Assignment Report

Operating Systems

Sahas Gunasekara

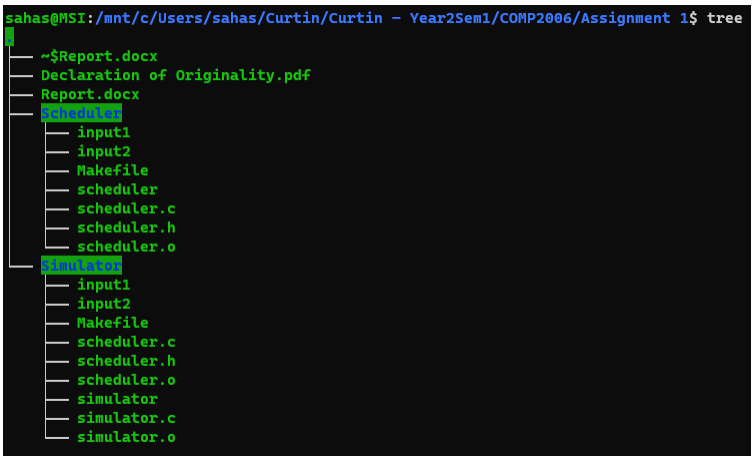
20462075

07/05/2022

1. Signed Cover Page

A signed cover page is included within the directory labelled as Declaration of Originality.

2. Usage Instructions (README)



As you can see, the file structure is as follows.

The code for Part A is inside the Scheduler folder and the code for Part B is inside the Simulator folder.

To compile each part, we have individual Make files which can be run using the command “make”. Usage of the “make clean” command will remove all the executable and object files.

Speaking of the executable files. To run the program for Part A you should run ./scheduler which will open the program. I have assumed that if you give a wrong file name it should shut down which I have done so. Else, the program will keep on asking for new inputs from the user until the keyword QUIT is provided.

To run the program for Part B you should also go to the Simulator folder and run the command “make” which allows you to run an executable by the name ./simulator. I have used a Header file for a slightly modified Schedular file (which contains the disk scheduling algorithms and no main function). Here 6 child processes are created when you run the program for the first time and will only be destroyed when the term QUIT is entered. Even if you enter a wrong file name the program won’t terminate but display an error message and ask for another name.

3. Discussion on Mutual Exclusion and Shared Resources

There are three statuses that are being shared by all threads, fileRead which signals whether a file has been read or not, written which signals to the threads whether the buffer2 contains a seek value that has not been read by the parent thread and nextLoop which signals if all the variables have been reset and the child processors can be looped again.

buffer1 is a specification resource that contains the requested locations, total cylinder amount, current Header Location, and previous Header Location.

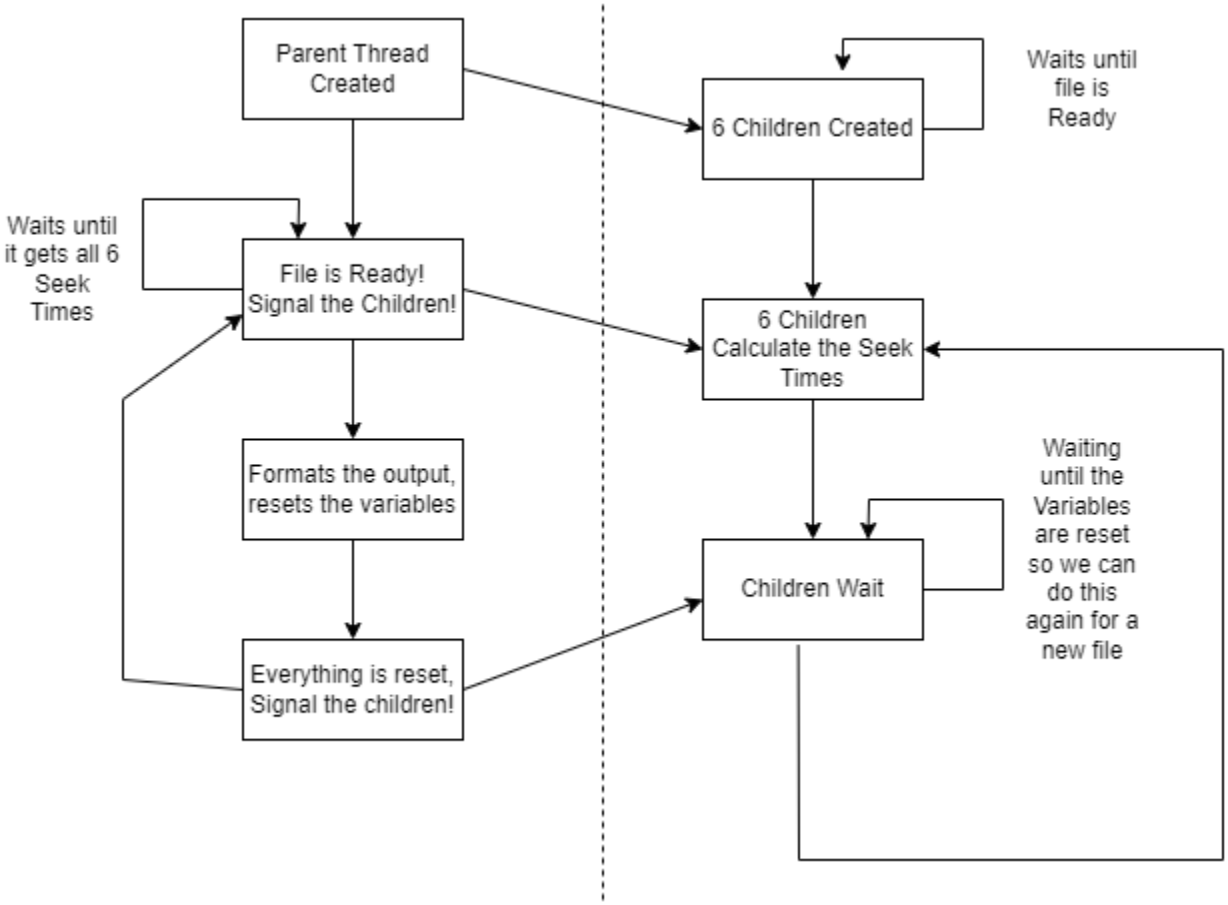
buffer2 is a specification resource that is of size 1 and only contains a seek time. returnAlgo is a shared variable that contains an integer with correlates to which algorithm's seek value is currently stored inside the buffer2 (I would have ideally had a buffer2 of size 2 and contain an index and a seek much like a Map however the specification demanded that the buffe2 be only of size 1. Hence, I created another shared variable since the pthread has no idea which seek value it might be).

totalWritten is an incremental shared variable that the threads will increment everytime they write to the buffer2 and the parent thread has read the buffer2. This is there to ensure that the parent thread only loops once all the child threads are done writing to the buffer2 and we have fetched them. Alternatively, could have used pthread_join (assuming the processes exit but they don't), however we used signals and mutex's.

To aid in mutual exclusion, we have Mutex's and Condition's that we use within our child threads and parent thread to wait and solve synchronization issues. Not all the shared variables are changed always within a pthread_mutex_lock and pthread_mutex_unlock. For example, we reset the fileRead, written and totalWritten variables without a pthread_mutex_lock after each parent thread loop but that's because we know for a fact that all our 6 child threads are currently inside a while loop waiting on a different condition (waiting on the nextLoop variable to be exact).

Although not a shared variable I would also like to mention how I cancel my threads. I have set the children threads to PTHREAD_CANCEL_ASYNCHRONOUS and use a cleanUpHandler. I could have also alternatively exclusively use a Mutex, Condition and Signal whether to cancel or not and have an if() statement within the thread to print out its ID and pthread_exit(). However, since the specification detailing the cancellation methodology (Part B/2/b) did not specify only to use pthread_create(), pthread_mutex_lock(), pthread_mutex_unlock(), pthread_cond_wait(), and pthread_cond_signal() unlike Part C, I decided to take a different method to showcase understanding and knowledge of pthread libraries. I have synthesized this information from the following official documentation https://www.ibm.com/docs/en/i/7.2?topic=ssw_ibm_i_72/apis/users_45.html. This is an assumption that I have made. However, it does not affect final functionality.

To visualize how it looks I have drawn a diagram.



4. Description on any cases where program might not work correctly

Gladly as far as I know there are no known instances of my program not working correctly. The CPU that I have tested this program on (Both Simulator and Schedular) is an 8 Core 16 Thread AMD Processor and all the synchronization lines up perfectly. I use the pthread_cond_broadcast() in some areas as opposed to pthread_cond_signal() as pthread_cond_signal does not work properly with all 6 threads in some conditions. I have come across instances where only one or two child threads have woken up and pthread_cond_broadcast() broacasts to all threads who are waiting on a condition whereas pthread_cond_signal() will only do at least one according to the documentation.

A limitation of my program is that it only accommodates a 1000 char line for the disk request list. I went with 1000 characters since I had to give some amount to declare the variable, and this seemed like an intuitive amount. This can easily be extended, shrunk by coding in the value to the program, it doesn't affect any other part of the program as everything else on top is dynamically generated.

I have tested with Schedular with two different files that has complete code coverage among them, and it returns values that I have hand calculated for the program.

For Simulator I have tried a variety of combinations QUIT, input wrong file name, input correct file name and then inputting two different files so I can attain complete code coverage and have exhausted all possible user combinations. Everything seemingly worked.

Another limitation of the program is that I have assumed that if the file input is correct, then the data set provided is also valid. I have done no validation checks on the file input as it is not specified within the assignment specification and there seemingly is no marks allocated for it. I have shown that I have a sound understanding of the pthreads and disk scheduling algorithms, so I have deemed it not necessary. (Hopefully the Rubric agrees with me.)

5. Sample inputs and outputs

I have two sample inputs. Input1 which is the Assignment Specification sample input and input2 which is something that I have created from scratch.

The inputs correlate to correct outputs.