Hayden Kendall
Prof. Jean-Yves Hervé
CSC 412: Operating Systems and Networks
14 June 2020

Programming Assignment Three

This third programming assignment is centered around the concept of parallel processing. Computations at a simultaneous manner is an imperative practice in the world of operating systems. In the book *Operating Systems Concepts and Applications* by Donald R. Horner, it is described as the true value behind multiple processor systems. By using this fundamental parallelism, jobs are completed much quicker as opposed to single processor systems that rapidly cycle between them. It has allowed for more efficiency and optimization in system architectures.

The program presented takes a pattern search problem provided in the course and asks us to create a C program to complete the task. From that point on two versions of the program are made. One where it is a single executable program. It internally will fork for any number of image files found in a directory and passed as an argument. The second version will be a main program that manages and generates child processes as a separate search program entirely to find patterns. This is done using a call to a member of the exec family as well as some fork calls.

In this project I found three major algorithmic and data structure decisions that needed to be made. The first, while I consider it to be a learning curve of the C language, is the use of dynamically allocating pointers with malloc. The malloc function is intended to give the programmer some manual memory management. Good programming practice as a whole is to avoid hard coding numbers where possible. Malloc is good because it allows us to dynamically allocate memory blocks. The sizes cannot change once an array or pointer is initialized when not using malloc.

When it came down to deciding the pattern search, I felt it best to read the entire image file as a one dimensional array and do a linear-style search on the string. To elaborate, a lot of image processing in modern technology does take a linear approach in a way that visits each pixel, reads the data, and does some computation. In this case, I read each individual character to see if it matches with the first character of the pattern. If this is found to be true, a “look-ahead” is run to identify a match. This is completed by reading the first row and counting the width of the pattern. If I get to three matches in the context of these patterns, the algorithm jumps to the next row in the image by an addition of the image width. A sub-index will jump to a later, calculated part of the array to check the subsequent row. Once we count the threshold of matching characters we determine a match is found. My reasons for this design decision is that it will generally be slightly computationally optimized to keep processes on a single loop. The other is for overall simplicity in C. Not to say that two dimensional were actively avoided in this assignment, this learning objective was also completed in the use of gathering all pattern file names.

My final design decision came as a result of my stubbornness and ultimately probably wasted more time than it was worth. Originally, I was really trying to find a way to maintain a

pointer data structure that needed to be dynamic, reallocatable, and easy to traverse. In the native C language this is a pipe dream to be built in. By the end of my research of trial and error, I decided to use a linked-list to maintain all occurrences in the pattern search. This is because they are simple to traverse and the head address can easily be passed as arguments to other functions. Generally the linked-list of occurrences was always going to be traversed linearly. It seemed like linked-lists were the right decision to make for the nature of the program.

The C language is never to be underestimated as an easy to pick up language for professional level programming. Though through assignment I've seemed to develop an appreciation and almost preference for being this intimate at the hardware level. It comes with its challenges. Being a programmer who hasn't used pointers since before this course, I had a steep climb to not only conceptually understand what they were but also to execute their properties in an advanced way. Memory allocation was also a great difficulty in this assignment. Having failed most of the programming assignments at the time of the course, allocation and deallocation was foreign to me. Memory leaks were rampant throughout this project. Through the difficulties, I now have proficient understanding of pointers and their necessary value in the C language. In addition to that, I now have good practice with memory management. Through many hours of tracing, I can happily say this program should have no memory leaks.

Partly out of frustration, but also with forethought, I decided to refrain from pursuing the extra credit portion this time around. This was to use the program on any size of images and patterns to search. I think with a better understanding of C now, and what I have to learn in the near future, I'll have the knowledge to return to this project and see it fully completed. As of writing this, the program can only do pattern searches on the provided dimensions. I think there's a decent chance of data overloads and constraints in this program. For example, the compiled output directory string has a limitation in array size and excessive amounts of subdirectories could cause the array to overload. There may also be some inefficiencies with pointer usage as C is still in an infantile learning stage for me.

Overall, the assignment was challenging and required an intense amount of focus. It requires a willingness to research and self-learn a language that can easily break apart and seem intimidating compared to more modern languages. While difficult, the benefits were numerous. In just one assignment I learned many fundamental concepts in computer science and operating systems that I know will be remembered and used in the near future.