# Assignment 2: Thread Synchronization

## Data Results:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case | Threads | Zero Index | Sequential (ms) | Thread Join (ms) | Busy Wait  (ms) | Semaphore (ms) |
| 1 | 2 | 50M+1 | 240 | 149 | 196 | 0 |
| 2 | 4 | 75M+1 | 239 | 143 | 170 | 0 |
| 3 | 8 | 88M | 240 | 138 | 199 | 1 |
| 4 | 2 | n/a | 239 | 142 | 201 | 0 |
| 5 | 4 | n/a | 240 | 144 | 209 | 1 |
| 6 | 8 | n/a | 240 | 143 | 202 | 1 |

### Computer Architecture:

The computer that was used had a total of two cores each with a single thread. The machine running the code is running a very slim Arch Linux, with very little daemons and processes running simultaneously. Running the code without the window manager running resulted in even higher speeds. This latest experiment rendered all but one timing at 0ms. The speeds remained somewhat consistent throughout the experiments.

### Sequential Search:

The sequential time took the longest, and stayed static throughout the experiment, creating a good consistent benchmark.

### Thread Join:

The thread join was second fastest, with consistent run times of 138-149ms. The parent was sitting in a waiting state, which prevented it from competing for the CPU.

### Busy Wait:

With only two hardware threads, the Busy Wait times are significantly higher than that of the thread join processes. During the busy wait, the parent was competing with the child threads for the CPU, which resulted in slower run times. Trial two was faster than average, but upon several more runs I gathered normalized run times from 170-204ms.

### Semaphore:

When the processes came to the semaphore, times were significantly faster. Some even read 0ms, which means it was almost instantaneous. These results confirm that semaphores are the fastest way to synchronize threads in this specific application.