Mahfuz Uddin

CSCI 340 - OS

3/30/2020

Dr. John Svadlenka

A write-up of my software design and an explanation of the issues I encountered and resolved.

During the span of my time working on this project I encountered many issues and bugs across the board, which took more time to actually solve than actually writing the code. This project implements a parent and child process that splits work with each other. Using fork, we created a child process, which will consume text from shared memory and write to the target file. The parent produces text into memory from a source file. For this to be possible, I needed to create a piece of shared memory that the processes can read and write to. I also created two other pieces of shared memory for our *IN* and *OUT* variables which both the processes need to access in order to write to memory and read from memory. For the parent Process to write to memory, we need to use pointer manipulation. Using the base pointer to our main shared memory, *CHUNK_SIZE* and *IN* variable, we can find where in memory we have to write to. Before this, we need a local parent buffer of size *CHUNK_SIZE* to store the text from the source file. We read from the source file in *CHUNK_SIZE*, store it into a local parent buffer, and from the buffer, we write to memory by copying each character in our local buffer into memory using loops. For the child to consume text from memory, a similar process occurs using pointer manipulation, *CHUNK_SIZE*, and *OUT* variables. Multiplying *CHUNK_SIZE, OUT*, the size of a char, and adding it to the base pointer of the main shared memory we can find the starting location from where we have to consume. Using loops, we consume text of *CHUNK_SIZE* into a local child buffer. From the buffer, we write to our target file using *WRITE*. In our child process, we also check for a null terminator in memory, a signal that we have finished consuming from memory. This null character is passed into memory by the producer after it has finished producing into memory. When the parent and child are respectively producing and consuming, they also write to the terminal using *WRITE* of what they produced and consumed as well as the *IN* and *OUT* variables they used.

Before starting the project, I took some time to understand our example code from the textbook as well as the code you provided as examples. One issue I had when I started the project was creating shared memory between parent and child. For some reason, it kept failing. Tinkering with the code, I discovered that the *name* parameter passed into shm_open was the problem. The name was "OS", which I believe already existed in the computer memory hence, making it fail. Initially, I thought just declaring *IN* and *OUT* variables would be enough, but I soon discovered that after doing a fork, the processes create their own copy of each variable, they don’t share it. Struggling through, I discovered that I needed to create shared memory for both IN and *OUT* variables, so both the processes can share it. One other problem I had was accessing the value of *IN* because I had the function call *truncate* after mapping it, so I had to move the truncate function call before mapping to the shared memory. Another problem I had was printing the output to the screen without messing up the format. At first, I was writing out to terminal the *ITEM* and *IN/OUT* independently of each other causing it to make the output messy. To solve this, I created a big buffer for the child process and copied “CHILD: OUT =” into it, then copied in the value of the IN variable, then “CHILD: OUT =” and finally copied into the buffer what the child consumed during that iteration. After that, I outputted the big buffer into the terminal. I did similar processes for the parent's output. This prevented improper output and outputted things in order. When printing to the terminal, there were weird characters being printed, using *PRINTF* throughout the code to trace all the values variables, I discovered that I passed the wrong parameter for *WRITE*, which led to weird characters being printed to the terminal.

This project was definitely on the difficult side of the spectrum. I struggled every step of the way, had to learn new things and adapt. Over the span of a few days, long hours I was able to make my program work. At the end of the day, I learned so much from doing this project.