Timothy Hinds

P2 Report

I followed the structure given in the P2 manual, so first I created read_file.cpp which would return a string that contains the contents of a file. For this I needed to utilize the open() POSIX call. I passed to open() the parameters filename and O_RDONLY, what this system call will do is open the file 'filename' under read only mode. It then returns the file descriptor value. Once I have the file descriptor value 'fd', I call fstat() to acquire information about the file 'fd'. Following this I can use the filestats acquired from fstat() to create a dynamic buffer the size of the file using malloc(filestat.st_size). After this is completed I now have char *buff which is the size of the file. Then I use ifstream to fill the buffer with every character in the file by using the myfile.get() function. I then close the file and return the buffer which contains the information in the file.

Once read_file.cpp is executing correctly, I then created displayfile.cpp which would utilize this function to display it in the cmd. It is as simple as printing the char *buff until it reaches the end of the buffer which is indicated by a '\0' character, then it frees the buffer and finishes. After compiling, a simple call such as './displayfile /home/reptilian/example.txt' will print the contents of example.txt to the console.

Creating the GUI required me to play around with the activity_p2.xml file to acquaint myself with the button and text view component ID's. I decided to incorporate my read_file.cpp into the android directory. I copy pasted it into the cpp folder, then created its own library in the CMakeLists.txt file. I left it the exact same as when I made it in reptilian. I decided to incorporate the given JNI function in native-calls.cpp and modify it to essentially run the same as displayfile.cpp does in the cmd. I added a jstring parameter to the function, and used JNI methods to manipulate the input string to be changed to a c-style string. Then I used this c-style string to call the read_file() function which returns another c-style string. Then I create a string from this ctyle string, and return it.

Now I moved back over to P2Activity.java to finish the GUI implementation. At the bottom of this function I changed the stringFromJNI() to now have a (String s) parameter. Then created the method submitButtonClick() which will execute the C functions when the button is clicked. It calls the stringFromJNI() function with the filename parameter taken from the textbox. I then set the text of the displayBox TextView to be the string received from the stringFromJNI() function.

I spent most of my time trial and erroring with the android components because I ran into a problem where when I clicked the button the program would just close. I found that the problem was being caused by me not correctly changing the displayBox's textview. I also found a lot of helpful commands for JNI implementation from the following website.:

http://www.ntu.edu.sg/home/ehchua/programming/java/javanativeinterface.html

Using fstat: https://stackoverflow.com/questions/238603/how-can-i-get-a-files-size-in-c