COMP 3510 – Embedded Systems Development  
Rain Li, David Harris

**Lab 2 Report**

Our implementation of the Lab 2 assignment successfully detects when a device has generated an event and processes events in the buffer to make way for new events. Furthermore, our code stores events that would be lost and stores them so they may be serviced after the current event is processed. Put simply, our code works.

Below is the information we collected by testing our code with preset values:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number of Devices | lambda | mu | Avg Missed Events % | Avg Response Time (s) | Avg Turn Around Time (s) |
| 2 | 2 | 10 | 0.5 | 0.001985 | 0.393403 |
| 2 | 2 | 30 | 0.5 | 0.002447 | 1.455421 |
| 2 | 2 | 60 | 0.5 | 0.002993 | 7.334774 |
| 2 | 2 | 90 | 22.05 | 0.002084 | 30.791606 |
| 4 | 2 | 10 | 0.25 | 0.001952 | 1.171597 |
| 4 | 2 | 30 | 0.25 | 0.002691 | 1.117774 |
| 4 | 2 | 60 | 0.50 | 0.002112 | 1.684308 |
| 4 | 2 | 90 | 12.13 | 0.002950 | 14.036064 |
| 8 | 4 | 10 | 0.12 | 0.003832 | 1.034293 |
| 8 | 4 | 30 | 0.12 | 0.003953 | 1.825247 |
| 8 | 6 | 60 | 0.25 | 0.005247 | 2.355760 |
| 8 | 6 | 90 | 0.98 | 0.00577 | 28.341445 |

Due to the nature of Round Robin, the code is bound to miss events. When a flag indicates that there are buffered events, the program services Device 0, and then moves on to Device 1 and Device 2, all the way to Device n, and services the events in that order. Device 0 does not stop generating events while Round Robin iterates through n devices and may generate another event before the server makes a complete cycle. Since our program cannot respond to events while serving, this results in misses.

In an attempt to reduce the number of misses, we store events as they come in in another array, and service them in a similar Round Robin fashion. All the events that are successfully stored in the array are serviced, but any that are overlooked are missed.

Our results indicate that, with optimal service time, our code is able to process generated event with over a 99% hit rate. As shown in Figure 1, the time it takes to service each event (**mu**) increases, our code misses more events since the blackout time due to servicing an event is longer and code cannot react to events generated in that period.

Figure 1

We encountered a bit of an anomaly as we worked. At seemingly random times, the program would miss large amounts of events, and take long stretches of time to service the ones it had. We don’t know what causes this, and we have no way to recreate or prevent it. We did, however, keep this information and included it in our data collection. As pictured in Figure 2, this makes it hard to find a trend in the number of missed events as response time increases, mostly due to the fact that response time was sporadically changing.

Figure 2

Similar to our Lab 1 findings, we found that with a longer turnaround time, the number of missed events generally increased, as the program could not effectively handle the arrival of new events, even with interrupts (see Figure 3 below).

Figure 3

Our code minimizes the time it takes to completely service all buffered events in a single cycle by only cycling through the number of devices that are actively generating events, not by checking each flag individually on each cycle. We also reduced memory footprints and execution times by using minimal variables and bitwise operations. Implementing interrupts helped our program overall, with a few anomalies that remain outliers. The software keeps track of events and stores them to be serviced as best as it can, but in the current environment there are still not enough resources to effectively service the generated events.

To further improve performance, we may have implemented the device buffer in such a way that it would process each buffered event based on the time it had buffered, rather than in a Round Robin fashion. This would reduce overall response (and turnaround) time, and improved the effectiveness of the solution.

For the reader’s convenience, graphs from Lab 1 have been included below.

**Graphs from Lab 1**

Figure 1-1

Figure 1-2

Figure 1-3