Programming Assignment 3

Sharanya Sudhakar

CSS 430

11/10/19

Part 1:

SyncQueue.java

SyncQueue allows a thread to sleep and wakeup on a specific condition. This monitor is implemented with a *QueueNode* as its inbuilt property. *SyncQueue* enables the implementation of *SysLib.join()* and *SysLib.exit().* Join and exit are implementations in the Kernel.java that utilizes the *SyncQueue* for implementing the *SysLib.join()* and *SysLib.exit()* calls of the thread based on a condition. The underlying Queue is *QueueNode*. This implementation is based on the Unix/Linux platform where the parent thread waits for the child thread to terminate. The child thread terminates by calling *SysLib.exit()* and it return the id of the child thread that wakes up the sleep parent thread in turn.

To implement *SysLib.join()* and *SysLib.exit(), Kernel.java* is modified to implement *SyncQueue* and *SyncQueue* is implemented to have an underlying *QueueNode*.

Kernel.java

In this file, under the WAIT and EXIT cases are modified to ensure the parent id is retrieved and put to sleep. And in the exit case, the parent is woken up by sending the id of the child that exited.

Private / Public	Method / Date	Description
CASE	WAIT	Get the *TCB* of the currently running thread and use it to get the *Tid* in order to *enqueueAndSleep()* that thread.
CASE	EXIT	Get the *TCB*, *dequeueAndWakeup()* thread using *pid* and *tid*, then delete the thread in scheduler.

In ***SyncQueue.java*** file,

The class is initialized with a default constructor having 10 threads or a parameterized constructor with a custom number of threads. Then the two methods within the class are *enqueueAndSleep* and *dequeueAndWakeup*. In *enqueueAndSleep*, the thread with the condition is put to sleep and in *dequeueAndWakeup*, the thread is woken up with its child *tid* as a parameter or a default child of 0(zero). The design of Syncqueue.java follows the assignment specifications:

Private/Public	Methods/Data	Descriptions
private	QueueNode[] queue	maintains an array of QueueNode objects, each representing a different condition and enqueuing all threads that wait for this condition. You have to implement your own QueueNode.java. The size of the **queue** array should be given through a constructor whose spec is given below.
public	SyncQueue( ), SyncQueue(int condMax)	are constructors that create a queue and allow threads to wait for a default condition number (=10) or a *condMax* number of condition/event types.
public	enqueueAndSleep(int condition)	enqueues the calling thread into the queue and waits until a given *condition* is satisfied. It returns the ID of a child thread that has woken the calling thread.
public	dequeueAndWakeup(int condition), dequeueAndWakeup(int condition, int tid)	dequeues and wakes up a thread waiting for a given *condition*. If there are two or more threads waiting for the same *condition*, only one thread is dequeued and resumed. The FCFS (first-come-first-service) order does not matter. This function can receive the calling thread's ID, (*tid*) as the 2nd argument. This *tid* will be passed to the thread that has been woken up from *enqueueAndSleep*. If no 2nd argument is given, you may regard *tid* as 0.

In ***QueueNode.java***

The class has an underlying queue which holds the child threads. This class has a vector that represents the threads with the same condition. There are two methods, *sleep()* and *wake()*. In the *sleep()*method, the thread is put to sleep until notification. It will return the id of the calling thread. In *wake()* method, the thread with *tid* is added to the queue and notifies the sleep method.

Private / Public	**Method / Date**	**Description**
Private	Vector<Integer> queue	Data structure to hold the threads that are enqueued by *wake()*
Public synchronized	Sleep()	The sleep method returns the id of the first thread that wakes up the thread in sleep()
Public synchronized	Wake()	This method enqueues the *tid* of the thread into the vector array and notifies *sleep*()

```
sharanu@uw1-320-01:~/U/ThreadOS$ 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 = 3980 turnaround time = 4981 execution time = 1001
Thread[e]: response time = 6980 turnaround time = 7480 execution time = 500
Thread[c]: response time = 4979 turnaround time = 7982 execution time = 3003
Thread[a]: response time = 2979 turnaround time = 7983 execution time = 5004
Thread[d]: response time = 5979 turnaround time = 11984 execution time = 6005
shell[2]% exit
exit
-->q
q
sharanu@uw1-320-01:~/U/ThreadOS$
```

Output for part1.

Part 2

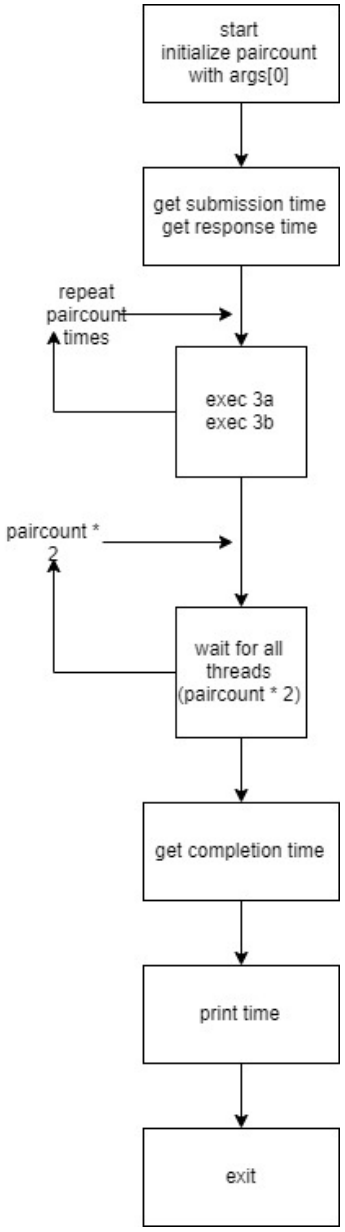
In this section, three test classes where created to test the Kernel implementation of the RAWREAD, RAWWRITE and SYNC so that disk access no longer has spin loops.

Modifying the *Kernel.new* / ***Kernel.java***

Instead of the spin loop, the *ioqueue* is added to *enqueue* and *sleep* the threads based on certain condition.

MyTest3.java

This class executes user threads as pairs. There are two threads to be executed. TestThread3a and TestThread3b. 3a, contains computational execution, while 3b contains disk read/write(I/O) operations. These threads are executed alternately, with an input argument number of times.



TestThread3a.java

This thread performs computation. It produces the factorial and in-turn finds the *tan* and *atan* multiple times. The out put is never printed, this is a computationally intense thread. Its one and only purpose.

TestThread3b.java

This thread is a read operation only. It reads 512 byte blocks of data 250 times. Making it I/O intense.

Both threads print when done.

```
sharanu@uw1-320-01:~/U/ThreadOS$ 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]% MyTest3 7
MyTest3
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)
threadOS: a new thread (thread=Thread[Thread-25,2,main] tid=11 pid=2)
threadOS: a new thread (thread=Thread[Thread-27,2,main] tid=12 pid=2)
threadOS: a new thread (thread=Thread[Thread-29,2,main] tid=13 pid=2)
threadOS: a new thread (thread=Thread[Thread-31,2,main] tid=14 pid=2)
threadOS: a new thread (thread=Thread[Thread-33,2,main] tid=15 pid=2)
threadOS: a new thread (thread=Thread[Thread-35,2,main] tid=16 pid=2)
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
ThreadTest3: response time = 1801 turnaround time = 50671 execution time = 48870
shell[2]% exit
exit
```

New Kernel Output.

Discussion:

The output for 25 pairs of threads and 7 pairs of threads was implemented for both old and new Kernel. Looking at the output, it is obvious that the old kernel is faster than the new kernel. In the new kernel execution, the thread with computational intensity is completed first, while the I/O intense thread executes at the end. On the other hand, the old kernel has interleaved output. Meaning the waiting

doesn’t happen due to context switch as much as in the new Kernel. Every I/O operation in the new kernel is interrupted and context switched to the computational threads, enabling this overhead that slows down the execution.

With the given thread tests in small pairs, it is clear that the old Kernel has better execution time. In other cases when number of threads increase, NEW kernel is faster by a margin. As shown in the case of 25 pairs. So every test case is different and old or new the kernel efficiency depends on the number of threads executed.

NEW Kernel: 25 pairs

```
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
ThreadTest3: response time = 1708 turnaround time = 161769 execution time = 160061
shell[2]% shell[2]% exit
exit
-->q
q
sharanu@uw1-320-01:~/U/ThreadOS$
```

OLD Kernel: 25 pairs

```
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
ThreadTest3: response time = 1023 turnaround time = 161729 execution time = 160706
shell[2]% exit
exit
-->q
q
sharanu@uw1-320-01:~/U/ThreadOS$
```

Output for 7 pairs below:

NEW KERNEL 7 pairs:

```
sharanu@uw1-320-01:~/U/ThreadOS$ 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]% MyTest3 7
MyTest3
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)
threadOS: a new thread (thread=Thread[Thread-25,2,main] tid=11 pid=2)
threadOS: a new thread (thread=Thread[Thread-27,2,main] tid=12 pid=2)
threadOS: a new thread (thread=Thread[Thread-29,2,main] tid=13 pid=2)
threadOS: a new thread (thread=Thread[Thread-31,2,main] tid=14 pid=2)
threadOS: a new thread (thread=Thread[Thread-33,2,main] tid=15 pid=2)
threadOS: a new thread (thread=Thread[Thread-35,2,main] tid=16 pid=2)
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
ThreadTest3: response time = 1801 turnaround time = 50671 execution time = 48870
shell[2]% exit
exit
```

OLD KERNEL 7 pairs:

```
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]% MyTest3 7
MyTest3
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)
threadOS: a new thread (thread=Thread[Thread-25,2,main] tid=11 pid=2)
threadOS: a new thread (thread=Thread[Thread-27,2,main] tid=12 pid=2)
threadOS: a new thread (thread=Thread[Thread-29,2,main] tid=13 pid=2)
threadOS: a new thread (thread=Thread[Thread-31,2,main] tid=14 pid=2)
threadOS: a new thread (thread=Thread[Thread-33,2,main] tid=15 pid=2)
threadOS: a new thread (thread=Thread[Thread-35,2,main] tid=16 pid=2)
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3a done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
TestThread3b done.
ThreadTest3: response time = 1588 turnaround time = 49046 execution time = 47458
shell[2]% exit
exit
-->q
```