Week 09 Tutorial Answers

1. How is the assignment going?

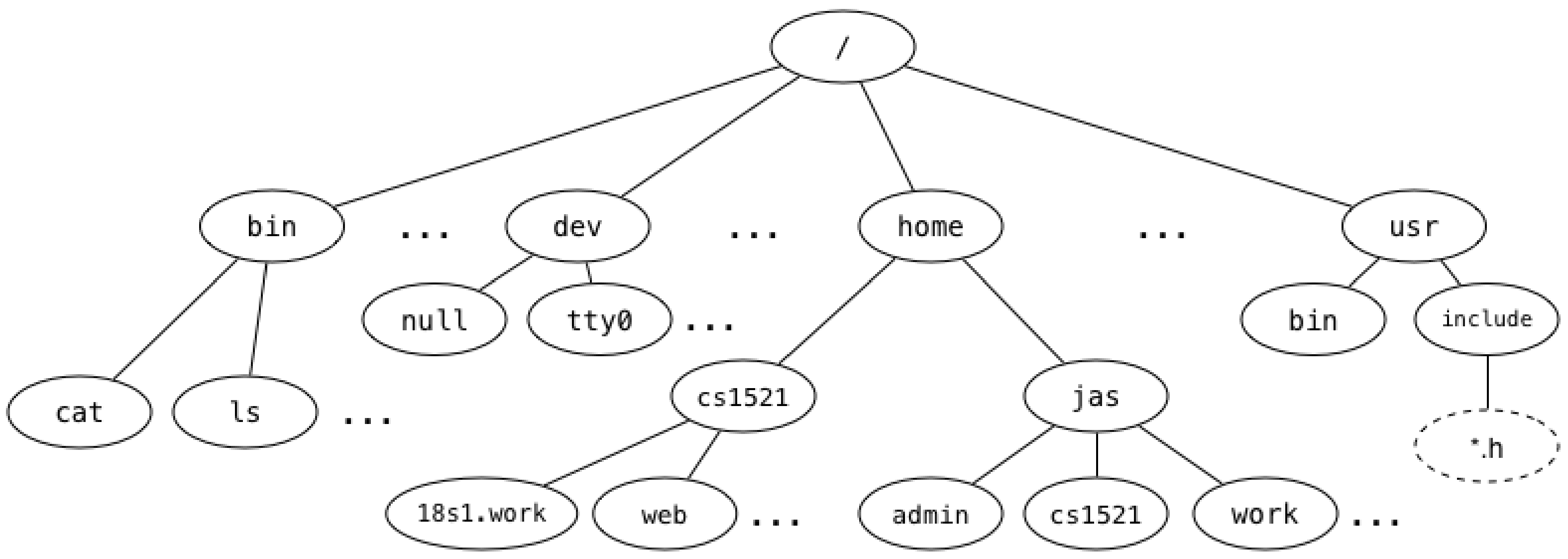
   Does anyone have hints or advice for other students?

   Has anyone discovered interesting cases that have to be handled?

   Answer:

   Discussed in tutorial.

2. We say that the Unix filesystem is *tree-structured*, with the directory called `/` as the root of the tree, e.g.,



   Answer the following based on the above diagram:

   a. What is the full pathname of COMP1521's `web` directory?

   b. Which directory is `~jas/../..`?

   c. 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?

   d. What kind of filesystem object is `cat`?

   e. What kind of filesystem object is `home`?

   f. What kind of filesystem object is `tty0`?

   g. What kind of filesystem object is a symbolic link? What value does it contain?

   h. Symbolic links change the filesystem from a tree structure to a graph structure. How do they do this?

   Answer:

   a. COMP1521's `web` directory is `/home/cs1521/web`

   b. `~jas` is shorthand for `/home/jas`, so `~jas/../..` is the root directory (`/`)

   c. The link to the parent directory is also stored as an entry in the directory structure, called `..`

   d. `cat` is a regular file, which happens to contain executable machine code.

   e. `home` is a directory.

   f. `tty0` is a *character special file*, a file that represents a *device* which can read and write a byte-stream, which is typically interpreted as characters.

   g. A symbolic link is a special kind of file that simply contains the name of another file.

   h. Symbolic links produce a graph because they allow arbitrary links between filesystem objects. Without symlinks, the only "connections" in the filesystem are parent/child links, which produce a tree.

3. The `stat()` and `lstat()` functions both take an argument which is a pointer to a `struct stat` object, and fill it with the meta-data for a named file.

   On Linux, a `struct stat` contains the following fields (among others, which have omitted for simplicity):

```
struct stat {
    ino_t st_ino;         /* inode number */
    mode_t st_mode;       /* protection */
    uid_t st_uid;         /* user ID of owner */
    gid_t st_gid;         /* group ID of owner */
    off_t st_size;        /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for filesystem I/O */
    blkcnt_t st_blocks;   /* number of 512B blocks allocated */
    time_t st_atime;      /* time of last access */
    time_t st_mtime;      /* time of last modification */
    time_t st_ctime;      /* time of last status change */
};
```

Explain what each of the fields represents (in more detail than given in the comment!) and give a typical value for a regular file which appears as follows:

```
$ ls -ls stat.c
8 -rw-r--r--  1 jas  cs1521  1855  Sep  9 14:24 stat.c
```

Assume that `jas` has user id 516, and the `cs1521` group has group id 36820.

Answer:

The `struct stat` fields are:

`ino_t st_ino`
An inode number, giving an index into the filesystem's table of file metadata structures. For `stat.c`, it could be any largish positive integer. The inode number can be accessed using `ls -li`.

`mode_t st_mode`
Contains information about the file type and the file permissions, encoded as a bit-string. These bit-strings are usually written in octal, to make it easy to see the 3 groups of 3 bits defining the file permissions. A regular file like `stat.c` would have an `st_mode` value of `0100644` (from `S_IFREG | S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH`).

`uid_t st_uid`
Gives the numeric user id (uid) of the user to whom the file belongs (its owner); in this case, 516. This can be retrieved using `ls -ln`.

`gid_t st_gid`
Gives the numeric id (gid) of the group to which the file belongs; in this case, 36820. This can be retrieved using `ls -ln`.

`off_t st_size`
Gives the total size of the file in bytes. For a text file like `stat.c`, it's simply the number of characters in the file's content (i.e., 1855).

`blksize_t st_blksize`
Gives the size of a block on the storage device useful for files of this type. Typical block size are 512, 1024, 4096, 8192.

`blkcnt_t st_blocks`
Gives the amount of space on the storage device allocated for this file. Since it's allocated in 512B chunks, more space might be allocated than is actually required to store the bytes. Often blocks are allocated in groups of size 2^n. The total bytes allocated in the blocks must, of course, be larger than `st_size`. For `stat.c`, there are 8 blocks allocated (a total of 4096 bytes, to store the 1855 actually in the file).

`time_t st_mtime`
Gives the last time the file was modified. A `time_t` value is typically implemented as an integer giving the number of seconds since midnight on Jan 1 1970. For the `stat.c` file, the most recent update time is shown in the `ls` output as `Sep 9 14:24`, which, here, is implied to be `2017/09/09 14:24`, a value around 1504931040.

`time_t st_atime`
The last time the file content was accessed (read or written). This value can be retrieved using `ls -lu`.

`time_t st_ctime`
The last time the file status was changed. This could mean changing the file contents, or changing its associated metadata. This value can be retrieved using `ls -lc`.

4. Consider the following (edited) output from the command `ls -l ~cs1521`:

```
drwxr-x--- 11 cs1521 cs1521 4096 Aug 27 11:59 17s2.work
drwxr-xr-x  2 cs1521 cs1521 4096 Aug 20 13:20 bin
-rw-r-----  1 cs1521 cs1521   38 Jul 20 14:28 give.spec
drwxr-xr-x  3 cs1521 cs1521 4096 Aug 20 13:20 lib
drwxr-x--x  3 cs1521 cs1521 4096 Jul 20 10:58 public_html
drwxr-xr-x 12 cs1521 cs1521 4096 Aug 13 17:31 spim
drwxr-x---  2 cs1521 cs1521 4096 Sep  4 15:18 tmp
lrwxrwxrwx  1 cs1521 cs1521   11 Jul 16 18:33 web -> public_html
```

a. Who can access the `17s2.work` directory?

b. What operations can a typical user perform on the `public_html` directory?

c. What is the file `web`?

d. What is the difference between `stat("web", &info)` and `lstat("web", &info)`?
(where `info` is an object of type `(struct stat)`)

Answer:

a. The user `cs1521`, and any member of the `cs1521` group can `cd` into the `17s2.work` directory and list the directory contents. The user `cs1521` can also create new entries in `17s2.work`.

b. Assume that "typical user" means someone who is not a member of the `cs1521` group. Such a user can `cd` into the directory, but they cannot list the contents of the directory. If they know the name of a file in the directory, they can also open that file (assuming they have read permission on the file).

c. The file `web` is a *symbolic link* (or *symlink*). It effectively makes `web` an alternative name for the `public_html` directory.

d. The function call `stat("web", &info)` follows the symlink, and places meta-data about the `public_html` directory in the `info` struct.

The function call `lstat("web", &info)` places meta-data about the symlink itself into the `info` struct.

5. Write a C program, chmod_if_public_write.c, which is given 1+ command-line arguments which are the pathnames of files or directories
If the file or directory is publically-writeable, it should change it to be not publically-writeable, leaving other permissions unchanged.

It also should print a line to stdout as in the example below

```
$ dcc chmod_if_public_write.c -o chmod_if_public_write
$ ls -ld file_modes.c file_modes file_sizes.c file_sizes
-rwxr-xrwx 1 z5555555 z5555555 116744 Nov  2 13:00 file_sizes
-rw-r--r-- 1 z5555555 z5555555    604 Nov  2 12:58 file_sizes.c
-rwxr-xr-x 1 z5555555 z5555555 222672 Nov  2 13:00 file_modes
-rw-r--rw- 1 z5555555 z5555555   2934 Nov  2 12:59 file_modes.c
$ ./file_modes file_modes file_modes.c file_sizes file_sizes.c
removing public write from file_sizes
file_sizes.c is not publically writable
file_modes is not publically writable
removing public write from file_modes.c
$ ls -ld file_modes.c file_modes file_sizes.c file_sizes
-rwxr-xr-x 1 z5555555 z5555555 116744 Nov  2 13:00 file_sizes
-rw-r--r-- 1 z5555555 z5555555    604 Nov  2 12:58 file_sizes.c
-rwxr-xr-x 1 z5555555 z5555555 222672 Nov  2 13:00 file_modes
-rw-r--r-- 1 z5555555 z5555555   2934 Nov  2 12:59 file_modes.c
```

Make sure you handle errors.

Answer:

```c
// remove public write from specified as command line arguments if needed

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

void chmod_if_needed(char *pathname);

int main(int argc, char *argv[]) {
    for (int arg = 1; arg < argc; arg++) {
        chmod_if_needed(argv[arg]);
    }
    return 0;
}

// chmod a file if publically-writeable

void chmod_if_needed(char *pathname) {
    struct stat s;
    if (stat(pathname, &s) != 0) {
        perror(pathname);
        exit(1);
    }

    mode_t mode = s.st_mode;

    if (!(mode &  S_IWOTH)) {
        printf("%s is not publically writable\n", pathname);
        return;
    }

    printf("removing public write from %s\n", pathname);

    mode_t new_mode = mode & ~S_IWOTH;

    if (chmod(pathname, new_mode) != 0) {
        perror(pathname);
        exit(1);
    }
}
```

6. Write a C program, fgrep.c, which is given 1+ command-line arguments which is a string to search for.
   If there is only 1 command-line argument it should read lines from stdin and print them to stdout iff they contain the string specified as the first command line argumenbt.

   If there are 2 or more command line arguments, it should treat arguments after the first as fiilenames and print any lines they contain which contain the string specified as the first command line arguments.

   When printing lines your program should prefix them with a line number.

   It should print suitable error messages if given an incorrect number of arguments or if there is an error opening a file.

Answer:

```c
// Print lines containing  specified substring from the files specified as arguments

#include <stdio.h>
#include <string.h>

#define MAX_LINE 65536

void search_stream(FILE *stream, char stream_name[], char search_for[]);

int main(int argc, char *argv[]) {

    if (argc < 2) {
        fprintf(stderr, "Usage: %s <prefix> <files>\n", argv[0]);
        return 1;
    } if (argc == 2) {
        search_stream(stdin, "<stdin>", argv[1]);
    } else {

        for(int argument = 2; argument < argc; argument = argument + 1) {
            FILE *in = fopen(argv[argument], "r");
            if (in == NULL) {
                fprintf(stderr, "%s: ", argv[0]);
                perror(argv[argument]);
                return 1;
            }

            search_stream(in, argv[argument], argv[1]);
        }

    }

    return 0;
}

void search_stream(FILE *stream, char stream_name[], char search_for[]) {
    char line[MAX_LINE];

    int line_number = 1;
    while (fgets(line, MAX_LINE, stream) != NULL) {
        if (strstr(line, search_for) != NULL) {
            printf("%s:%d:%s", stream_name, line_number, line);
        }
        line_number = line_number + 1;
    }
}
```