COMP1521

WEEK 9

Announcements

- Lab start this week!
  - Lab9 has been released, due on **Week 10 Monday 12:00:00 (midday)**

- weekly quiz 8 be **due Week 9 Thursday 21:00:00**
- **Assignment2** will due on **Week 10 Friday 18:00:00**

- **Final Exam (25/08/2025)**
- **There will be 2 sessions(Morning/afternoon), only need to attend one of these sessions.**
  - **Details……**
  - **Week10's lab!!!!!**

Contents

Common Environment Variables

Variable	Description
PATH	Specifies directories to search for executable programs (e.g., ls, gcc).
HOME	The current user's home directory path (e.g., /home/yourname).
PWD	The present working directory (used when running pwd).
USER	The name of the current user.
SHELL	The path to the current shell program (e.g., /bin/bash).
LANG	Default language and locale setting (e.g., en_US.UTF-8).
TERM	Specifies the terminal type (e.g., xterm-256color).

getenv

- char *getenv(const char *name);

- - Retrieves the value of an environment variable.
- - Returns the value as a string, or NULL if not found.

- Example:
- char *path = getenv("PATH");

opendir

- DIR *opendir(const char *name);

- - Opens a directory stream.
- - Returns a DIR* on success, NULL on failure.

- Example:
- DIR *dir = opendir("./");

readdir

- struct dirent *readdir(DIR *dirp);

- - Reads the next directory entry.
- - Returns struct dirent* or NULL when done.

- Example:
- while ((entry = readdir(dir)) != NULL) {
- printf("%s\n", entry->d_name);
- }

closedir

- int closedir(DIR *dirp);

- - Closes an open directory stream.
- - Returns 0 on success, -1 on failure.

write recurse-sample.c:

a file that recurses through a directory and prints out permissions and filenames of all files with "hello" in it's name.

getenv() — get an environment variable

```c
#include <stdlib.h>

char *getenv(const char *name);
```

- search environment variable array for **name=value**
- returns **value**
- returns **NULL** if **name** not in environment variable array

```c
int main(void) {
    // print value of environment variable STATUS
    char *value = getenv("STATUS");
    printf("Environment variable 'STATUS' has value '%s'\n", value);
```

source code for get_status.c

write recurse-sample.c:
a file that recurses through a directory and prints out permissions and filenames of all files with "hello" in it's name.

// part 0: get and print out current directory pathname.

write recurse-sample.c:
a file that recurses through a directory and prints out permissions and filenames of all files with "hello" in it's name.

// part 0: get and print out current directory pathname.

// part 1: print out all files in current directory.

Searching Directories - Readdir

```c
    // opens DIR pointer
    DIR *dirp = opendir(current_path);
    if (dirp == NULL) {
        perror(current_path);
        exit(1);
    }

    struct dirent *de;
    //loops through all files in DIR, populating struct direct de.
    while ((de = readdir(dirp)) != NULL) {
        printf("%s\n", de->d_name);
    }
```

write recurse-sample.c:
a file that recurses through a directory and prints out permissions and filenames of all files with "hello" in it's name.

// part 0: get and print out current directory pathname.

// part 1: print out all files in current directory.

// part 2: recurse through all subdirectories and print out pathnames.

Recursing through Directories -

- We want to Search through each subdirectory using readdir().

Steps:

- Search through current directory using readdir()
- Skip current (.) and parent directories (..)
- If readdir() returns a directory, recursively search through it.
  - use stat.h to find S_IFDIR.
  - Append filename to old pathname to create next directory's pathname.

```c
while ((de = readdir(dirp)) != NULL) {
    printf("%s\n", de->d_name);

    //check if is self or parent
    char *self = ".";
    char *parent = "..";
    if (!strcmp(de->d_name, self) || !strcmp(de->d_name, parent)) {
        continue;
    }
    //if directory, recurse into it.
    int new_length = strlen(de->d_name) + strlen(current_path) + 2;
    char *new_path = malloc(new_length * sizeof(char));
    sprintf(new_path, "%s/%s", current_path, de->d_name);
    struct stat s;
    // use lstat
    lstat(new_path, &s);
    mode_t perms = s.st_mode;
    if((perms & S_IFMT) == S_IFDIR) {
        recurse(new_path);
    }
```

Stat.h

quick reference guide :

[stat.h quick reference]

man 2 stat = stat(pathname, &s) reference
man 7 inode = permissions reference

Usage:

```
struct stat s;
if (stat(pathname, &s) != 0) {
    perror(pathname);
    exit(1);
}

printf("ino =  %10ld # Inode number\n", s.st_ino);
printf("mode = %10o # File mode \n", s.st_mode);
printf("nlink =%10ld # Link count \n", (long)s.st_nlink);
printf("uid =  %10u # Owner uid\n", s.st_uid);
printf("gid =  %10u # Group gid\n", s.st_gid);
printf("size = %10ld # File size (bytes)\n", (long)s.st_size);

printf("mtime =%10ld # Modification time (seconds since 1/1/70)\n",
       (long)s.st_mtime);
```

stat.st_mode

- Probably the most important part of stat.h alongside length (for 1521)
- Usage:

```
if (s.st_mode & S_IRUSR)          {printf(“user has read permission”);}
if ((perms & S_IFMT) == S_IFDIR) {printf(“This file is a directory!!”);}
```

- use this to check for permissions and check if file is a directory!

write recurse-sample.c:
a file that recurses through a directory and prints out permissions and filenames of all files with "hello" in it's name.

// part 0: get and print out current directory pathname.

// part 1: print out all files in current directory.

// part 2: recurse through all subdirectories and print out pathnames.

// part 3: only print out pathnames with "hello" in the filename.

Search for a string

Syntax

Following is the syntax of the C library **strstr()** function –

```
char *strstr (const char *str_1, const char *str_2);
```

Parameters

This function takes two parameters –

- **str_1** – This is a main string.
- **str_2** – The substring to be searched in main string i.e. str_1.

Return Value

The function return the value based on following conditions –

- The function returns a pointer to the first characters of str_2 in str_1 or null pointer if str_2 is not found in str_1.
- If str_2 is found as an empty string, str_1 is returned.

write recurse-sample.c:
a file that recurses through a directory and prints out permissions and filenames of all files with "hello" in it's name.

// part 0: get and print out current directory pathname.

// part 1: print out all files in current directory.

// part 2: recurse through all subdirectories and print out pathnames.

// part 3: only print out pathnames with "hello" in the filename.

// part 4: print out the permissions of these files.

Permissions - s.st_mode

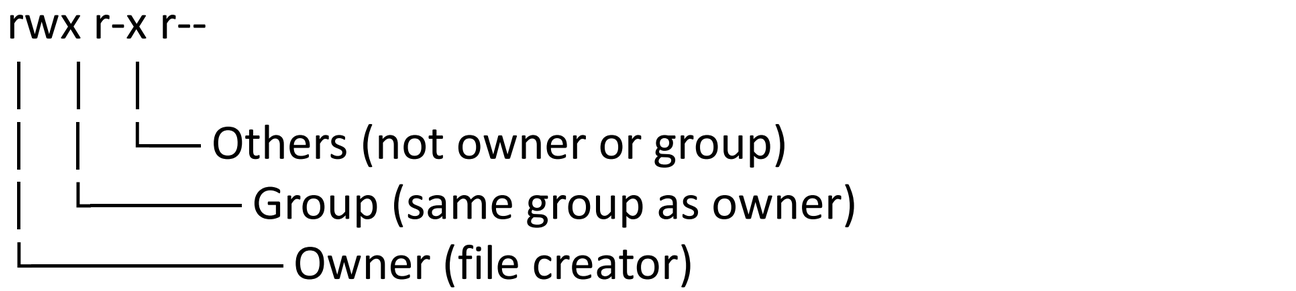
```
if (s.st_mode & S_IRUSR)         {printf(“user has read permission”);}
if ((perms & S_IFMT) == S_IFDIR) {printf(“This file is a directory!!”);}
```

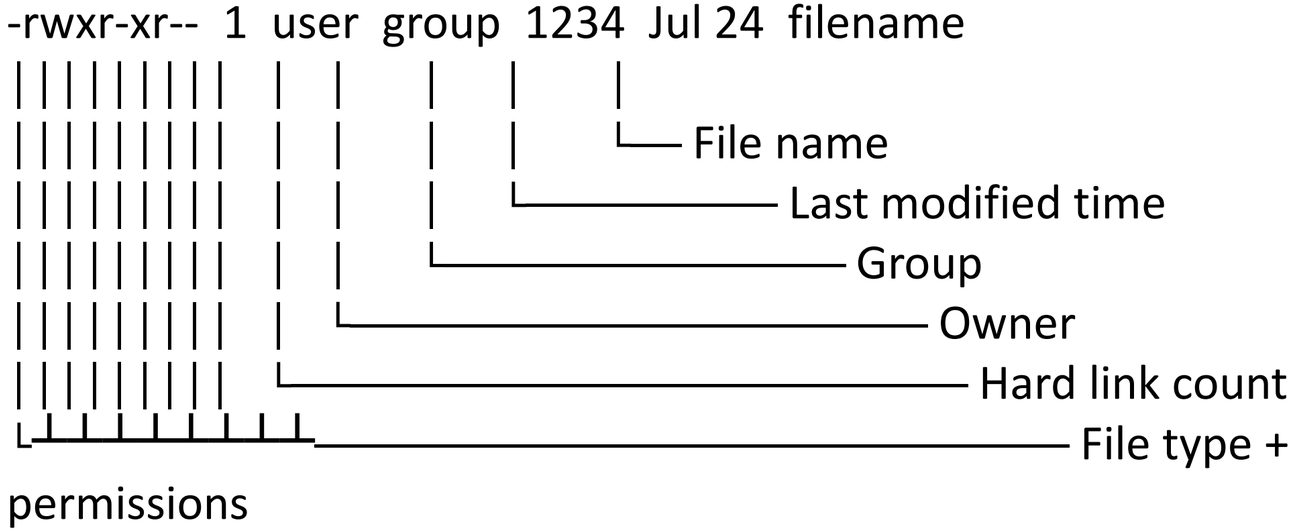
Mode_t (octal):
755

```c
int main() {
    struct stat s;
    stat("a_a", &s);
    printf("whole mode: %o\n", s.st_mode);

    // only look at last 9
    uint32_t mask = 0777;
    uint32_t perms = s.st_mode;
    perms = perms & mask;


    printf("%o\n", perms);
    // printf("%b\n", perms);
    return 0;
}
```

```
z5360323@vx14:~/COMP1521-25T1/tutes/week9$ ./demo
whole mode: 40755
755
```

Linux ls -l Permission Format：





chmod：

What is the meaning for chomd 777?

chmod：

What is the meaning for chomd 777?

Octal	Binary	Symbolic	Meaning
7	111	rwx	Read, Write, Execute
6	110	rw-	Read, Write
5	101	r-x	Read, Execute
4	100	r--	Read only
0	000	---	No permission

chmod：

What is the meaning for chomd 777?

Octal	Binary	Symbolic	Meaning
7	111	rwx	Read, Write, Execute
6	110	rw-	Read, Write
5	101	r-x	Read, Execute
4	100	r--	Read only
0	000	---	No permission

chmod 777 = Everyone can do anything (rwx)

Linux Permission Macro Matrix

	Owner	**Group**	**Others**
Read (r)	S_IRUSR	S_IRGRP	S_IROTH
Write (w)	S_IWUSR	S_IWGRP	S_IWOTH
Exec (x)	S_IXUSR	S_IXGRP	S_IXOTH

Permissions - s.st_mode

```
if (s.st_mode & S_IRUSR)         {printf("user has read permission");}
if ((perms & S_IFMT) == S_IFDIR) {printf("This file is a directory!!");}
```

Mode_t (octal): 755

Binary: 111 101 101

```c
int main() {
    struct stat s;
    stat("sample", &s);
    printf("whole mode: %o\n", s.st_mode);

    // only look at last 9
    uint32_t mask = 0777;
    uint32_t perms = s.st_mode;
    perms = perms & mask;


    printf("%o\n", perms);
    printf("%b\n", perms);
    return 0;
}
```

```
z5360323@vx14:~/COMP1521-25T1/tutes/week9$ ./demo
whole mode: 40755
755
111101101
```

Permissions - s.st_mode

```
if (s.st_mode & S_IRUSR)         {printf("user has read permission");}
if ((perms & S_IFMT) == S_IFDIR) {printf("This file is a directory!!");}
```

Mode_t (octal): 755

Binary: 111 101 101

String: rwx r-x r-x

```c
int main() {
    struct stat s;
    stat("a_a", &s);
    printf("whole mode: %o\n", s.st_mode);

    // only look at last 9
    uint32_t mask = 0777;
    uint32_t perms = s.st_mode;
    perms = perms & mask;


    printf("%o\n", perms);
    printf("%b\n", perms);
    return 0;
}
```

```
z5360323@vx14:~/COMP1521-25T1/tutes/week9$ ./demo
whole mode: 40755
755
111101101
z5360323@vx14:~/COMP1521-25T1/tutes/week9$ ls -l
total 216
drwxr-xr-x 2 z5360323 z5360323   4096 Apr 14 10:05 a_a
```

Permissions - s.st_mo

```
if (s.st_mode & S_IRUSR)        {printf("user has read permission");}
if ((perms & S_IFMT) == S_IFDIR) {printf("This file is a directory!!");}
```

Mode_t (octal): 755

Binary: 111 101 101

String: rwx r-w r-w

To convert from binary to string,

- print “r/w/x” if bin = 1
- else print “-”.
- each binary digit has a hash define!

```c
int main() {
    struct stat s;
    stat("a_a", &s);
    printf("whole mode: %o\n", s.st_mode);

    // only look at last 9
    uint32_t mask = 0777;
    uint32_t perms = s.st_mode;
    perms = perms & mask;


    printf("%o\n", perms);
    printf("%b\n", perms);
    return 0;
}
```

```
z5360323@vx14:~/COMP1521-25T1/tutes/week9$ ./demo
whole mode: 40755
755
111101101
z5360323@vx14:~/COMP1521-25T1/tutes/week9$ ls -l
total 216
drwxr-xr-x 2 z5360323 z5360323   4096 Apr 14 10:05 a_a
```

Permissions - s.st_mode

```
if (s.st_mode & S_IRUSR)         {printf(“user has read permission”);}
if ((perms & S_IFMT) == S_IFDIR) {printf(“This file is a directory!!”);}
```

```
55        (perms & S_IRUSR) ? printf("r") : printf("-");
56        (perms & S_IWUSR) ? printf("w") : printf("-");
57        (perms & S_IXUSR) ? printf("x") : printf("-");
58        (perms & S_IRGRP) ? printf("r") : printf("-");
59        (perms & S_IWGRP) ? printf("w") : printf("-");
60        (perms & S_IXGRP) ? printf("x") : printf("-");
61        (perms & S_IROTH) ? printf("r") : printf("-");
62        (perms & S_IWOTH) ? printf("w") : printf("-");
63        (perms & S_IXOTH) ? printf("x") : printf("-");
64
```

What is UTF-8?

- UTF-8 stands for 8-bit Unicode Transformation Format

- It's a variable-length encoding for Unicode codepoints

- Each character (codepoint) is encoded using 1 to 4 bytes

- Compatible with ASCII (0x00 to 0x7F)

UTF-8 (cont.)

This is the layout of UTF-8

#bytes	*#bits*	*Byte 1*	*Byte 2*	*Byte 3*	*Byte 4*
1	7	`0xxxxxxx`	-	-	-
2	11	`110xxxxx`	`10xxxxxx`	-	-
3	16	`1110xxxx`	`10xxxxxx`	`10xxxxxx`	-
4	21	`11110xxx`	`10xxxxxx`	`10xxxxxx`	`10xxxxxx`

A single UTF-8 character can be anywhere from 1 to 4 bytes long

The entire ASCII character set can be represented in 1 byte with zero wasted bits

The entire BMP can be represented in 3 bytes, being 8 bits more efficient than UTF-32

and the entire UNICODE character set can be represented in 4 bytes/ using exactly the same number of bits as UTF-32 in the worst case

Bitmask Explanation

#bytes	#bits	Byte 1	Byte 2	Byte 3	Byte 4
1	7	0xxxxxxx	-	-	-
2	11	110xxxxx	10xxxxxx	-	-
3	16	1110xxxx	10xxxxxx	10xxxxxx	-
4	21	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Mask	Matches Pattern	Meaning
0x80	0xxxxxxx	1-byte (ASCII)
0xE0	110xxxxx	2-byte start
0xF0	1110xxxx	3-byte start
0xF8	11110xxx	4-byte start

How to Determine Character Length in C

```c
int utf8_char_length(unsigned char byte) {
    if ((byte & 0x80) == 0) return 1;              // 0x80 = 10000000
                                                   // 0xC0 = 11000000
    else if ((byte & 0xE0) == 0xC0) return 2;      // 0xE0 = 11100000
    else if ((byte & 0xF0) == 0xE0) return 3;      // 0xF0 = 11110000
    else if ((byte & 0xF8) == 0xF0) return 4;      // 0xF8 = 11111000
    else return -1;
}
```

Tut.q8

View the number of characters (codepoints)

echo -n "早上好中国现在我有冰淇淋" | wc –m

View the number of bytes (total length after UTF-8 encoding):

echo -n "早上好中国现在我有冰淇淋" | wc -c

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`.