Margaret Connor
CSS 430 Spring 2019
P2 Report

SyncQueue, QueueNode, and Kernel Description

My SyncQueue allowed for asynchronized computing, this prevent I/O-bound threads from wasting CPU power. The SyncQueue provides a queue in which processes could "wait" while until a condition was fulfilled. The process would be given to the SyncQueue where its process ID would become a condition key, wait() would then essentially put the process to sleep until a condition was fulfilled. When the condition was fulfilled the process would resume and be sent the child process ID of the process what fulfilled the condition. The SyncQueue was then integrated into the Kernel.java to work with the SysLib.join( ) and SysLib.exit( ) calls, allow for asynchronized computing.

Test3 Description

My test is similar to Test3.class provided by the instructor in the fact that it simply took in a number and ran a computational and read/write thread that many times. My computational thread performed $x_n = x_{n-1}{}^n$ 5000 times, starting with $x_0 = 1$. My read/write thread read into a buffer from a random block using SysLib.rawread and then wrote to a random block using SysLib.rawwrite, it performed this task 100 times.

Test 3 Performance Comparisons

During my testing I received slightly better performance results on the new (asynchronous) kernel than I had received on the old kernel(busy wait). In my trial the new kernel was 2,564 msec faster than the old kernel, test results can be found in table 1 (Full results can be found in attached Performance_Results_Test3.txt). This is most likely due to the new kernel being asynchronous, meaning it allowed the shell to work on other processes while it waited to read or write from the disk. While the performance was only slightly shorter, this optimization would be greatly useful in cases where more processes needed to be ran as fast as possible.

Table 1: Test 3 Performance Results

	Elapsed time (msec)
Test3 New Kernel (asynchronous)	86437
Test3 Old Kernel (busy wait)	89001