**Rutgers ECE 434, Spring 2018, Prof. Maria Striki**

**Project 1: Inter Process Communication**

**Report**

Brendan Li, Brett Lechner, Bryan Benalcazar, Leonardo Roman

**Problem 0 (Bryan Benalcazar)**

Process Structure:

We used an Inter-Process Communication that is a recursive function, which is named processIPC(int StartOfProcess, maxminsumcount \*new\_info). Maxminsumcount is a structure that is created that contains the maximum, minimum, sum, and count. This function also has access to the number of integers in the file and the amount of Processes that is responsible for a specific number of integers.

In the program, we used a pipe that enables communication from one process to another. This is done when the child process needs to send the information that is computed to the nth process. The nth process is responsible for collecting and merging all the information from the other processes. The child process is responsible for collecting the information it needs.

We also used shared memory as part of our process structure. The system calls that were used are shmget, shmat, shmdt, and shmctl. The shared memory helped us with memory and CPU usage. The shared memory helped us stored the integers from the file into the shared memory. In each process, there was a pointer to access this shared memory.

Design Decision Explanation:

The number of processes used by the program in Problem 0 is one of the input arguments, which is denoted as AmtofP. However, it seems best to use as many processes as there are CPU cores available because this helps with time. There should not be benefits from using multiple processes than the number of cores. Every process does the work that it needs to do. The 10 active processes in part B and C, which means they are waiting for the other child process to be completed, will saturate the process of a computer. This is also true for 20 active processes.

One thing that we have learned as a result of this project is that increasing the number of processes does not always improve the time of a program. Another concept that we learned is the idea of having multiple processes being able to spawn from another process. Overall, this problem made me learn more about C programming and how to implement a program that creates multiple processes, which is more difficult than it seems.

Time to Process List of Numbers (Microseconds):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # of Processes | Problem # | 10 | 100 | 1k | 10k | 100k |
| 1 | A | 785 | 5700 | 5945 | 7522 | 20386 |
| 5 | B | 6922 | 2671 | 2490 | 4081 | 15069 |
|  | C | 2524 | 2886 | 5687 | 8714 | 20936 |
| 10 | B | 3167 | 4079 | 4467 | 5737 | 16471 |
|  | C | 29483 | 29609 | 31797 | 49124 | 46571 |
| 20 | B | - | 11959 | 13263 | 13579 | 21928 |
|  | C | - | 254653 | 498178 | 551799 | 473465 |

**Problem 1: Building a given Process Tree (Leonardo Roman)**

Questions:

1. What happens if root process A is terminated prematurely, by issuing: kill -KILL <pid>?

The children roots, c , b and d will become horfan and become children of init.

1. What happens if you display the process tree with root getpid() instead of pid() in main()? Which other processes appear in the tree and why?

1. What is the maximum random tree you can generate? Why?

**Problem 2: Arbitrary process tree generation (Brett Lechner)**

For this assignment, I was unsure how to go about accomplishing this project. I decided to implement a program that reads the text file, and translates that information into a n-ary tree data structure. From the n-ary tree, the process tree is built and from there all the appropriate messages are printed out. The program is built to only allow one character names for nodes, so that is one very distinct attribute (or flaw if you like). As such, the maximum number of processes that can be created is the number of ASCII characters.

There are a number of functions in the program to accomplish all of this. I was able to make a recursive function to create the tree (both the n-ary, and process tree), a function to read the file, create the tree, and return a pointer to the root, and a recursive function to print the tree.

The appearance of the start messages runs through the tree in a Parent-Child fashion, so the first parent and its children are created, and then the second parent and its children are created. The termination messages vary in their order, so there is no distinguishable pattern of messages. However, the first parent to have their children all terminate will print its termination message sometimes within the terminating messages of other children. The termination messages tend to start with parents that has less children, and end with the parents that have the most children. Of course the ending message is the termination message of the root process.

Included are three text files, FileSmall.txt, FileMid.txt, FileBig.txt. FileSmall holds a small tree to see the process, FileMid contains a decent sized tree, and FileBig contains the near maximum size (26).

**Problem 3: Handling and Sending Signals (Brendan Li)**

For this problem I was tasked with creating a process tree with parents and children. Unlike Problem 2, where we used a sleep() function to wait for the child to finish it’s processes before we continued, this problem asked us to use signals to communicate between processes. The use of signals required me to create signal handlers so that I can more easily see which process is currently running. One exception to this is SIGSTOP, this is because SIGSTOP cannot be interrupted or ignored. In this example, I created a simple tree with four nodes to demonstrate signals.

Questions:

1. **We have used sleep() in the previous parts to synchronize processes. What is the advantage of the use of signals?**

With the use of signals, we have to create signal handlers that intercept and process the received signal. Whenever the parent forks a child, instead of using sleep(), we can use the signal SIGSTOP to pause the parent process. The process will then stop executing until it receives SIGCONT from it’s child. The advantage of this implementation is we don’t have to implement sleep() as long as we make sure that whenever the child process is completed, we send a signal back to the parent to continue it’s execution.

1. **What can be the role of your function wait\_for\_children()? What benefit does it ensure and what potential problem could its omission bring about?**

The function wait\_for\_children() is used to check the children PID’s with the use of getppid(). We can use the use the return value of waitpid() to determine the status of the children. With a return status of 0, we know that the child process is still running. If waitpid() returns a value greater than zero, we know that it is safe to terminate the parent process.

If we did not include a way to check if the children were stopped, we could accidently stop the parent process while a child was still executing some code. This could cause errors and possibly lead to corruption of data.

**Problem 4: Parallel calculation of numerical expression (Leonardo Roman).**

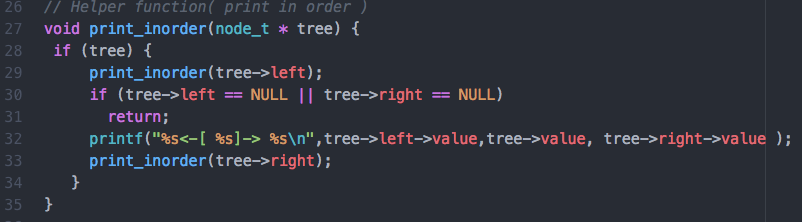
Question:

1. How many pipes per process do we need to use in this problem? Would it be possible for every parent process to use only one pipe for all the children processes? Or in general, can we use only one pipe for every numerical operand?

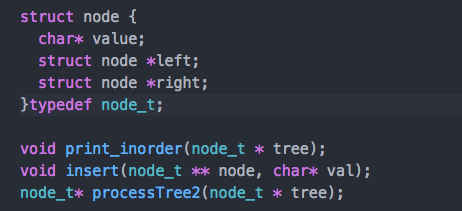
Answer:

We needed to use two different pipes in order to send left and right child of root node. As the root node is sent to the child process a recursive function call will be executed and send left or right child of current node. Hence this process will be initialized over again until a leaf node is found making the program traverse back the tree and in this case child will be sending or writing back to the respective parent. Since we are checking for two children per a root node, we need two pipes one for each child node. Moreover, in this problem I decided to simulate the final output. Final expected output was successfully acquired,10\*(5+6), via IPC. It was quite difficult to work and convert both characters to int and character to operation overloaded. I try everything to overload “\*” and “+” operators but the functions where else were in memory, reason why I decided just to simulate the final output as you can see in the figure below. Lastly, all the pieces necessary to complete this program were as follow: make a struck data structure for a tree node, helper function to traverse the tree in “inorder” traversal, finally the process tree function, to traverse the tree using parent and child processes using fork() syscall.

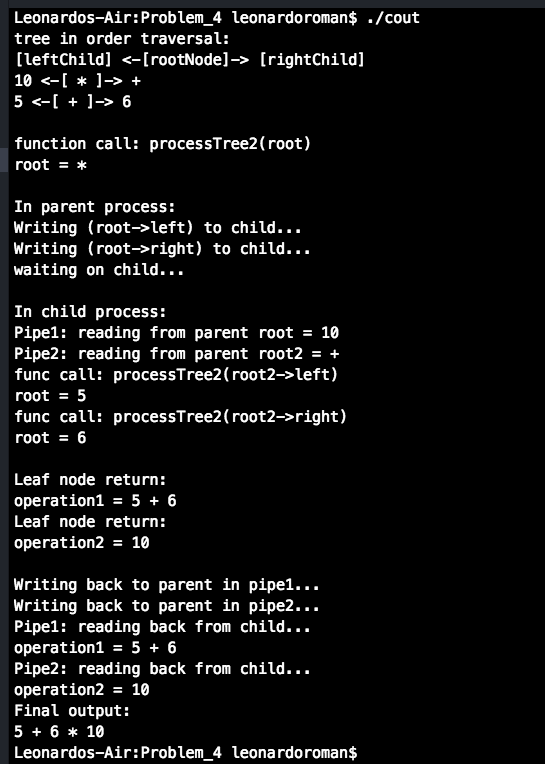
**Helper function to print tree:**



**Tree struct and other functions:**

****

**Final command line output:**

****