Jun Zhen
CSS 430
11/10/19

Program 3 Report

Part 1:

The premise for part 1 is to improve the overall thread synchronization for ThreadOS by creating and using monitor as our synchronization technique. One of the main objectives for a monitor utility is to prevent threads from accessing and mutating the same files at the same time. To achieve this, we put threads into a queue and within this queue, we will able to either suspend a thread or resume it's operation.

Main components of the program:

SyncQueue.java – holds a list of QueueNode object that represent a certain state (sleep, wakeup). The kernel will use this class to access the array of threads that are either sleeping or operating.

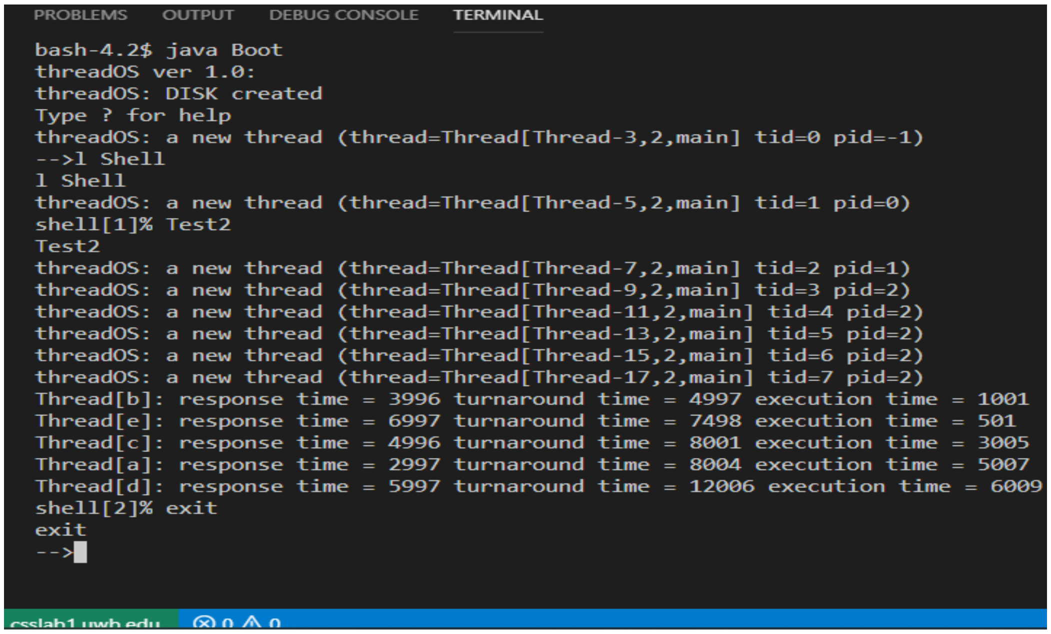
- **EnqueueAndSleep()** – put the thread into the queue and have it sleep.
- **dequeueAndWakeup()** – remove the first thread in the queue and wake it up.

QueueNode.java – This class contain the methods necessary for the kernel to change a thread state from sleep to wake and vice versa.

- **Sleep()** – put that thread to sleep and return it's tid.
- **Wake()** – put the thread into the queue and notify the parent thread.

KernelOld.java – A modified version of the kernel.java from ThreadOS/. We change the way it behaves by having the parent waiting for all of its child to terminate.

- Enable the ioQueue. dequeueAndWakeup for requesting the kernel's acceptance and completion.



```
bash-4.2$ java Boot
threadOS ver 1.0:
threadOS: DISK created
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Shell
l Shell
threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)
shell[1]% Test2
Test2
threadOS: a new thread (thread=Thread[Thread-7,2,main] tid=2 pid=1)
threadOS: a new thread (thread=Thread[Thread-9,2,main] tid=3 pid=2)
threadOS: a new thread (thread=Thread[Thread-11,2,main] tid=4 pid=2)
threadOS: a new thread (thread=Thread[Thread-13,2,main] tid=5 pid=2)
threadOS: a new thread (thread=Thread[Thread-15,2,main] tid=6 pid=2)
threadOS: a new thread (thread=Thread[Thread-17,2,main] tid=7 pid=2)
Thread[b]: response time = 3996 turnaround time = 4997 execution time = 1001
Thread[e]: response time = 6997 turnaround time = 7498 execution time = 501
Thread[c]: response time = 4996 turnaround time = 8001 execution time = 3005
Thread[a]: response time = 2997 turnaround time = 8004 execution time = 5007
Thread[d]: response time = 5997 turnaround time = 12006 execution time = 6009
shell[2]% exit
exit
-->
```

Result after

Conclusion:

By running Test2.java, we can see that after it spawns multiple threads for round robin scheduling, Test2 will wait for all the child threads to terminate before terminating itself. Also, once the Shell terminates with the exit command, we see that the loader will appear indicating that it waited for Shell.java to complete before resuming its own operation.

Part 2:

The premise for part 2 is to test and compare the result of two different techniques to achieve mutual exclusion. By default, the ThreadOS is using spin loops to avoid threads from colliding with each other. We will modify the kernel even more to allow it to put thread to sleep when it tries to access data that is being written on. We will achieve this by incorporating SyncQueue as our data structure to store thread's id and their current status. We then allow the kernel to wake and sleep any threads.

Main components of the program:

KernelNew.java – A further modification of the Kernel java from ThreadOS/. Change disk access to disable spinning loops.

- Changed the rawread, rawwrite, and sync case statement to utilize our SyncQueue instead of ioQueue.

Test3.java – a test case that will stress test the new kernel and then record the change in millisecond.

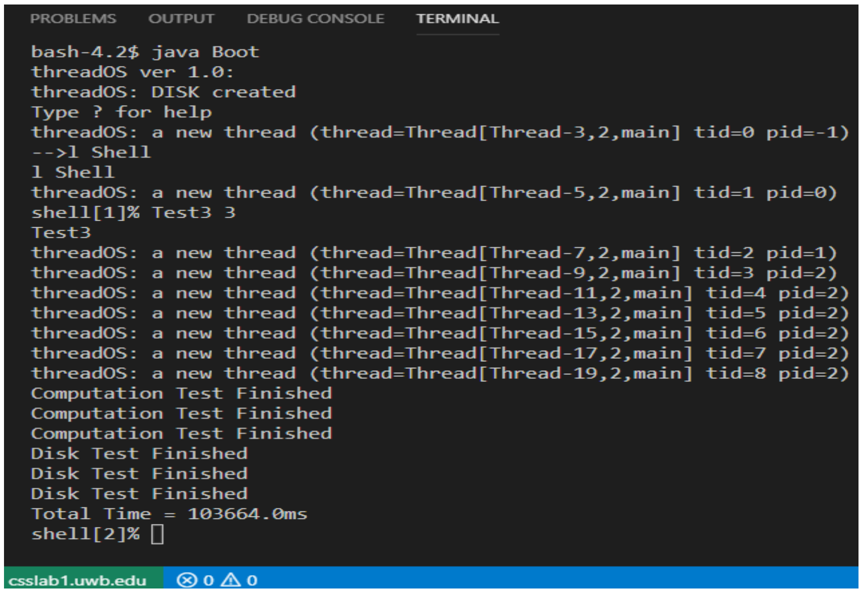
- **Run()** – main method to run the two test threads. It will run as many as the user specifies.

TestThread3a.java – a thread test case that will commute the iterative version of the Fibonacci sequence.

- **Run()** – compute the Fibonacci up to very large amount (100,000) to stress test the kernel and SyncQueue.

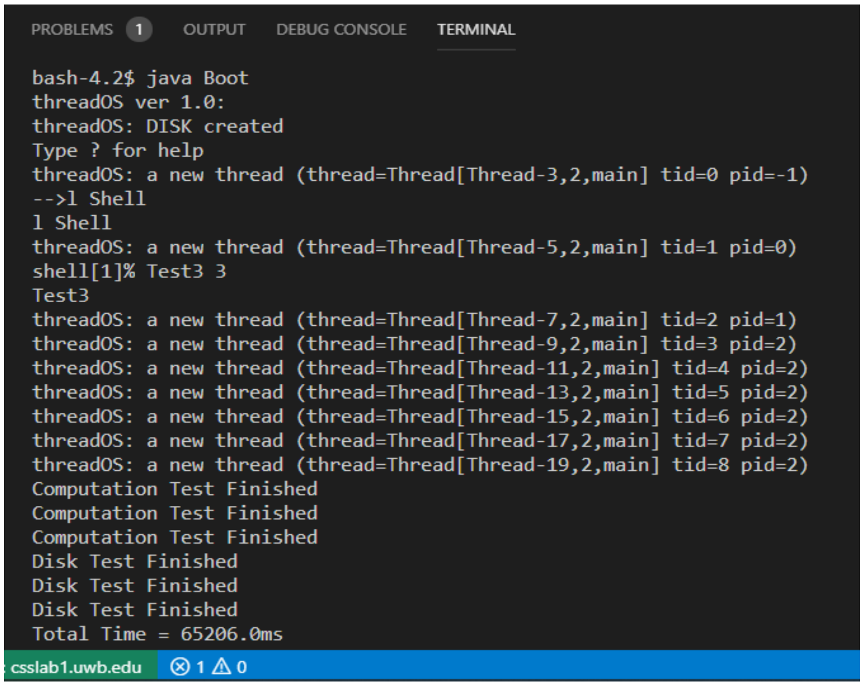
TestThread3b.java – a thread test case that will read and write a large amount of data into the system.

- **Run()** – compute the rawread and rawwrite system function for 500 time to stress test the kernel and SyncQueue.



```
bash-4.2$ java Boot
threadOS ver 1.0:
threadOS: DISK created
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Shell
l Shell
threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)
shell[1]% Test3 3
Test3
threadOS: a new thread (thread=Thread[Thread-7,2,main] tid=2 pid=1)
threadOS: a new thread (thread=Thread[Thread-9,2,main] tid=3 pid=2)
threadOS: a new thread (thread=Thread[Thread-11,2,main] tid=4 pid=2)
threadOS: a new thread (thread=Thread[Thread-13,2,main] tid=5 pid=2)
threadOS: a new thread (thread=Thread[Thread-15,2,main] tid=6 pid=2)
threadOS: a new thread (thread=Thread[Thread-17,2,main] tid=7 pid=2)
threadOS: a new thread (thread=Thread[Thread-19,2,main] tid=8 pid=2)
Computation Test Finished
Computation Test Finished
Computation Test Finished
Disk Test Finished
Disk Test Finished
Disk Test Finished
Total Time = 103664.0ms
shell[2]%
```

old Kernel.java



```
bash-4.2$ java Boot
threadOS ver 1.0:
threadOS: DISK created
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Shell
l Shell
threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)
shell[1]% Test3 3
Test3
threadOS: a new thread (thread=Thread[Thread-7,2,main] tid=2 pid=1)
threadOS: a new thread (thread=Thread[Thread-9,2,main] tid=3 pid=2)
threadOS: a new thread (thread=Thread[Thread-11,2,main] tid=4 pid=2)
threadOS: a new thread (thread=Thread[Thread-13,2,main] tid=5 pid=2)
threadOS: a new thread (thread=Thread[Thread-15,2,main] tid=6 pid=2)
threadOS: a new thread (thread=Thread[Thread-17,2,main] tid=7 pid=2)
threadOS: a new thread (thread=Thread[Thread-19,2,main] tid=8 pid=2)
Computation Test Finished
Computation Test Finished
Computation Test Finished
Disk Test Finished
Disk Test Finished
Disk Test Finished
Total Time = 65206.0ms
```

new Kernel.java

Discussion:

After running Test3 with TestThread3a and TestThread3b, we can see the difference between the old kernel and new. Comparing these two, we can see that with monitor and better synchronization, we were able to decrease the total time of writing and reading from the disk. Old kernel timing is 10,3664 milliseconds whereas the new Kernel timing is at 65,206 milliseconds. A major factor in the performance boost has to do with changing the kernel's rawread and rawwrite method. In the old kernel, threads are actually placed into a while-loop that continues on until the disk that the thread want to access is open. This implementation of kernel is a spinning lock implementation and it is used to establish mutual exclusivity. It is to ensure that no two threads are writing and reading from the same disk at the same time. This technique is great for threads that want to perform quick reads or quick writes, but bad for long duration of writes or reads. With the new kernel, instead of putting threads into a loop, we will actually put them into our SyncQueue. SyncQueue will keep track of the current threads and it will wake or put threads to sleep. With this implementation, threads that don't have access to read or write will not hold onto the CPU while they wait. As the result indicate, it greatly boosted the result of our performance. While the new Kernel did perform better, but it is not without flaws. Note that our thread test is doing a large amount of read/write. Have it been short burst of read/write, the old kernel might be faster due to threads leaving their while-loop faster. The new Kernel will perform poorer because of it will have to call multiple context switches as threads are leaving and entering their critical sections.