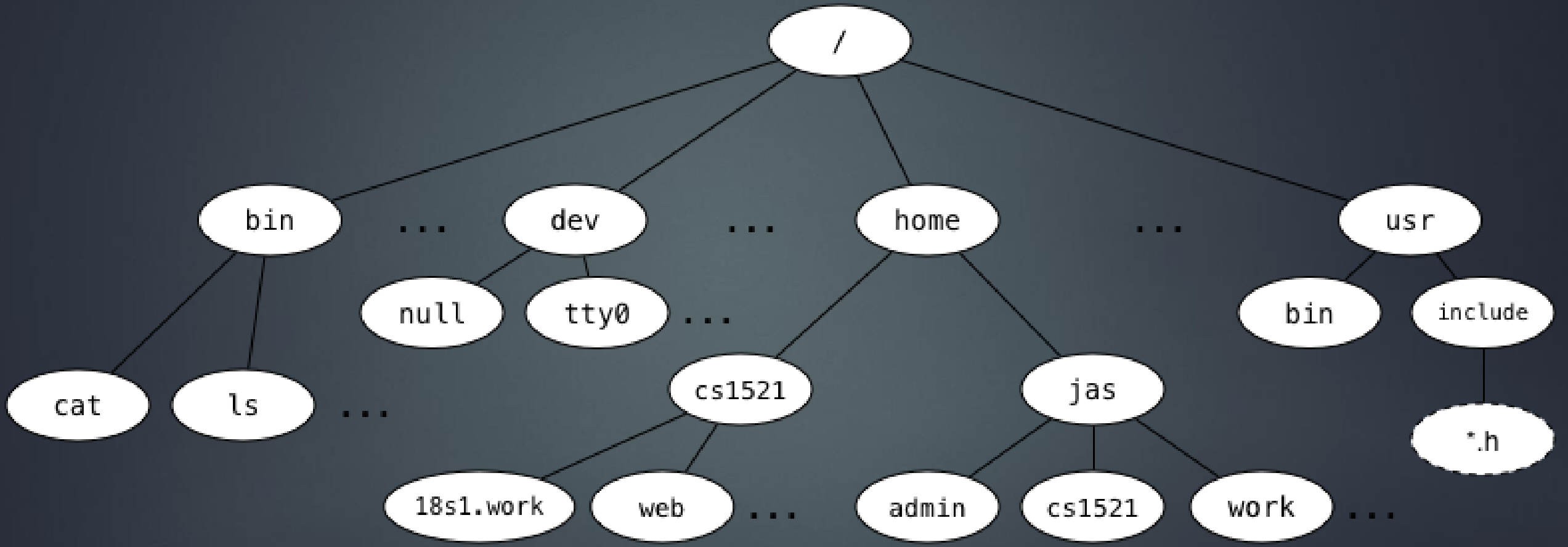
What is a file in UNIX systems

We also have special files in Unix that you may have heard of, but never realised they were files:

- stdin (input to the console)
- stdout (output to the console)
- stderr (output to the console from error stream)

These files typically are connected to the console, and we can interact with them in the same ways we interact with any other file

What does our file system look like?



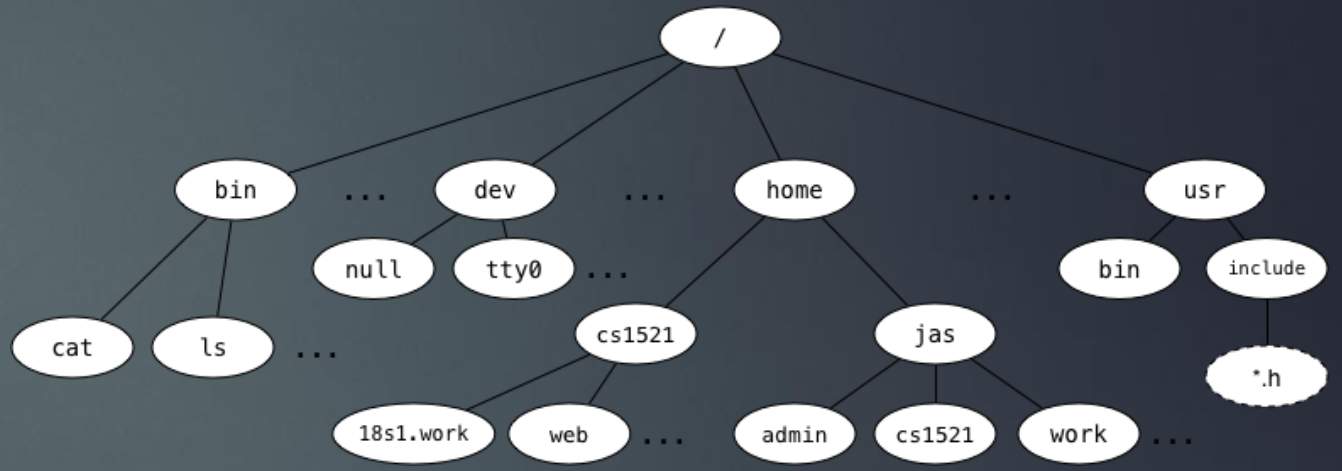
Path names

Files and directories are accessed via pathnames; we have two types of pathnames:

- Absolute Pathnames (start with / and give the full path from root directory)
  - /home/z1234567/lab08/main.c
  - Every process has a current working directory (CWD) which is its absolute path name
- Relative Pathnames (do not start with / and are appended onto the CWD of the process using them)
  - eg ./a.out
  - if exists within the absolute path previous example we have the absolute pathname as;
    - /home/z1234567/lab08/a.out

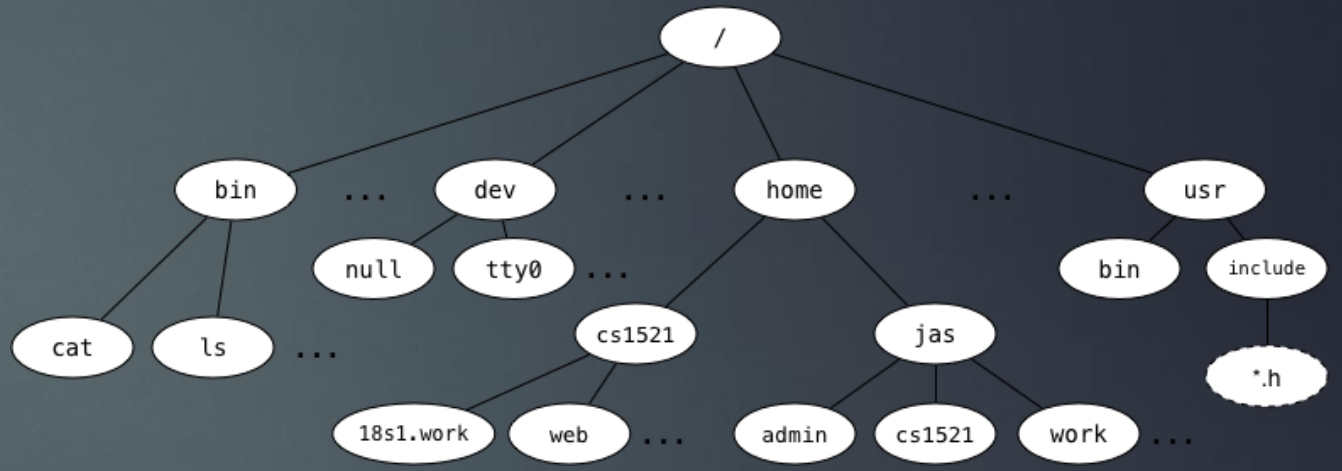
Example: q1

1. What is the full pathname of COMP1521's web directory?
2. Which directory is ~jas/../..?
3. Links to the children of a given directory are stored as entries in the directory structure. Where is the link to the parent directory stored?
4. What kind of file system object is cat?
5. What kind of file system object is home?
6. What kind of file system object is tty0?
7. What kind of file system object is a symbolic link? What value does it contain?
8. Symbolic links change the filesystem from a tree structure to a graph structure. How do they do this?



Example: q1

1. COMP1521's web directory is /home/cs1521/web
2. ~jas = /home/jas, so ~/jas/../.. is the root directory (/)
3. The link to a parent directory is stored as an entry in the structure (..)
4. cat is a regular file, that happens to be an executable
5. home is a directory
6. tty0 is a character special file, a file that represents a device which can read and write a byte-stream, typically interpreted as characters
7. A symbolic link is a special kind of file, that simply contains the name of another file
8. Symbolic links produce a graph because they allow arbitrary links between filesystem objects, without symlinks, the only connections are between parent/children, producing a tree



FILES OPERATIONS

System Calls

File systems use system calls executed by the operating system to perform operations, some of the common file operations we'll use are:

- 0 – read – read some bytes from a file descriptor
- 1 – write – write some bytes to a file descriptor
- 2 – open – open a file system object – returns a file descriptor
- 3 – close – stop using a file descriptor
- 4 – stat – get file system metadata for a pathname
- 8 – lseek – move a file descriptor to a specified offset within a file

File Operations in C

The typical file system calls, however, not an exhaustive list that you may use in this course

```c
FILE *fopen(const char *filename, const char *mode);  // Opens a file, returns a FILE pointer.
int fclose(FILE *stream);                             // Closes a file stream, flushing any buffered data.

int fgetc(FILE *stream);                              // Reads one character (byte) from a file.
int fputc(int c, FILE *stream);                       // Writes one character (byte) to a file.

char *fgets(char *s, int size, FILE *stream);         // Reads a line of text into a string buffer.
int fputs(const char *s, FILE *stream);               // Writes a string to a file.

int fprintf(FILE *stream, const char *format, ...);   // Writes formatted text to a file.
int fscanf(FILE *stream, const char *format, ...);    // Reads formatted text from a file.

size_t fread(void *ptr, size_t size, size_t n, FILE *stream);        // Reads a block of binary data.
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream); // Writes a block of binary data.

int stat(const char *pathname, struct stat *statbuf); // Gets a file's metadata (size, permissions, etc.)
int fseek(FILE *stream, long int offset, int whence); // Moves the read/write position within a file.
```

Opening Files

```
FILE *fopen(const char *pathname, const char *mode);
```

Returns a file pointer for the path provided

Path name of the file we wish to open

The mode of the file (read/write/append etc)

The *fopen* modes translate into flags that are used by open()

- fopen(FilePath, "w") → open(FilePath, O_WRONLY | OCREAT | OTRUNC)

We only use the f version of the open function in this course, and is applicable for fread/fwrite functions as well

fopen() ✓

open() ✗

Opening Files (Modes)

Mode	Description	Mode	Description
"w"	Open or create a text file in write mode, existing data is erased	"wb"	Open or create a binary file in write mode
"r"	Open an existing text file in read mode, file must exist	"rb"	Open an existing binary file in read mode
"a"	Open or create a text file in append mode, existing data is not erased like "w"	"ab"	Open or create a binary file in append mode
"w+"	Open or create text file in both read and write mode, existing data is erased	"wb+"	Open or create a binary file in both read and write mode, existing data is erased
"r+"	Open an existing text file in read and write mode, initial position is the start	"rb+"	Open an existing binary file in read and write mode, initial position is the start
"a+"	Open or create a text file in read or write mode, initial position is the end	"ab+"	Open or create a binary file in read or write mode, initial position is the end

What if fopen returns NULL

Firstly, what are the circumstances that fopen returns NULL?

- If you tried to open a file for reading that does not exist
- If you try to open a file without sufficient permissions
- If the "mode" was invalid
- If the system is out of memory
- If the pathname is too long and many more reasons..

How do we print the specific reason fopen returns NULL:

- We can use a variable called errno, which is set when the system call fails, it is an error value that is mapped to a string of the reason for error
  - Check man errno for the list of defined errors
- We can print this using the perror() function, and a descriptive string can be passed as an argument

```c
char *filename = "/text.txt"
FILE *f = fopen(filename, "r");

if (f == NULL) {
  perror("fopen")
  // if you wanted a more specific message
  // or for reporting other errors (read, write etc)
  // fprintf(stderr, "Error opening %s: %s", filename, strerror(errno))
}
```

Writing/Reading Files

The easiest way to write to a file, is using functions writing or reading one byte at a time:

- int fgetc(FILE *stream)
  - fgetc() has 257 possible return values, numbers 0-255, as well as -1 for an EOF value
- int fputc(int c, FILE *stream)

Similarly, we can read/write one byte at a time from a buffer using, though without proper care can have portability issues (more on this later):

- size_t fread(void *ptr, size_t size, size_t n, FILE *stream)
- size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream)

Additionally, we can read/write strings with the following operations, but they have many scenarios they cannot be used (more on this later)

- char *fgets(char *s, int size, FILE *stream)
- int fputs(const char *s, FILE *stream)
- int fscanf(FILE *stream, const char *format, ...)
- int fprintf(FILE *stream, const char *format, ...)

Text Files vs Binary Files

File system operations technically are just inputs and outputs of bytes, why can't we use string functions to read/write binary data?

- char *fgets(char *str, int n, FILE *stream);
- int fputs(const char *str, FILE *stream);
- int fscanf(FILE *stream, const char *format, ...);
- int fprintf(FILE *stream, const char *format, ...);

We cannot use this on binary data, as they may or may not contain zero bytes (null terminator)

They can be used on ascii or unicode text, but not to read file types such as jpg

Portability

(relevant for assignment)

Writing multi-byte data types (int, uint32_t) with fwrite is not portable. It copies raw bytes and byte order is machine dependent.

This byte order is called endianness consider a number 0x001E663A:

- Big Endian: Stores the most significant byte first

Address	0x00	0x01	0x02	0x03
Value	00	1E	66	3A

- Little Endian: Stores the least significant byte first

Address	0x00	0x01	0x02	0x03
Value	3A	66	1E	00

- For the assignment:
  - To write portable files only use fwrite with elements of size 1 (char, uint8_t)
  - To do this for larger number types (int, uint32_t), you must write it out byte-by-byte in a defined consistent order
    - Consider making a helper function to perform this

Example: q5 (first_line.c)

Write a C program, first_line.c, which is given one command-line argument, the name of the file, of which prints the first line of that file to stdout. If given an incorrect number of arguments, or if there was an error opening the file, it should print a suitable error message.

Example: q6 (write_line.c)

Write a C program, write_line.c, which is given one command-line argument, the name of the file, which reads a line from stdin, and writes it to the specified file; if the file exists, it should be overwritten.

Example: q7 (append_line.c)

Write a C program, append_line.c, which is given one command-line argument, the name of the file, which reads a line from stdin, and appends it to the specified file

How do we move around a file?

We can use a function called fseek() to move the position of our file descriptor within a file

```
int fseek(FILE *stream, long int offset, int whence);
```

Returns 0 on success, non-zero on failure

File pointer

Desired offset into the file (can be negative)

Where to base the offset from:
SEEK_SET = relative to start of file
SEEK_CUR = relative to current position
SEEK_END = relative to end-of-file

first_line_fseek.c for an example use of this function

How do we move around a file?

Additionally, we can use the function ftell(), while passing in our file pointer to get our current location in a file

```
long ftell(FILE *stream);
```

What is stat()?

The stat() function is a system call to retrieve metadata about a file, without needing to open or read its contents. Its useful for checking properties such as file size, permissions, and last modification time before proceeding with other operations.

```
int stat(const char *pathname, struct stat *statbuf);
```

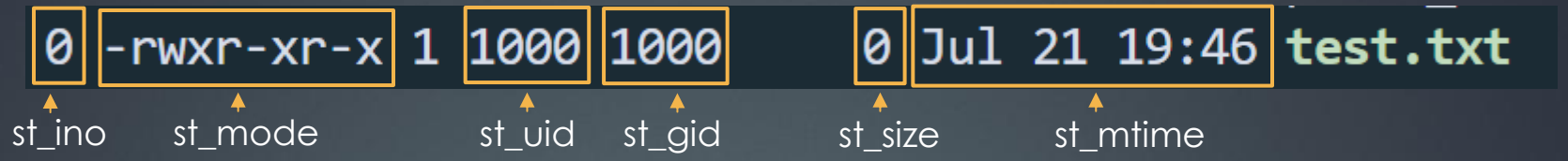
When using stat() we need to initialise a struct stat which we will pass a pointer into function call, if successful it will return 0, if unsuccessful the stat() will return -1

```
char *filename = "./hello.txt";
struct stat s;

if (stat(filename, &s) != 0) {
  perror("stat")
  return 1;
}

// proceed with using stat structs content
```

Fields of stat struct



Field	Use
ino_t st_ino	Inode number, index into filesystem metadata
mode_t st_mode*	File type and file permissions encoded as a bit string
uid_t st_uid	Numeric user id (uid) of whom the file belongs to
gid_t st_gid	Numeric group id (gid) of whom the file belongs to
off_t st_size*	Gives the total size in bytes of the file
blksize_st st_blksize	Gives the size of a block on the storage device
blkcnt_t st_blocks	Gives the amount of space allocated on the storage device
time_t st_atime	Last time the content was accessed (read or write)
time_t st_mtime*	Last time the file was modified
time_t st_ctime	Last time the file status was changed (file contents or metadata)

* = the fields you will most likely use in this course

File permissions

```
0 -rwxr-xr-x 1 1000 1000     0 Jul 21 19:46 test.txt
```

We have seen when we ls -l we have the dashes or d-r-w-x, these are the permissions for the file. We can access the bit string that represents these from the st_mode field within the stat struct

man 7 inode has a list of macros and bit masks, for checking file types or permissions

```c
// get the information about the permissions of this file
 mode_t mode = s.st_mode;

if (S_ISREG(mode)) {
    printf("File: %s is a regular file\n", argv[1]);
} else if (S_ISDIR(mode)) {
    printf("File: %s is a directory\n", argv[1]);
}

// can i write to this file as the owner?

if (mode & S_IWUSR) {
    printf("You can write to this file as the owner.\n");
} else {
    printf("You cannot write to this file as the owner.\n");
}
```

How to change file permissions

These same bit masks to check permissions from st_mode, we can use to update or set permissions of a file with a bitwise OR.

The function we use to do this chmod()

```
mode_t new_mode = S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH | S_IXOTH;
//                        Owner: rwx      Group: r-x        Others: r-x

if (chmod(argv[1], new_mode) != 0) {
    perror("chmod");
    return 1;
}
```

This example would be identical to doing chmod 755 file.txt in your terminal
Note: file permissions are in octal

FIN