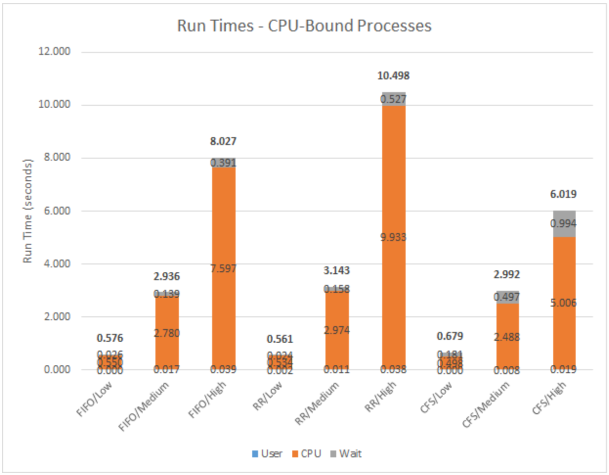
Adam Holt & Joe Cosenza
CSCI3753 Programming Assignment 4 Extra Credit

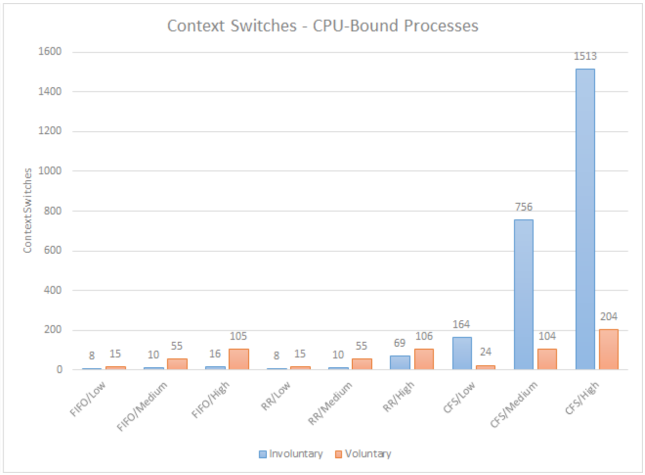
Test Environment: Course VM – Ubuntu 14.04
Memory: 3.8 GiB
Processor: Intel Core i7 2.3 GHz
OS Architecture: 64 bit

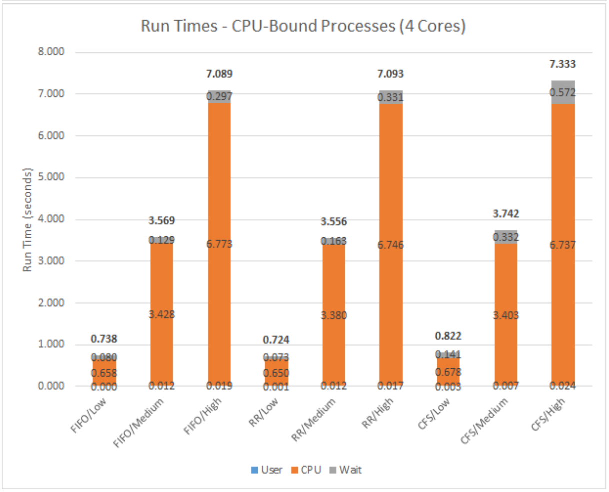
<u>Multi-Core:</u>

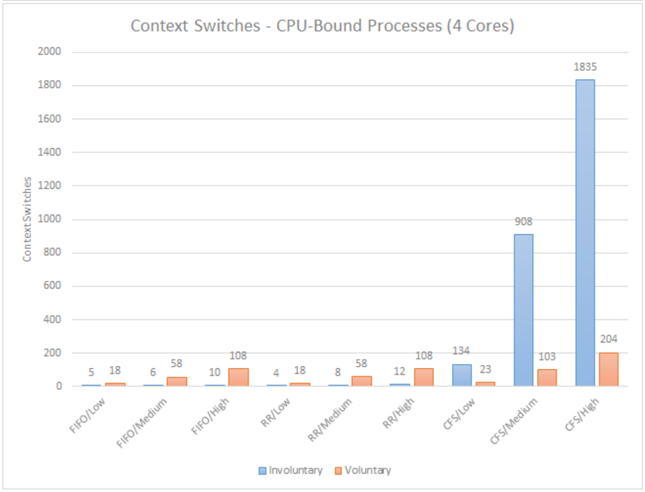
To benchmark the three different scheduling policies at different core amounts, the same testing method was repeated after changing the amount of CPU cores from 1 to 4 then 8. In order to receive relative results, the wait times had to be calculated differently to account for the addition of CPU cores by multiplying wall time by the amount of processors. Below are the comparisons of the results by amount of cores side-by-side.

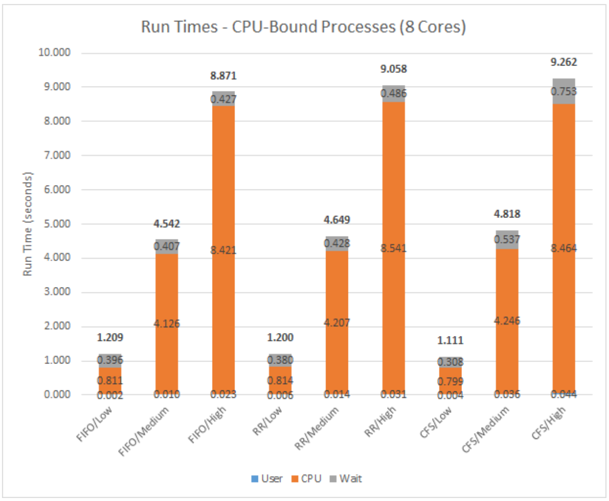
CPU Bound Processes

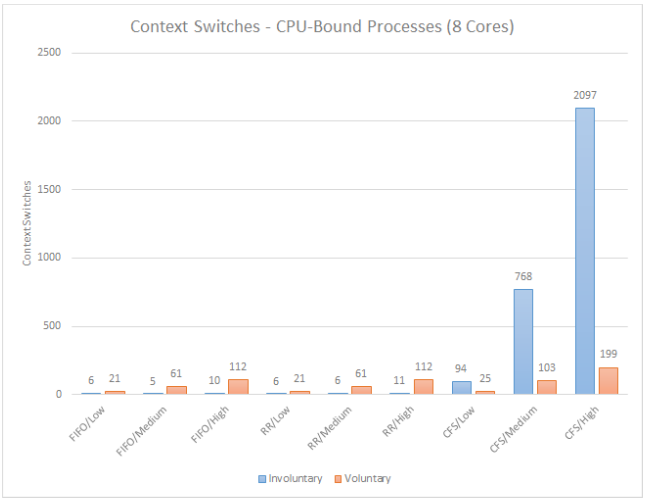






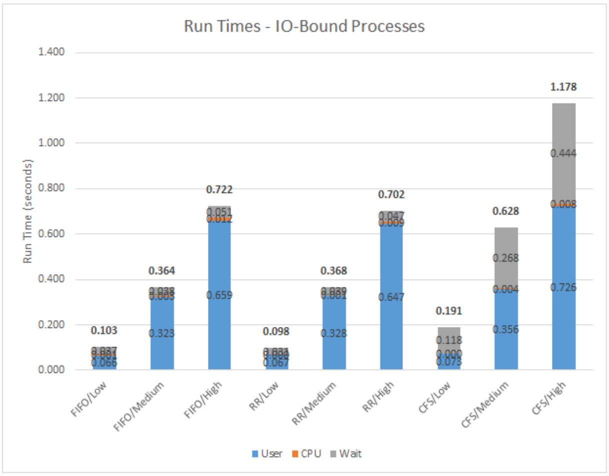


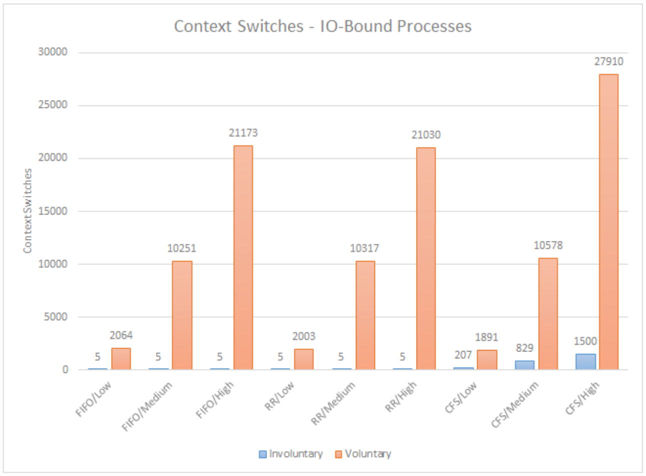


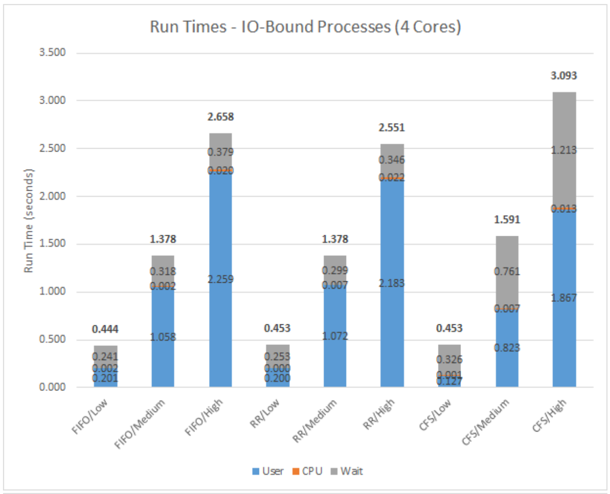


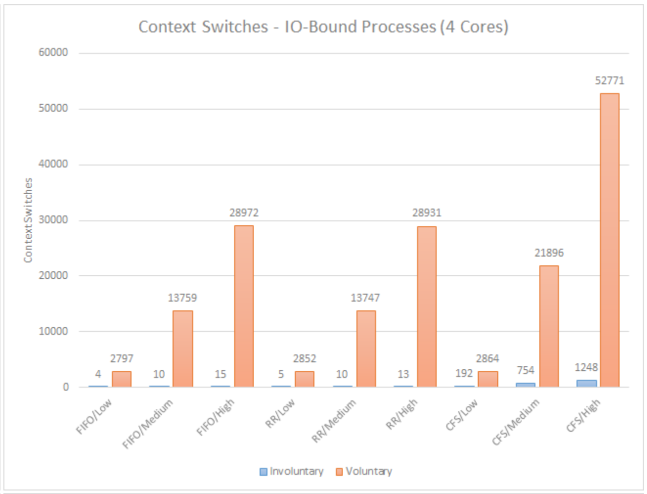
As the above shows, the comparison of 4 and 8 cores yield relatively similar results. However, if examined there are distinct traits within each comparison. For CPU-bound operations, the run time of the CFS scheduler is negatively impacted by increasing the number of cores. The FIFO and RR scheduling policies remain the same relative to one another regardless of number of cores. While faster than the other two policies when run on a single core, the CFS scheduler spends more CPU time at higher numbers of cores as well as having more involuntary context switches.

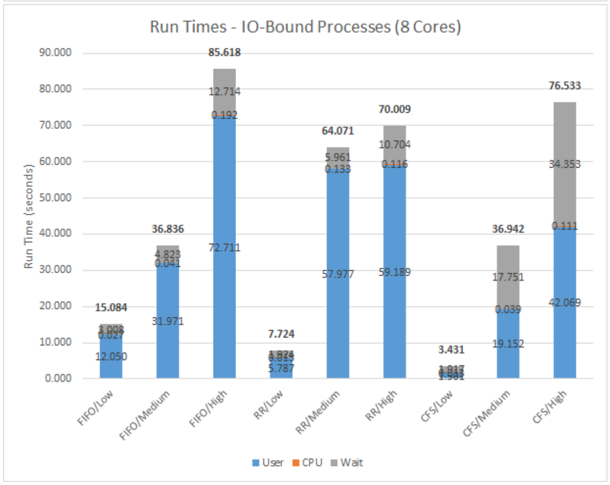
I/O Bound Processes

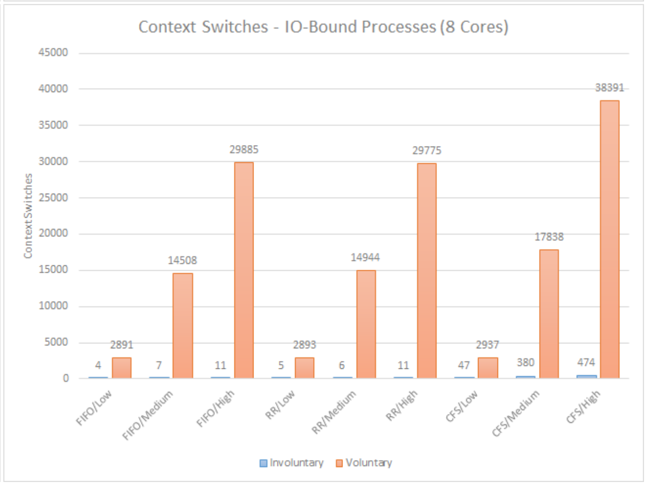






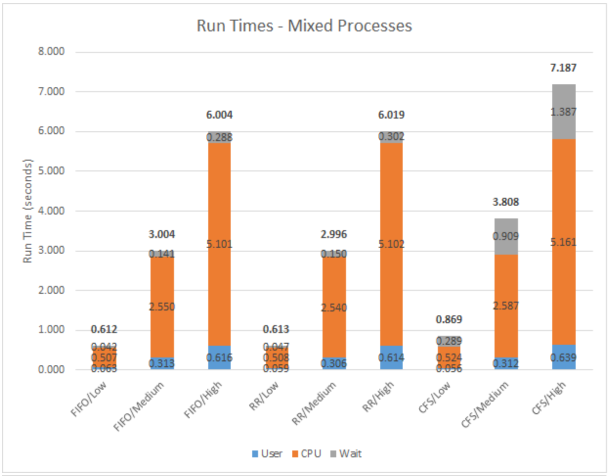


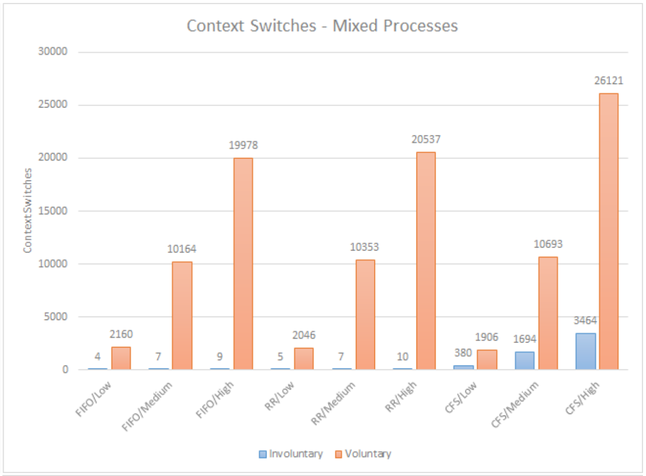


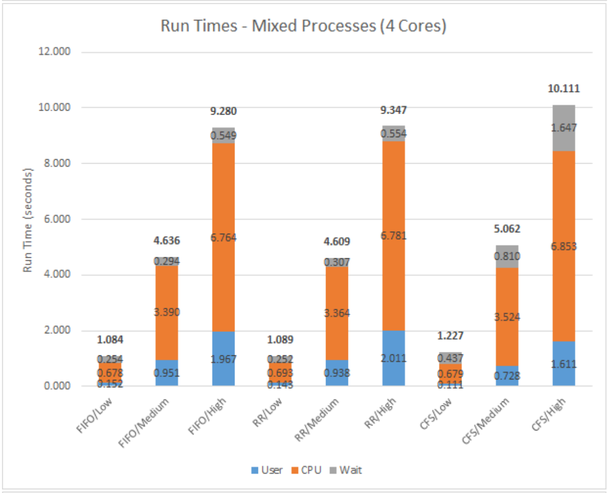


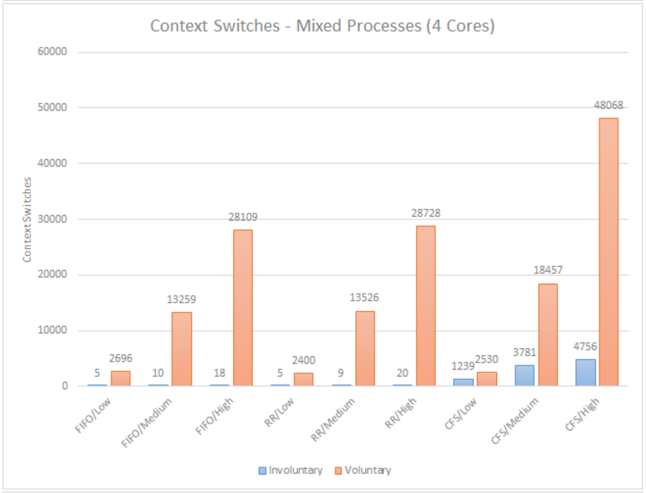
I/O-bound operations are much more diverse as the number of cores increases. Although they are similarly related, the scale of the run times is different for each result. At all number of cores, the amount of CPU time is, predictably very low, however, the wait times and user time increase greatly as the number of cores does. The amount of context switching was the lowest using 1 core, where at 4 cores, the more processes yielded a larger number of context switching. The same result was found with 8 cores, however was significantly lower than the amount in 4 cores with the use of more processes. These results would seem to stem from waiting for a resources in order to read or write, so that increasing the number of cores actually slowed the process overall.

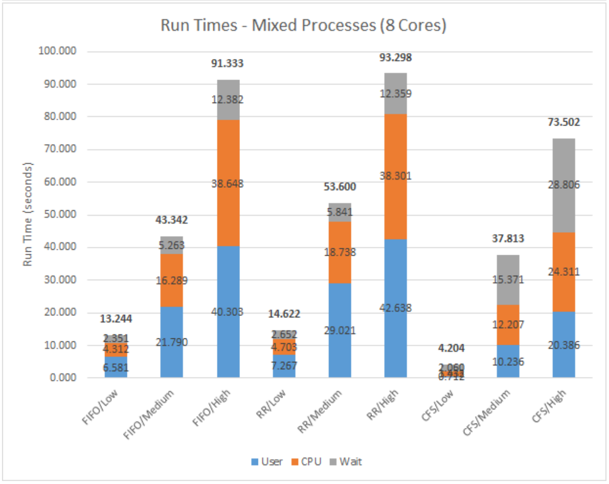
Mixed Processes

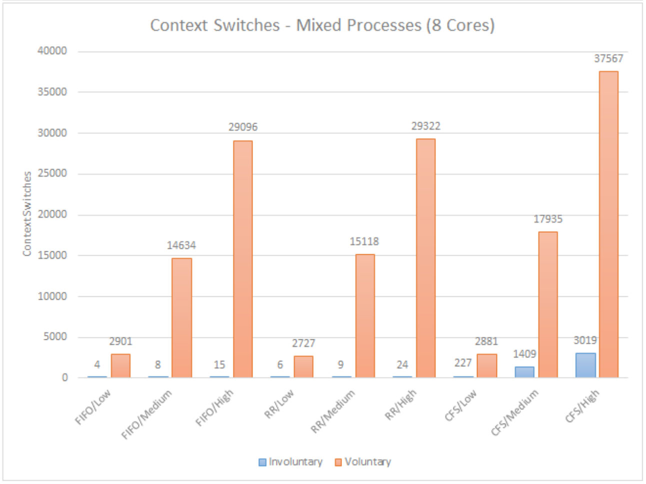












The mixed operations on 8 cores showed results similar to the I/O-bound operations where the run times were much larger than either of the other core amounts. However the CPU time returned and yield a similar ratio to the user time that was reported with the use of 1 core. The context switching also followed a similar pattern to the I/O bound operation.

Conclusion:

The addition of cores can reach a point of inefficiency when using 8 cores in the case of I/O-based operations. However, the CPU-based operations maintain around the same amount of time regardless of the amount of cores above 1, where a single core is still more efficient in all cases. 8 cores becomes extremely inefficient when there are more processes, especially using the RR and the CFS scheduling policies. It is interesting to see from these benchmarks that adding additional cores, not only doesn’t always increase the efficiency of a program, but can actually cause it to be less efficient depending on what types of operations it relies on.