File I/O

Comp-206 : Introduction to Software Systems
Lecture 16

Alexandre Denault
Computer Science
McGill University
Fall 2006

Take Note

Class on Thursday, November 9th, is canceled
Assignment 2 is due Tuesday, November 14th.
The paper document for Assignment 2 is due
Wednesday, November 15th.

Files in Unix

- As previously mentioned, in Unix, everything is a file.
  - STDIN, STDOUT and STDERR behave like files.
  - Peripherals connected to the computer are also behave like files.
  - Network communication sockets behave like files.
- As such, if you know how to manipulate files, you can manipulate STDIO, peripherals and sockets.

Types of files

- Text or ASCII : These files are used to store text using the ASCII character encoding. Each byte of the file represents a character.
  - Special control characters are used to represent an end of line (015) or an end of file (003).
  - There is a total of 256 different characters in an Ascii file.
- Binary : All files that use an encoding scheme other that ASCII are considered binary. The include image files, music files and PDF documents.
  - Each byte has a special meaning for that file.
  - The meaning of that byte is defined by the encoding format.

stdin, stdout, strerr

- When a C application is started, three files are opened by the operating systems.
  - stdin, stdout and stderr
- The corresponding file pointers are declare in <stdio.h>.
- You can normally use stdin to read from the keyboard and stdout to print to the console.
- However, stdin and stdout can be redirected to/from files, as we saw during the shell programming courses.

Formated Output - printf

- `printf` takes a variable number of arguments, the first being a format string.
- The function returns the number of character it printed.
- The format string contains the string to output with variable tags.
  - For example : printf(“The temperature is %d. \n”, temperature);
  - Variable tags are denoted by a percent sign % and a code.
  - In this case, %d is use to indicate an integer.
  - The second argument will replace the first tag.
  - If a second tag was used, it would be replaced by the third argument.
  - The string is terminated by \n. This is a newline character.

Printf Conversion

- %d or %i : signed integer
- %x : unsigned hexadecimal integer
- %u : unsigned decimal integer
- %c : unsigned char
- %s : char* (string)
- %f : float or double of the form [-]mmm.ddd
- *%m.d*f : float or double of the form [-]mmm.ddd where m and d specifies the maximum number of digits.
- %E : double of the form [-]m.dddExx

Formated Input - Scanf

- Scanf is printf's analog, providing functionality to read formated input.
- The format string uses the same convention as printf.
- One important difference is that scanf expects pointers as the arguments.
- The pointer should indicate where the input should be stored.
- Scanf will read from STDIN until it matches every token in the format string, or until it hits an error (or an incorrect conversion).
- Scanf will return the number of characters read.

Dangers of Scanf

- Scanf is a tricky function to use because it assumes the input matches the format string.
- For example, then using scanf with the %s, you must assume that the string will fit inside the supplied character array.
  - That is not always the case.
  - Vulnerable to buffer overflow.
- You can control the maximum number of character scanned using the %ns option.
  - For example, %40s will only read the first 40 characters.

Controlled Input

- Most veteran C programmer suggest the use of fgets to read in input.

  char *fgets(char *s, int size, FILE *stream);

- This function allows a programmer to read in a string and a controlled number of characters (or until it hits a newline or EOF).
- The programmer must then manually parse the string himself.

Character input

- The simplest input function is getchar.

```
int getchar(void)
```

- It retrieves one character from STDIN.

Flushing StdIn

- Like all files, StdIn is a buffer.
- Functions like scanf, getchar and fgets will only read parts of a buffer.
- If you want to discard the content of a buffer, you'll need to do so manually.
  - Never use fflush() on an input stream.

```
void flushStdIn() {

  char c = 'a';

  while(c != '\n') {
    c = getchar();
  }
}
```

Sequential vs Random Access

- Normally, access to a file is sequential.
  - You open a file and you read from start to finish, in that order.
- However, you might also need to jump around in the file.
  - fseek allows you to change the position of the file position indicator.
  - ftell returns the position of the file position indicator.
  - rewind sets the file position indicator at the beginning of the file.

Opening a file

- You can open a file using the fopen function.
- FILE* fopen(const char* file, const char* mode)
- Once a file is opened, it returns a FILE pointer. That pointer can then be used to modify the file.
  - The file pointer points to a structure that contains information about the file, such as
    - Location of buffer
    - Current character position in buffer
    - Open mode
    - Any errors that might have occured.
- The “mode” indicates what type of access is required.
- In case of error, fopen returns null.

Open mode

- The mode is specified by a single character:
    - r : opens a file in read mode
    - w : opens a file in write mode
    - a : opens a file in append mode
- If a non-existing file is opened in write or append mode, it is first created.
- If an existing file is opened in write mode, it's original content is discard.
- To open a file in binary mode, a “b” should be appended to the mode string.

Character IO

- The simplest file IO functions are getc and putc.
  - int getc(FILE *fp)
  - int putc(int c, FILE *fp)
- The getc function reads a single character from the supplied file pointer.
- The putc function writes a single character to the supplied file pointer.

Formated File IO

- The fprintf and fscanf functions correspond to their printf and scanf counterpart.
  - int fscanf(FILE *fp, char *format, ...)
  - Int fprintf(FILE *fp, char *format, ...)
- The only difference is that fscanf and fprintf explicitly require a file pointer as their first parameter.
  - printf assumes output should go to stdout
  - scanf assumes that input should come stdin

Line IO

- The fgets and fputs functions can be used to manipulate lines of IO.
  - char* fgets(char *line, int maxline, FILE *fp)
  - Int fputs(char *line, file *fp);
- A line of text is defined as a character of array termined with either an end-of-line character or a null character,
- The fgets function reads the next line and stores in the provided character array.
  - Note that the function does not create a character array.
  - Before calling this function, you should have allocated a memory space large enough to review the input.
  - At most maxline characters are read.
- The fputs function writes a new string into the specified file.

Closing a file

- Once you've finished with a file, you should use the fclose function to close the file.

```
int fclose(FILE *fp);
```

- This will write any data that might have remained in the buffer.
- This is particularly a good idea if you open and close files often in your application.
  - A process can only open a specific number of files at a time.

String IO

```
int sprintf(char *str, const char *format, ...)
int sscanf(const char *str, const char *format, ...);
```

- You can also use the printf and scanf functions on strings.
- With sprintf, you can concatenate multiple values to create a new string.
  - You can use snprintf if you want to control the maximum size of the output string.
- With sscanf, you can parse an existing string.
  - Note that sscanf has the same dangers of scanf and fscanf.

Data Storage Strategies

- When storing data, the first step is to determine if storage should be in text or binary.
  - Text : ideal if the content if made out of only ASCII characters.
  - Binary : for everything else.
- Once you have decided on the encoding, you need to decide on the storage format :
  - Text : Comma Delimited, XML, etc
  - Binary : Buffers with length, etc
- Each of them has strengths and weaknesses.

Comma Delimited

- In this format, separate records are stored on different lines.
- The fields of the record are separated by comma's (or semi-colons, or whatever control character you choose).
- Great care must be taken to make sure that the control character is not found in the data set.

```
Lotr;Tolkien;300
Harry Potter;JK Rowing;240
```

XML

- XML (eXtensible Markup Language) is a W3C-recommended general-purpose markup language that supports a wide variety of applications.
- It's a hierarchical storage format that is easy to parse and where the content can easily be transformed into a tree.

```
<book>
   <title>Lotr</title>
   <author>Tolkien</author>
   <page>300</page>
</book>
```

Happy Halloween