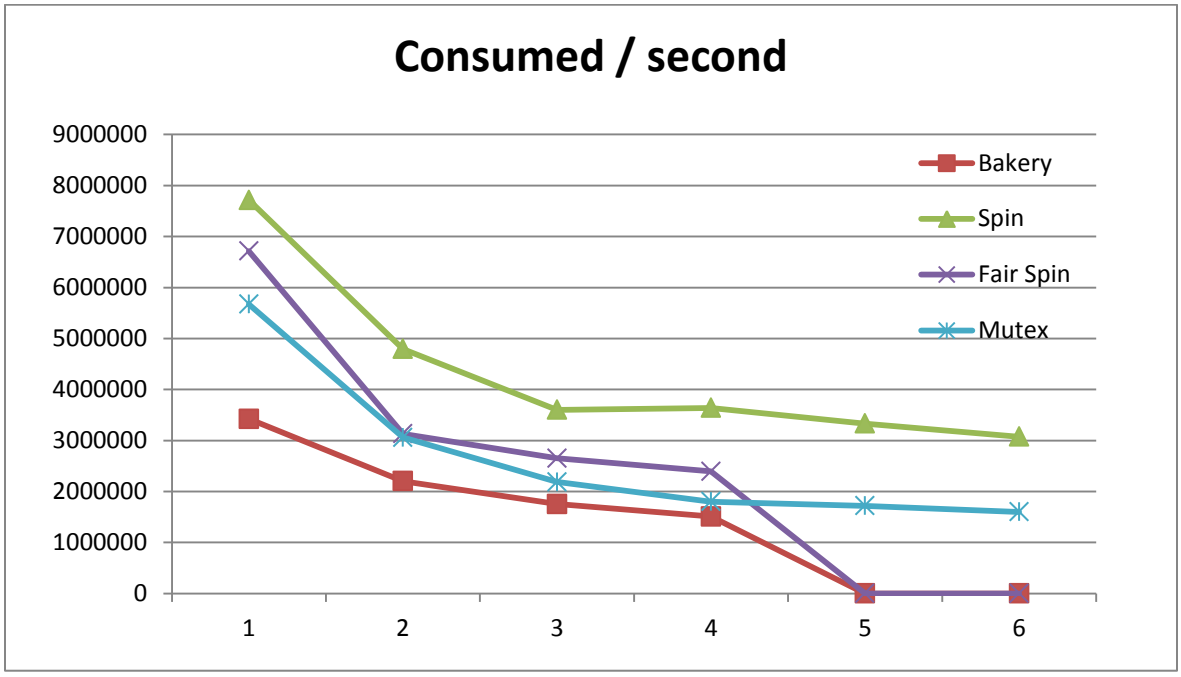
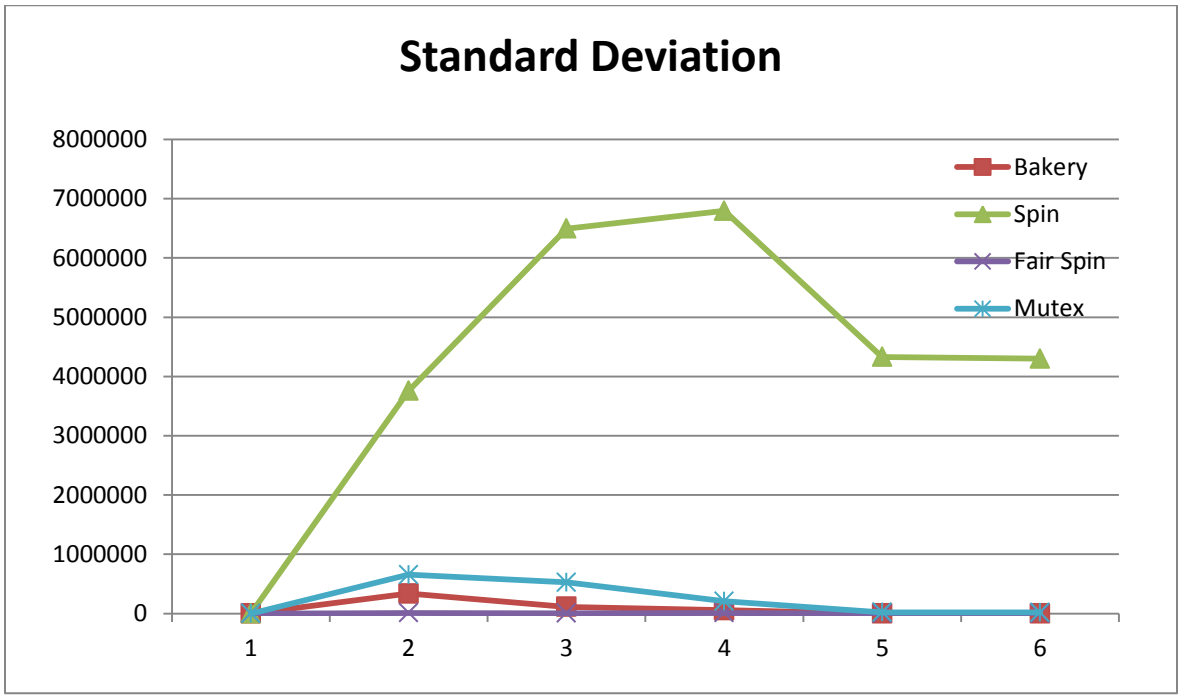
Jakub Szpunar
CS5460
HW3





As we can see from the previous graphs, performance varies drastically between the locks. I'll discuss the locks one by one.

The Bakery lock had the lowest overall throughput. However, it had the second best variance. The throughput overall was lower than the other locks because of the mechanism of the bakery lock. Each time the bakery lock wishes to obtain the lock, it must loop through all the other locks and find the next available number. The actual lock function also involves memory fences which add some slight delay. The function also must loop through all the information of the other threads to see if it can go which may add to its delay. The relatively decent variance of the lock is likely due to the ticket system. Threads are more or less served in the order they come. However, threads that request locks at similar times are likely to get the same number. In this case the tie breaker is not as fair as it is based on thread id. The bakery lock had an odd crash of performance when the thread count was five or higher. I'm not exactly sure of the reasoning behind this, but it was likely due to passing the core count of the computer and perhaps context switches and memory trashing. Otherwise, the performance of this lock increased about on average as the thread count increased. Higher thread counts cause more contention on the still single data structure of interest.

The Spin lock had the highest overall throughput. This is likely due to its very simple nature. The spin lock has no advanced logic. It simply relies on a hardware supported atomic operation which is very fast. However, the spinlock is also by far the least fair lock. Performance between threads varied must more than other locks. Because the spinlock lacks any sort of first come first serve ticket or token, it is very unfair. The spin lock faired best with an increase in thread count. This is once again likely to do with its very simple mechanism.

The fair spin lock had the second best throughput. This is due to it also relying on fairly simple hardware supported operations. This lock was a little slower because it relies on two operations instead of one. The lock was by far the fairest lock. This is due to its atomic ticket system. No two threads can have the same number. The first thread to come will be the first thread to be served. This lock shared the bakery lock's issue of poor performance with 5 or more threads. I'm guessing similar reasoning applies here.

The mutex lock was the most balanced lock. While it did not have the highest throughput, or the best variance, it balanced the two parameters. The mutex likely is heavier weight than a spin lock which led to somewhat lowered performance. However, it also has some sort of a wait queue inside that helps keep the variance in check. The mutex also did not have any issues as thread count increased.