CSCI 4100
Assignment 2
Writing a Linux Utility Program

Learning Outcomes

Write a Linux utility program.

Required Reading

None, but you may find the links to the C language and C library documentation helpful.

Instructions

For this programming assignment you are going to implement a simple C version of the UNIX `cat` program called `kitten`. The `cat` program allows you to display the contents of one or more text files. The `kitten` program will only display one file. The correct usage of your program should be to execute it on the command line with a single command line argument consisting of the name you want to display. However, your program should also respond well when it is used incorrectly.

Processing Command-Line Arguments

Unless you have done Linux programming before you probably haven't needed to process command line arguments. A typical C program has a `main` function that looks like this:

```
int main()
{
  // body of main function
}
```

This works just fine if your program does not take any command line arguments. If it does, as is the case with `kitten`, you will need access to those arguments. You will need to write your `main` function like this:

```
int main(int argc, char *argv[])
{
  // body of main function
}
```

The argc parameter is the number of command line arguments provided to the program including the name of the command itself. The kitten program should have two command line arguments if it is used correctly: the name of the command and the file to display. The argv parameter is an array of C-strings, or null-terminated character arrays. This means that argv[0] is the name of the program, argv[1] is the first command line argument, argv[2] is the second command line argument, and so on.

The kitten program should display a usage message to standard error and exit the program using exit(1) if argc is anything other than 2. Otherwise it should use the C-string contained in argv[1] as the name of the input file to open. Note that exit function requires the following preprocessor statement:

```
#include <stdlib.h>
```

Streams

The standard way to deal with files, the console, and other sources of input and output is by using **streams**. For historical reasons, the data type used to deal with a stream, whether or not it uses a file, is FILE *. Working with streams requires the following preprocessor statement:

```
#include <stdio.h>
```

To declare a stream variable use the FILE * data type:

```
FILE *stream;
```

To open a file use the fopen function:

```
stream = fopen(filename, opentype);
```

- fopen has two parameters: filename is the name of the input file as a C-string, and opentype is a C-string containing information about how the file is to be opened.
- If the file is to be opened for reading only, the second argument should be "r".
- fopen returns a value of type FILE * if the file opened successfully, and NULL otherwise.
- If your program can not open the file, display a message that the file was not found to standard error and exit the program.

To read a single character from a file, use the fgetc function:

```
character = fgetc(stream);
```

- fgetc has one parameter: the stream that was returned by fopen.
- fgetc returns the character read if it was successful and the special EOF character if it was not successful.

To close a file use the fclose function:

```
fclose(stream);
```

- fclose has one parameter: the stream representing the file to be closed.
- fclose returns 0 if the file closed correctly and EOF if it did not. The latter case is rare, so the return value is typically ignored.

To write a string to standard output use the `puts` function:

```
puts(string);
```

- `puts` has one parameter: the C-string to be printed. Note that if you want a newline to be displayed you have to use the special character `\n` at the end of the string.
- `puts` returns a non-negative value if successful and `EOF` if unsuccessful. The return value is typically ignored.
- You will not need this function for this assignment, but it may be helpful to you for debugging purposes.

To write a string to standard error use the `fputs` function:

```
fputs(string, stream);
```

- `fputs` has two parameters: `string` is the C-string to be printed, and `stream` is the stream representing the destination of the output.
- To print to standard error, use `stderr` as the second argument.
- `fputs` returns a non-negative value if successful and `EOF` if unsuccessful. The return value is typically ignored.

To write a single character to the console use the `putchar` function:

```
putchar(character);
```

- `putchar` takes a single argument: the character to be printed.
- `putchar` returns the character if it printed successfully and `EOF` if it does not. The return value is typically ignored.

Linux Development Tools

You should not be using Windows development tools for this class! Instead, you should use the development tools provided on the Linux server.

Writing Your Code

The best way to write a Linux program is to use one of the many text editors provided on a typical Linux distribution. If you are using the Linux server you have several options, but the `nano` text editor mentioned in Assignment 1 is probably the simplest. If you want to use a text editor with more features for writing source code you can try using `vim` (see Chapter 6 of the Linux book) or `emacs` (see Chapter 7 of the Linux book.)

Compiling Your Code

Your source file should be called `yourlastnameAssign2.c` except with your actual last name. To compile this code you should use the `gcc` compiler on the Linux server. To create an executable file called `kitten` use the following command:

```
gcc -o kitten yourlastnameAssign2.c
```

If your program compiled, you should see an entry for `kitten` when you execute the `ls` command.

Running Your Code

Since your home directory on the Linux server is not in your execution path, you will need to specify the file you are executing directly by putting a `./` before the name of the executable. Here are some examples of what several runs should look like:

```
$ ./kitten foo.txt
This is a text file that I created
in a text editor in order to test
out the kitten program.

$ ./kitten
usage: kitten <filename>

$ ./kitten foo.txt otherFile.txt
usage: kitten <filename>

$ ./kitten no_such_file.txt
error: file not found
```

What to Hand In

Download the source file `yourlastnameAssign2.c` to your local machine, then upload it to D2L in the dropbox called Assignment 2.