C and Linux Programming
Eastern Washington University
Computer Science
March 30th – June 12th, 2020

Lecture 4

C and Linux File Handling

Joe Dumoulin

Mar 30, 2020

Eastern Washington University

C and Linux File Handling

The Story So Far

Basic C syntax

- Variables, Statements, and Functions
- ASCII Characters and Strings
- Arrays

Next we will use these concepts to see how to work with sets of strings and files in Unix and Linux systems.

EASTERN WASHINGTON UNIVERSITY

What is a File?

A **File** is a way to store information.

Unlike memory, Files are persistent.

Files provide input to programs and provide ways to store output from programs.

Files are also used to pass input from one process to another.

Everything you store on your computer is a file.

EASTERN WASHINGTON UNIVERSITY

What is a File?

We can think of files as being divided into two useful categories:

Binary Files - Files which are composed of bytes of data.

Text Files - Files that are composed of ASCII character data only.

EASTERN WASHINGTON UNIVERSITY

Binary Files

Binary files typically store non-text content like music, pictures, movies, drawings, compiled programs, etc.

we can look at binary files using the command line program **hexdump**. For example:

```
>hexdump -C Rainbow_Unicorn.jpg | head -n 20
00000000  ff d8 ff e0 00 10 4a 46  49 46 00 01 01 00 00 01  |......JFIF......|
00000010  00 01 00 00 ff db 00 43  00 08 06 06 07 06 05 08  |.......C........|
00000020  07 07 07 09 09 08 0a 0c  15 0e 0c 0b 0b 0c 19 12  |................|
00000030  13 0f 15 1e 1b 20 1f 1e  1b 1d 1d 21 25 30 29 21  |..... .....!%0)!|
00000040  23 2d 24 1d 1d 2a 39 2a  2d 31 33 36 36 36 20 28  |#-$..*9*-13666 (|
00000050  3b 3f 3a 34 3e 30 35 36  33 ff db 00 43 01 09 09  |;?:4>0563...C...|
00000060  09 0c 0b 0c 18 0e 0e 18  33 22 1d 22 33 33 33 33  |........3"."3333|
00000070  33 33 33 33 33 33 33 33  33 33 33 33 33 33 33 33  |3333333333333333|
```

Programs are Binary Files

Compiled programs are binary files as well. We have already seen how we can display the structure of binary files using **objdump**.

```
> hexdump -C simple_file_open
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  03 00 3e 00 01 00 00 00  60 06 00 00 00 00 00 00  |..>.....`.......|
00000020  40 00 00 00 00 00 00 00  e8 19 00 00 00 00 00 00  |@...............|
00000030  00 00 00 00 40 00 38 00  09 00 40 00 1d 00 1c 00  |....@.8...@.....|
00000040  06 00 00 00 04 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000050  40 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |@.......@.......|
00000060  f8 01 00 00 00 00 00 00  f8 01 00 00 00 00 00 00  |................|
00000070  08 00 00 00 00 00 00 00  03 00 00 00 04 00 00 00  |................|
```

The file type of a linux executable is 'ELF'.

EASTERN WASHINGTON UNIVERSITY

Text Files

Text files are composed entirely of ASCII characters. These are programs, configuration files, some datafiles, and text corpora like the gutenberg press.

You can view text files using tools like **cat** and **less**.

Editing text files can be done with **nano**, **vim**, or **emacs**.

EASTERN WASHINGTON UNIVERSITY

Working With Files in C - opening a file

C programs often expect to open a file to get input for the program.

You identify the file to open by its path and the progrm identifies the open file by a **file descriptor**.

The file descriptor is an integer that uniquely represents an open file in your program.

EASTERN WASHINGTON UNIVERSITY

Working With Files in C - open

To read a file into a program, we first open the file. We use the **open** function to open a file.

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char** argv)
{
  int fd = open("../corpora/austen-emma.txt", O_RDONLY);
  if (fd == -1) {           // if the fd is less than 0, there was a problem
    perror("open");         // print an error.
    return EXIT_FAILURE;    // exit the program with failure
  }

  printf("the file descriptor for emma is: %d\n",fd);
  int re = close(fd);  // when you are done with the file, close it.
  if (re == -1){       // close will return -1 for an error
    perror("close");   // print the error.
    return EXIT_FAILURE; // exit with fail
  }
  return EXIT_SUCCESS;   // success!
}
```

EASTERN WASHINGTON UNIVERSITY

Working With Files in C - open

What does this program do?

- load modules for constants and functions we need
- call **open** on the file for reading (the text of Jane Austen's *Emma*)
- **open** could fail if the file isn't found for example. so check and end the program on failure.
- Do the work of the program. in this case, just print out the file descriptor.
- close the file before exiting using the **close** function.
- **close** returns -1 if it fails. If so, return with failure.
- if all goes well, return success.

EASTERN WASHINGTON UNIVERSITY

Reading the File Contents

What if I want to read some of the contents of a file? Then I follow two steps (well, three).

- open the file for reading. (using O_RDONLY).
- read the data I need.
- close the file.

Let's see an example...

EASTERN WASHINGTON UNIVERSITY

Reading the File Contents - open

Opening the file

Open the file. Specify a size for the string to read and identify the file path.

```
#define MAX_BUF_SIZE 1024          // a constant character array
                                   // reading text from the file.
int main(int argc, char** argv)
{
  char buffer[MAX_BUF_SIZE];       // the char array (string to h
                                   // from the file.
  // open the file if possible
  int fd = open("../corpora/austen-emma.txt", O_RDONLY);
  if (fd == -1) {
    perror("open");
    return EXIT_FAILURE;
  }
...
```

Reading the File Contents - read

Next we use the **read** function to read the first MAX_BUF_SIZE characters (or bytes) from the input file.

```
...
  // read the buffer
  long int rd_sz = -1; // assume nothing
  rd_sz = read(fd, buffer, MAX_BUF_SIZE-1);
  if (rd_sz == -1) {
    perror("read");
    return EXIT_FAILURE;
  }
  printf("%s", buffer);      // print the contents f the buffer
  printf("\n");

  printf("%ld\n",rd_sz);     // print the amount of text read
...
```

Closing the file - close

Before the end of a program that opens files, all files open using the **open** function should be closed.

```
...
  int re = close(fd);
  if (re == -1){
    perror("close");
    return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
```

Question: what does this program do?

file reading - stdin

The program described above reads the first 1024 bytes (characters) of the file called *austen-emma.txt* and then prints it to the console. Finally it closes the text file.

There is a file that is always open for reading. That is called **stdin** and it points at the console. We will see more about this later.

EASTERN WASHINGTON UNIVERSITY

reading and writing - copying a file's contents

To demonstrate file writing, we will build a program that creates a new file containing the contents of the file we opened for reading.

In order to write to a file, we need to tell the system a few things about the file:

- The name and location of the new file
- the user attributes of the file
- possibly the type of file when it is other than a disk file (more about this later)
- whether we should truncate an existing file, append to an existing file or create a new file.

When we open a file for writing, we get another file descriptor.

One file descriptor is always open for writing in our programs: **stdout**. this file descriptor is connected to the terminal so that we can see the output to this file.

EASTERN WASHINGTON UNIVERSITY

copying a file

In our first example, we add code to open a file for writing

```
int fdwt = open(outfile, O_WRONLY | O_CREAT | O_TRUNC,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)
if (fdwt == -1) {
  perror("open write");
  return EXIT_FAILURE;
}
```

This **open** call is more complicated than the one for reading, so let's break it down.

EASTERN WASHINGTON UNIVERSITY

open for writing

- int fdwt = open(...) declare and assign the file descriptor
- ... (outfile , ...) this is a string that contains the name of the file to create or write.
- (..., O_WRONLY | O_CREAT | O_TRUNC, ...) open the file for writing, create the file if it doesn't exist, and if it does exist, delete its contents before writing.
- (..., S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | ...); set read and write privileges for all future users of the file.

EASTERN WASHINGTON UNIVERSITY

Copying a file

Now that we can open a file for writing, we can write our first **copy** program.

First we need to identify the input and output file names. We will use a pair of constant strings for the simple version.

```
const char infile[] = "../corpora/austen-emma.txt";
const char outfile[] = "../corpora/austen-emma-copy.txt";
```

EASTERN WASHINGTON UNIVERSITY

Copying a file

Next we need to read some text and then write some text until we have read and written the entire input file.

```
// while the buffer can be filled
while ((rd_sz = read(fdrd, buffer, MAX_BUF_SIZE-1)) > 0) {

  if((wt_sz = write(fdwt, buffer, rd_sz) > 0)) {

    // loop while writing successfully
  }
  else break;
```

rd_sz is the number of characters read. each time through the loop, **read** will get the next set of bytes and **write** will write as many bytes as it can until the input file is completely read.

When the last characters are read from the input, **read** will return 0 and the loop will end.

EASTERN WASHINGTON UNIVERSITY

Copying a file - Getting file names

Getting file names from the command line is a small step from the previous code. First we use the alternate Main declaration:

```
int main(int argc, char* argv[])
{
  char infile[200] = "";
  char outfile[200] = "";
```

Copying a file - Getting file names

Next we check the arguments to extract the file name. If the command line isn't right, we will tell the user how to type it correctly.

```
// check command line arguments
if (argc >= 3) {
  strcpy(infile, argv[1]);
  strcpy(outfile, argv[2]);
}
else {
  printf("better_copy:\n\tusage: better_copy <<in_file_path>> \
     <<out_file_path>>\n");
  return EXIT_FAILURE;
}
```

EASTERN WASHINGTON UNIVERSITY

working with files - homework

What have we done above? We have implemented a simple version of the program **cp**!

Remember that we have the file descriptors **stdin** and **stdout** defined automatically in our program. we can use these for input and output from our programs without needing to open or close the files. They are always open.

The next assignment will involve using the sample programs we described above to implement a version of **cat**.

EASTERN WASHINGTON UNIVERSITY