Eemil Aspholm

CT30A3370 Käyttöjärjestelmät ja systeemiohjelmointi

Harjoitustyöprojektit

**Project 1: Warmup to C and Unix programming**

I started the project by building a simple menu based main function. The menu has three different options 1. Reading user input from terminal and printing the input into the terminal in reverse order. 2. Reading user input from file and printing the input into the terminal in reverse order and 3. Reading the user input and printing the input into an output file. The program in overall was easy and I had no problems while programming. Some of the functionalities were similar to ones in C-programming basics course’s project work, so I was able to use previously created code. The functionality pictures are shown below:

First, testing the function without any command line arguments, the program takes input until a blank line is inserted. After this the program reverses and print it to terminal. The second test is with only applying an input file testi.txt. The program reverses the file to terminal. The last part is running the program with an input and output files. The output file is also shown below.

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

**Project 2: Unix Utilities**

This project was more challenging than the previous one and required quite a lot of time. In the project I used the xv6 terminal functions as a guideline for my own functions. I started this project with the my-cat function which was the easiest one to do. After that I continued to my-grep and lastly to my-zip and my-unzip functions. The functionalities of the programs are shown below in pictures.

Starting with he my-cat.c program, first I created a loop that will go through all the command line arguments, second, I added a while clause that will loop through the file and print the file into the terminal. Last I did some error handling and answered all the assumptions the assignment had.

My-cat is tested without any command line arguments (doesn’t work), with 1 command line argument and with 2 arguments. The program also has error handling for faulty input files.

Text

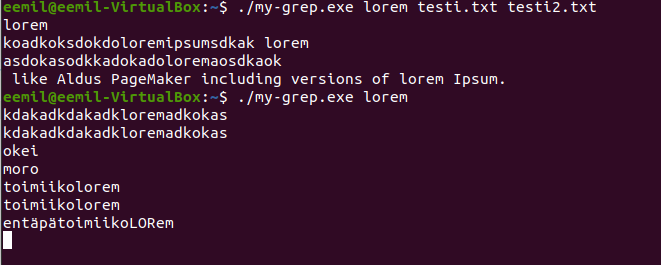
Description automatically generated

My-grep on the other hand was a little harder than my-cat. First, I created a switch case to determine the amount of command line parameters, the second I created the functionality for 1 parameter, where the programs read input from terminal. After it was done, I continued to the default case with two or more parameters, where the program reads files and tries to find certain word in it. The functionality was pretty much applied from my-cat program since they were doing pretty much the same thing with a little difference.

Functionality is shown below:

Text

Description automatically generated



Last I programmed the my-zip and my-unzip programs. These were the hardest ones yet. I started this part of the project with programming the my-zip first. I used the same kind base in this project as I had in the previous, error handling and looping through the command line parameters. I also find a good source for this and by altering the code I was able to make to function to work. The program itself reads the file character by character, and test if the next character in line is same as the previously read char. If it were adding +1 to the count parameter and continue to the next character. When the character changes, were writing the count and the character itself into the output file. This will continue if there are characters in the input file.

The my-unzip was a lot easier compared to my-zip. I used the same kind of basics to this one (loops to go through parameters, and error handling.) This time the program had to read the count number from the file and translate it back to letters. First, I checked that there are elements in the input file, the second I used freed to transfer the next char in line into a buffer. After this I only had to keep looping and writing the number to terminal until the character changed.

Pictures above of the functionalities of the programs. (I couldn’t open the zip.z file with text editor to show the insides)

Text

Description automatically generated

A picture containing graphical user interface

Description automatically generated

**Project 4: Kernel Hacking**

I chose Kernel hacking project because it seemed interesting to me, although I had no idea of how hard it could be. First, I watched the intro YouTube video about xv6. After watching the video I had to google how to install xv6 into Ubuntu that is ran through virtual box, this wasn’t that hard at all.

Graphical user interface, text, application

Description automatically generated

For this project I also found useful guide from GitHub: <https://gist.github.com/bridgesign/e932115f1d58c7e763e6e443500c6561>

First, I had to start study how the xv6 works and what I would have to do to make own command line function. After that I read the step-by-step guide from bridgesign’s github and made my own implementation of the counter. The guide taught me a lot about xv6, and in my opinion I have a lot better thought how the whole xv6 is built. After I had finished the counter, I tested that it works, and implemented a command line function getCount that returns the count of the calls made. After this I started working on the functionality of the counter reset that is included in command line function modifyCounter. Modifycounter will take a number as an input (the number of syscall from 1 to 24 that is wanted to be counted.) and it will also reset the counter. Unfortunately, I was not able to make the setCounter to work (it did not edit the proc structs callID element, I’m not sure why.) But otherwise, the counter resetting and receiving the number of systems counts to the terminal works fine.

Text

Description automatically generated

In this particular project I had to edit almost every single of the files included in xv6, but the most editing was made in sysproc.c, Makefile, syscall.c and adding also the getCount.c and modifyCounter.c functions in the xv6 folder.