Isabelle Cochran and Emily Richardson

Introduction to Operating Systems, COMP 3500

Assignment 1 Part 3

**Part 1:**

2. (15 Points) Provide following:

1. (a)  Describe the parent/child process synchronization approach implemented in the

program.

In the parent/child process synchronization approach, we have allowed the child processes to access logical resources and initial input data. The initialization data, being the input file, is passed along from the parent process to its child. By using the fork() system call, we are able to create a child process that is able to inherit the open files from its parent. When fork() is implemented, zero is returned for the new child where a nonzero value is returned to the parent process.

W have implemented the parent/child synchronization in our main() function. Using a for-loop, the child process created by the fork() system call scans through each line in the file passed in, and the child has a variable that keeps count of the number of alphabetic characters. The child then places the information attained into its own memory with the UNIX command /bin/ls using execlp() system call. The parent is in a state of waiting as the child process is completing, and when completed, the information is shared with the parent since we used the fork() system call as previously stated.

1. (b)  Describe the inter-process communication (IPC) approach you have selected to

exchange data (both input and output) among parent and child processes. Provide

reasoning for your selection.

We have selected to use message passing as our IPC approach. We did this because you can declare and pass in a file, for us it was “file.txt” and then declare a file where the message will be passed and outputted for us “outfile.txt”. This allowed us to read through each line of the “file.txt” sort the data in it and determine the process id, line number, and letter count and pass it back out to “outfile.txt”.

1. (c)  Briefly describe a different inter-process communication approach that can be used in

this program. Provide reasoning for NOT implementing it in current program.

Another IPC that could have been used is pipes. This provides a one-way flow of data between related processes. We choose to use message passing because of it allows multiple processes to read and write data without being connected. We were able to create a vector to store the data and all processes were able to read that data.

While pipes are one of the simpler ways for allowing processes to communicate with one another, they do have some limitations. All ordinary pipes require a parent/child process; after the parent process and child process have communicated with each other, the ordinary pipes dissipate and do not communicate with each other again. This would have caused our child process to not repeat as needed when reading over the input file.

Another limitation of ordinary pipes is that they must only communicate between processes that are located on the same machine. While this did not directly affect our current program, as all of our processes are on the same machine, it could be a factor in modifications if we had to pull a different process from another machine

**Part 2**

2. (15 Points) Provide following:

1. (a)  Describe the parent/child thread synchronization approach implemented in the

program.

In the parent/child synchronization approach implemented in the program, we were able to share the parent’s operating resources with the child process. The thread is able to call the main function, and the parent thread creates its children processes and waits on them all to terminate before resuming. Once each of the child threads finishes its work, it joins its parents and the data is shared with the parent thread.

1. (b)  Describe the approach you have selected to exchange data (both input and output)

among parent and child threads. Provide reasoning for your selection.

In order to exchange data, the system implemented the Pthread specification approach. After creating the thread identifier and the attributes of the thread, another separate thread is created using the pthread\_create() function call. This separate thread will be used to carry out the sorting function and get the alphabetic letters that we desire from the parent’s open input file.

After initialization of the main() function, the program then has the first thread, being the parent head, and the child thread is created; the child thread then executes the function and performs the line scan to gather the number of alphabetic letters from the input. The thread identifier, attributes of the thread, and the name of the function where the next thread will start are all passed on to the parent if more children need to be created. As the child thread is called, the parent thread waits to terminate using the pthread\_join() function call, and it waits until the child thread finishes its work and gather the appropriate line number and letter count, and ultimately the character total. The pthread\_join() function call allows us to join the threads from the children to be passed to the parent in order to make our output file contain the correct line number, letter count, and character total.

**Part 3:**

1. Compare and contrast the two implementations: Part – 1 (process-based) and Part – 2 (thread-based). Is one approach (always) preferable over other? Provide reasoning for your views.

Using threads is the preferable approach for the implementation of this program. This is because the processes-based implementation is unable to share resources and data, has high resource usage, high context switch time and has scalability issues. These problems can be solved using threads. Threads implement the same program in a more efficient way. The only time when a process-based implementation would be better is if you are trying to run a plethora of threads at the same time.