CptS 370 Program 4: Synchronization

REPORT

Part 1: Implementing SysLib.join( ) and SysLib.exit( )

QueueNode

Initialize a vector named queue to store the tid. Two major function are included in this class:

- sleep()

  The incoming thread will be put into sleep by calling the wait() function, then return and removes the id from the queue top by calling queue.remove(0) => the very first tid.

- wake(int tid)

  Wake up the incoming thread by putting itd to the queue vector and notify the thread using the notify() function.

SyncQueue

Initialize the size of the array with and without arguments. Without the argument, by default DEFAULTCOND = 10. The index of the array shows the condition for the thread. Two major function are included in this class:

- dequeueAndSleep(int condition)

  The incoming thread will be put into sleep by calling the sleep() function once the condition input is in the range of the array size. The tid value will be returned for other threads to finish.

- dequeueAndWakeup(int condition, int tid)

  The incoming thread with or without the id (the one on the queue top) will be wake up by calling the wake() function once the condition input is in the range of the array size.

After modifying the Kernel(.old) by following the instructions, this is the result of running the Test2:

```
[ywang11@sigill prog4]$ javac Kernel.java
Note: Kernel.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
[ywang11@sigill prog4]$ java Boot
threadOS ver 1.0:
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 = 3998 turnaround time = 4999 execution time = 1001
Thread[e]: response time = 6999 turnaround time = 7499 execution time = 500
Thread[c]: response time = 4998 turnaround time = 8001 execution time = 3003
Thread[a]: response time = 2997 turnaround time = 8002 execution time = 5005
Thread[d]: response time = 5998 turnaround time = 12005 execution time = 6007
shell[2]% ^C[ywang11@sigill prog4]$
```

Part 2: Implementing Asynchronous Disk I/O*

For part 2, I run the Test3 in both old and new Kernel in order to test out the efficiency and make a comparison on the two of them. Two test files are created alongside:

- TestThread3a

  Calculate a math equation with some random pows and sqrts in it and looing the sum to an astronomical number in order to put some pressure on the CPU

- TestThread3b

  Write into the byte buffer and read, looping it through 500 blocks

After modifying the Kernel(new) by following the instructions, this is the result of running the Test3:

```
[ywang11@sigill prog4]$ java Boot
threadOS ver 1.0:
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 4
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)
threadOS: a new thread (thread=Thread[Thread-21,2,main] tid=9 pid=2)
threadOS: a new thread (thread=Thread[Thread-23,2,main] tid=10 pid=2)
Compute Thread Done!
Compute Thread Done!
Compute Thread Done!
Compute Thread Done!
Disk Thread Done!
Disk Thread Done!
Disk Thread Done!
Disk Thread Done!
Elapsed time: 89593 ms
[ywang11@sigill prog4]$
```

Switch back to Kernel.old, here is the result of running the Test3:

```
[ywang11@sigill prog4]$ java Boot
threadOS ver 1.0:
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 4
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)
threadOS: a new thread (thread=Thread[Thread-21,2,main] tid=9 pid=2)
threadOS: a new thread (thread=Thread[Thread-23,2,main] tid=10 pid=2)
Compute Thread Done!
Compute Thread Done!
Compute Thread Done!
Compute Thread Done!
Disk Thread Done!
Disk Thread Done!
Disk Thread Done!
Disk Thread Done!
Elapsed time: 107212 ms
shell[2]%
```

Comparison

By comparing the results of two tests, it is clear that the efficiency of the new implementation is obviously higher than the old one. Compared to the data, when x = 4, the new running result :89593 is 17619mm faster than the old :107212, which is more than 17 seconds. The main difference, however, is that the new Kernel allows the thread to sleep and it does not implement the spinlocks, which also shows that the interrupt-based scheduling saves the loss caused by waiting for other threads while reading one by one.