UNSW SYDNEY | Australia's Global University

COMP1911 - Computing 1A

11. Reading and Writing Files

In this lecture we will cover:

- Linux I/O direction
- Accessing files using C
- ASCII file and binary file

Unix Processes

- A process is a program in execution in memory – a running instance.
- Any program executed creates a process.
- When you execute a program on your Unix system, the system creates a special environment for that program.
- This environment contains everything needed for the system to run the program as if no other program were running on the system.

Standard Streams

- Stream is a sequence of bytes
- **stdio.h** define three streams
- **stdin** standard input stream

  scanf, getchar read from stdin
- **stdout** standard output stream

  printf, putchar write to stdout
- **stderr** standard error stream

  by convention used for error messages

I/O direction

- If program runs from terminal: stdin, stdout, stderr connected to terminal.
- Unix shells allow you to re-direct stdin, stdout and stderr.
- Run **a.out** with stdin coming from file **data.txt**
  ```
  ./a.out < data.txt
  ```
- Run **a.out** with stdout going to (overwriting) file **output.txt**
  ```
  ./a.out > output.txt
  ```
- Run **a.out** with stdout appended to file **output.txt**
  ```
  ./a.out >> output.txt
  ```
- Run **a.out** with stderr going to file **errors.txt**
  ```
  ./a.out 2> errors.txt
  ```
- Run **a.out** with stdout AND stderr going to file **allOut.txt**
  ```
  ./a.out &> allOut.txt
  ```

Using stderr for error Messages

- **fprintf** allows you to specify stream to print to
- For example:

```
fprintf(stderr, "error: can not open %s\n", argv[1]);
```

- *printf* actually just calls *fprintf* specifying stdout

```
fprintf(stdout, ...);
```

- Best if error messages written to stderr, so users sees them even if stdout is directed to a file.
- Common for stderr to be redirected separately to a log file for system programs.

Accessing Files

- **FILE *f = fopen(char *filename, char *mode)**

Create a stream to read from or write to file
returns NULL if open fails

- **int fclose(FILE *f)**

finish operations on a file
fclose called automatically on program exit
Beware: some output may be cached until fclose called

- **int fgetc(FILE *f)**

Read a single character from a stream
returns EOF if no character available

- **int fputc(int char, FILE *f)**

Writes a single character to a stream

- **int fscanf(FILE *f, ...)**

scanf from stream; returns number of values read

- **int fprintf(FILE *f, ...)**

printf to specified stream

FILE type

- While doing file handling we often use FILE for declaring the pointer to point to the file.

```
FILE *fp1, *fp2;
```

- A FILE is a type of structure typedef as FILE.
- We only use pointer to the type and library knows the internal of the type and can use the data.

Opening a File

```c
FILE *fp = fopen("filename.txt", "w");
```

- fopen - opens a file
- parameter 1 - the name of the file to be opened
- parameter 2 - the mode in which to open the file
- return value - a pointer to the file which has been opened.

  This pointer is then used to reference the opened file for operations such as reading, writing, and closing of the file. Return value will be NULL if file can not be opened.

fopen mode parameter

- “r”
  - Opens an existing text file for reading purpose.
- “w”
  - Opens a text file for writing.
  - If it does not exist, then a new file is created.
  - Starts writing from the beginning of the file.
- “a”
  - Opens a text file for writing in appending mode.
  - If it does not exist, then a new file is created.
  - Starts writing at the end of the existing file contents.
- “r+" “w+” “a+”
  - Opens a file for both reading and writing.
  - w+ truncates the file length to zero if it exists, while a+ starts writing at the end of the existing file contents.

Reading from a File

```
char line[MAX_LINE_LENGTH];
if (fgets(line, MAX_LINE_LENGTH, fp) != NULL) {
    printf("read line\n");
} else {
    printf("Could not read a line\n");
}
```

or

```
int c;
c = fgetc(fp);
if (c != EOF) {
    printf("read a '%d'\n", c);
} else {
    printf("Could not read a character\n");
}
```

Writing to a File

```c
fputs("the text I want to write in the file\n", fp);
```

or

```c
fprintf(fp, "the text I want to write in the file\n");
```

or

```c
int c = '!';
fputc(c, fp);
```

Binary Files

- Text and binary files both contain data stored as a sequence of bits
- The bits in text files represent characters
- The bits in binary files represent custom data in specific formats, that can be created or edited with custom software.
- You can view and edit them in hexidecimal editors such as ghex. This is painful!
- Examples are jpg, pdf, executables, mp4, bmp etc
- If you are able to open them in a text editor, you will see lots of garbage!
- Demo: bmp files

fopen mode parameter for binary files

To handle binary files, we will use following access modes instead of the previous mentioned ones:

"rb", "wb", "ab",
"rb+", "r+b", "wb+", "w+b", "ab+", "a+b"

Read and write binary files

```
fread( ptr, int size, int n, FILE *fp );
fwrite( ptr, int size, int n, FILE *fp );
```

ptr − This is the pointer to the element (such as array) to be written.

size − This is the size in bytes of each element to be written.

n − This is the number of elements, each one with a size of **size** bytes.

fp − This is the pointer to a FILE object that specifies a stream.

```
fread(&i, sizeof(int), 1, file);
fwrite(&b, sizeof(unsigned char), 1, file);
```

Questions