Week 09 Laboratory Sample Solutions

Objectives

- learning how to access file metadata via stat
- learning how to use file metadata
- practicing file operations generally
- understand make's core operation

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples.

Getting Started

Create a new directory for this lab called `lab09`, change to this directory, and fetch the provided code for this week by running these commands:

```
$ mkdir lab09
$ cd lab09
$ 1521 fetch lab09
```

Or, if you're not working on CSE, you can download the provided code as a zip file or a tar file.

EXERCISE — INDIVIDUAL:

Print Files Sizes

We are worried about disk usage and would like to know how much space is used used by a set of files

Write a C program, `file_sizes.c`, which is given one or more filenames as command line arguments. It should print one line for each filename which gives the size in bytes of the file. It should also print a line giving the combined number of bytes in the files.

Follow the output format below.

Do not read the file - obtain the file size from the function *stat*.

```
$ dcc file_sizes.c -o file_sizes
$ ./file_sizes bubblesort.c print_bigger.c swap_numbers.c unordered.c
bubblesort.c: 667 bytes
print_bigger.c: 461 bytes
swap_numbers.c: 565 bytes
unordered.c: 486 bytes
Total: 2179 bytes
$ ./file_sizes bubblesort.s print_bigger.s swap_numbers.s unordered.s numbers1.txt numbers2.txt sorted.txt
bubblesort.s: 1142 bytes
print_bigger.s: 1140 bytes
swap_numbers.s: 1173 bytes
unordered.s: 791 bytes
numbers1.txt: 59 bytes
numbers2.txt: 55 bytes
sorted.txt: 21 bytes
Total: 4381 bytes
```

HINT:

The stat.c prints the file size in bytes.

The number of bytes in a file will not fit in an *int*.

HINT:

You can assume only ordinary files (not directories, links, ...) are specified as arguments.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest file_sizes
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab09_file_sizes file_sizes.c
```

You must run `give` before **Monday 16 November 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

Sample solution for `file_sizes.c`

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

off_t file_size(char *pathname);

int main(int argc, char *argv[]) {
    off_t total_bytes = 0;
    for (int arg = 1; arg < argc; arg++) {
        total_bytes += file_size(argv[arg]);
    }
    printf("Total: %ld bytes\n", total_bytes);
    return 0;
}

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

    printf("%s: %ld bytes\n", pathname, (long)s.st_size);
    return s.st_size;
}
```

EXERCISE — INDIVIDUAL:

Print File Modes

We would like to print the access permissions for a set of files

Write a C program, `file_modes.c`, which is given one or more pathnames as command line arguments. It should print one line for each pathnames which gives the permissions of the file or directory.

Follow the output format below.

```
$ dcc file_modes.c -o file_modes
$ 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
$ ./file_modes file_modes file_modes.c file_sizes file_sizes.c
-rwxr-xr-x file_modes
-rw-r--r-- file_modes.c
-rwxr-xr-x file_sizes
-rw-r--r-- file_sizes.c
$ chmod 700 file_modes
$ chmod 640 file_sizes.c
$ chmod 600 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----- 1 z5555555 z5555555    604 Nov  2 12:58 file_sizes.c
-rwx------ 1 z5555555 z5555555 222672 Nov  2 13:00 file_modes
-rw------- 1 z5555555 z5555555   2934 Nov  2 12:59 file_modes.c
$ ./file_modes file_modes file_modes.c file_sizes file_sizes.c
-rwx------ file_modes
-rw------- file_modes.c
-rwxr-xr-x file_sizes
-rw-r----- file_sizes.c
```

The first character on each line should be '**-**' for ordinary files and '**d**' for directories. For example:

```
$ ./file_modes /tmp /web/cs1521/index.html
drwxrwxrwx /tmp
-rw-r--r-- /web/cs1521/index.html
```

HINT:

The stat.c lecture example prints the permissions as an octal number.

NOTE:

You can assume only ordinary files and directories (not links, devices, ...) are specified as arguments.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest file_modes
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab09_file_modes file_modes.c
```

You must run `give` before **Monday 16 November 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

Sample solution for `file_modes.c`

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

void file_mode(char *pathname);

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

// convert octal mode to -rwxrwxrwx string and print itm

void file_mode(char *pathname) {
    struct stat s;
    if (stat(pathname, &s) != 0) {
        perror(pathname);
        exit(1);
    }
    mode_t mode = s.st_mode;

    char permissions[] = "?rwxrwxrwx";
    int n_permissions = strlen(permissions);

    if (S_ISREG(mode)) {
        permissions[0] = '-';
    } else if (S_ISDIR(mode)) {
        permissions[0] = 'd';
    }

    for (int i = 1; i < n_permissions; i++) {
        if (!(mode & (1 << (i - 1)))) {
            permissions[n_permissions - i] = '-';
        }
    }

    printf("%s %s\n", permissions, pathname);
}
```

CHALLENGE EXERCISE — INDIVIDUAL:

Compile C Files If Needed

You have been given compile_if_needed.c, a C program that given the pathname of single file C programs compiles them. For example:

```
$ dcc compile_if_needed.c -o compile_if_needed
$ ./compile_if_needed file_sizes.c file_modes.c
/usr/local/bin/dcc file_modes.c -o file_modes
/usr/local/bin/dcc file_sizes.c -o file_sizes
```

Unfortunately the code you have been given is inefficient. It recompiles the C file even if this is not necessary.

Modify `compile_if_needed.c` so that it only compiles C files if necessary.

A compilation is necessary if the binary doesn't exist.

A compilation is also necessary if the C file has been changed since the last compilation. In other words, if the modification time of the C file is more recent than the binary.

Follow the output format below.

```
$ ./compile_if_needed file_sizes.c file_modes.c
/usr/local/bin/dcc file_modes.c -o file_modes
/usr/local/bin/dcc file_sizes.c -o file_sizes
$ ./compile_if_needed file_sizes.c file_modes.c
file_modes.c does not need compiling
file_sizes.c does not need compiling
$ rm file_sizes
$ ./compile_if_needed file_sizes.c file_modes.c
file_modes.c does not need compiling
/usr/local/bin/dcc file_sizes.c -o file_sizes
$ echo >> file_sizes.c # add a new-line to file_sizes.c
$ ./compile_if_needed file_sizes.c file_modes.c
file_modes.c does not need compiling
/usr/local/bin/dcc file_sizes.c -o file_sizes
$ ./compile_if_needed file_sizes.c file_modes.c
file_modes.c does not need compiling
file_sizes.c does not need compiling
$ ./compile_if_needed file_sizes.c file_modes.c
file_modes.c does not need compiling
file_sizes.c does not need compiling
```

HINT:

The stat.c lecture example prints a file modification time.

NOTE:

BTW this is essentially what the program *make* does.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest compile_if_needed
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab09_compile_if_needed compile_if_needed.c
```

You must run `give` before **Monday 16 November 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

Sample solution for `compile_if_needed.c`

```c
// compile .c files specified as command line arguments if needed
//
// if my_program.c is speicified as an argument
// /usr/local/bin/dcc my_program.c -o my_program
// will be executed unless my_program exists
// and my_program's modification time is more recent than my_program.c

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

void compile_if_needed(char *c_file);
int is_compile_needed(char *c_file, char *binary);
void compile(char *c_file, char *binary);
char *get_binary_name(char *c_file);

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


// compile a C file if needed
void compile_if_needed(char *c_file) {
    char *binary = get_binary_name(c_file);
    if (is_compile_needed(c_file, binary)) {
        compile(c_file, binary);
    } else {
        printf("%s does not need compiling\n", c_file);
    }
    free(binary);
}


// test if we need to recompile a C file
// return 1 if binary does not exist
//  or modification time of C file more recent than binary
// return 0, otherwise
int is_compile_needed(char *c_file, char *binary) {
    struct stat s_c_file;
    if (stat(c_file, &s_c_file) != 0) {
        perror(c_file);
        exit(1);
    }

    struct stat s_binary;
    if (stat(binary, &s_binary) != 0) {
        // stat of binary failed so recompile needed
        return 1;
    }

    // st_mtime contains file modification as seconds since Jan 1 1970
    return s_c_file.st_mtime >= s_binary.st_mtime;
}


#define C_COMPILER "/usr/local/bin/dcc"

// compile a C file
void compile(char *c_file, char *binary) {
    pid_t pid;
    extern char **environ;
    char *cc_argv[] = {C_COMPILER, c_file, "-o", binary, NULL};

    // print compile command
    for (char **p = cc_argv; *p; p++) {
        printf("%s ", *p);
    printf("\n");

    // run compile command
    if (posix_spawn(&pid, C_COMPILER, NULL, NULL, cc_argv, environ) != 0) {
        perror("spawn");
        exit(1);
    }

    int exit_status;
    if (waitpid(pid, &exit_status, 0) == -1) {
        perror("waitpid");
        exit(1);
    }

    if (exit_status != 0) {
        fprintf(stderr, "compile failed\n");
        exit(1);
    }
}


// give a string ending in .c
// return malloc-ed copy of string without .c

char *get_binary_name(char *c_file) {
    char *binary = strdup(c_file);
    if (binary == NULL) {
        perror("");
        exit(1);
    }

    // remove .c suffix
    char *last_dot = strrchr(binary, '.');
    if (last_dot == NULL || last_dot[1] != 'c' || last_dot[2] != '\0') {
        fprintf(stderr, "'%s' does not end in  .c\n", c_file);
        exit(1);
    }
    *last_dot = '\0';
    return binary;
}
```

CHALLENGE EXERCISE — INDIVIDUAL:

ls -ld

We need clean room implementation of the standard Unix program *ls*.

Write a C program, `lsld.c`, which is given zero or more pathnames as command line arguments produces exactly the same output as *ls -ld* given the same pathnames as arguments.

Except *ls -ld* sorts its output lines. You do not have to match this.

Follow the output format below.

```
$ dcc lsld.c -o lsld
$ ls -ld lsld.c lsld file_sizes.c file_sizes /home/cs1521/public_html
drwxr-xr-x 6 cs1511   cs1511      128 Sep 16 08:02 /home/cs1521/public_html
-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 lsld
-rw-r--r-- 1 z5555555 z5555555   2934 Nov  2 12:59 lsld.c
$ ./lsld lsld.c lsld file_sizes.c file_sizes /home/cs1521/public_html
-rw-r--r-- 1 z5555555 z5555555   2934 Nov  2 12:59 lsld.c
-rwxr-xr-x 1 z5555555 z5555555 222672 Nov  2 13:00 lsld
-rw-r--r-- 1 z5555555 z5555555    604 Nov  2 12:58 file_sizes.c
-rwxr-xr-x 1 z5555555 z5555555 116744 Nov  2 13:00 file_sizes
drwxr-xr-x 6 cs1511   cs1511      128 Sep 16 08:02 /home/cs1521/public_html
```

HINT:

The functions `getpwuid_r` and `getgrgid_r` may be useful.

NOTE:

Your solution must be in C only.
You are not permitted to run external programs. You are not permitted to use system, popen, posix_spawn, fork or exec.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest lsld
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab09_lsld lsld.c
```

You must run `give` before **Monday 16 November 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

Sample solution for `lsld.c`

```c
// lsld.c

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

char *rwxmode (mode_t);
char *username (uid_t);
char *groupname (gid_t);
char *gettime(time_t);

int main (int argc, char **argv)
{
    struct stat info;

    if (argc == 1)
        if (stat(".", &info) == -1) perror("stat");
        else {
            char *rwx_str = rwxmode(info.st_mode);
            char *uname_str = username(info.st_uid);
            char *gname_str = groupname(info.st_gid);
            char *time_str = gettime(info.st_mtime);

            printf ("%s %lu %s %s %ld %s %s\n",
                rwx_str,
                info.st_nlink,
                uname_str,
                gname_str,
                info.st_size,
                time_str,
                "."
            );
            free(rwx_str);
            free(uname_str);
            free(gname_str);
            free(time_str);
        }
    else for (int i = 1; i < argc; i++)
        if (stat(argv[i], &info) == -1) perror("stat");
        else {
            char *rwx_str = rwxmode(info.st_mode);
            char *uname_str = username(info.st_uid);
            char *gname_str = groupname(info.st_gid);
            char *time_str = gettime(info.st_mtime);

            printf ("%s %lu %s %s %ld %s %s\n",
                rwx_str,
                info.st_nlink,
                uname_str,
                gname_str,
                info.st_size,
                time_str,
                argv[i]
            );
            free(rwx_str);
            free(uname_str);
            free(gname_str);
            free(time_str);
        }
    return EXIT_SUCCESS;
}

// convert octal mode to -rwxrwxrwx string
char *rwxmode (mode_t mode)
{
    char *perms = calloc(11, sizeof(char));

    switch (mode & S_IFMT) {
        case S_IFREG: perms[0] = '-'; break;
        case S_IFDIR: perms[0] = 'd'; break;
        case S_IFDIR: perms[0] = 'd'; break;
        case S_IFLNK: perms[0] = 'l'; break;
        default:      perms[0] = '?'; break;
    }

    perms[1]  = (mode & S_IRUSR) ? 'r' : '-';
    perms[2]  = (mode & S_IWUSR) ? 'w' : '-';
    perms[3]  = (mode & S_IXUSR) ? 'x' : '-';
    perms[4]  = (mode & S_IRGRP) ? 'r' : '-';
    perms[5]  = (mode & S_IWGRP) ? 'w' : '-';
    perms[6]  = (mode & S_IXGRP) ? 'x' : '-';
    perms[7]  = (mode & S_IROTH) ? 'r' : '-';
    perms[8]  = (mode & S_IWOTH) ? 'w' : '-';
    perms[9]  = (mode & S_IXOTH) ? 'x' : '-';
    perms[10] = '\0';

    return perms;
}

// convert user id to user name
char *username (uid_t uid)
{
    char *uname = NULL;

    struct passwd uinfo, *result;
    char *buf;
    long bufsize;

    bufsize = sysconf(_SC_GETPW_R_SIZE_MAX);
    if (bufsize == -1) bufsize = 1024;

    buf = alloca(bufsize);
    if (buf == NULL) {
        perror("alloca");
        exit(EXIT_FAILURE);
    }

    getpwuid_r(uid, &uinfo, buf, bufsize, &result);

    if (result != NULL) {
        uname = realloc(uname, snprintf (NULL, 0, "%s", uinfo.pw_name) + 1);
        sprintf(uname, "%s", uinfo.pw_name);
    } else {
        uname = realloc(uname, snprintf (NULL, 0, "%d?", (int)uid) + 1);
        sprintf(uname, "%d?", (int)uid);
    }

    return uname;
}

// convert group id to group name
char *groupname (gid_t gid)
{
    char *gname = NULL;

    struct group ginfo, *result;
    char *buf;
    long bufsize;

    bufsize = sysconf(_SC_GETGR_R_SIZE_MAX);
    if (bufsize == -1) bufsize = 1024;

    buf = alloca(bufsize);
    if (buf == NULL) {
        perror("alloca");
        exit(EXIT_FAILURE);
    }

    getgrgid_r(gid, &ginfo, buf, bufsize, &result);

    if (result != NULL) {
        gname = realloc(gname, snprintf(NULL, 0, "%s", ginfo.gr_name) + 1);
        sprintf(gname, "%s", ginfo.gr_name);
    } else {
        gname = realloc(gname, snprintf(NULL, 0, "%d?", (int)gid) + 1);
        sprintf(gname, "%d?", (int)gid);
    }

    return gname;
}

char *gettime(time_t file_time)
{
#define TIME_SIZE 128
    char *time_str = calloc(TIME_SIZE, sizeof(char));

    time_t now = time(NULL);
    struct tm tm_file, tm_now;

    localtime_r(&file_time, &tm_file);
    localtime_r(&now, &tm_now);

    if (tm_file.tm_year == tm_now.tm_year) { /* if year is current output date/time */
        // strftime has no way of returning length like snprintf(3) does;
        // time_str = realloc(time_str, strftime(NULL, 0, "%b %e %H:%M", &tm_file));
        strftime(time_str, TIME_SIZE, "%b %e %H:%M", &tm_file);
    } else { /* if year is not current, output time/year */
        // strftime has no way of returning length like snprintf(3) does;
        // time_str = realloc(time_str, strftime(NULL, 0, "%b %e %Y", &tm_file));
        strftime(time_str, TIME_SIZE, "%b %e %Y", &tm_file);
    }

    return time_str;
#undef TIME_SIZE
}
```

Submission

When you are finished each exercises make sure you submit your work by running `give`.

You can run `give` multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Mon Nov 16 21:00:00 2020** to submit your work.

You cannot obtain marks by e-mailing your code to tutors or lecturers.

You check the files you have submitted here.

Automarking will be run by the lecturer several days after the submission deadline, using test cases different to those `autotest` runs for you. (Hint: do your own testing as well as running `autotest`.)

After automarking is run by the lecturer you can view your results here. The resulting mark will also be available via give's web interface.

Lab Marks

When all components of a lab are automarked you should be able to view the the marks via give's web interface or by running this command on a CSE machine:

```
$ 1521 classrun -sturec
```