COMP 3510 – Embedded Systems Development
Rain Li, David Harris

Lab 1 Report

Our implementation of the Lab 1 assignment successfully detects when a device has generated an event and processes events in the buffer to make way for new events. 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	0.028097	0.13624
2	2	30	14.5	0.099073	0.305058
2	2	60	30	0.19745	0.474583
2	2	90	46	0.293826	0.570439
4	2	10	0.5	0.013155	0.070574
4	2	30	4.5	0.09666	0.230552
4	2	60	14.75	0.2313945	0.443452
4	2	90	26.75	0.394554	0.61029525
8	4	10	0.25	0.011846375	0.070233125
8	4	30	1.375	0.089242625	0.247915625
8	6	60	6.77125	0.538235875	0.947026125
8	6	90	17.6525	1.144154	1.543744375

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.

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. Furthermore, as the average interval time between arrivals (**lambda**) increases, the percentage of missed events falls (Figure 2). This is because here is more time for the program to react to new events that arrive before another event comes after it (events are being generated slowly)

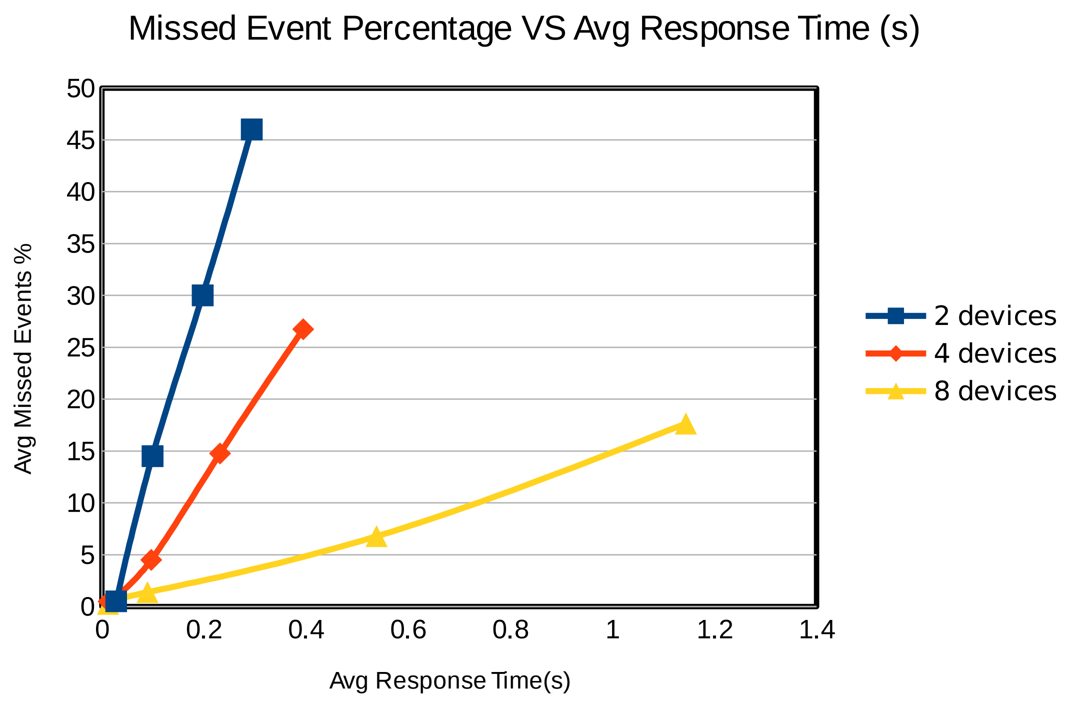

Figure 1

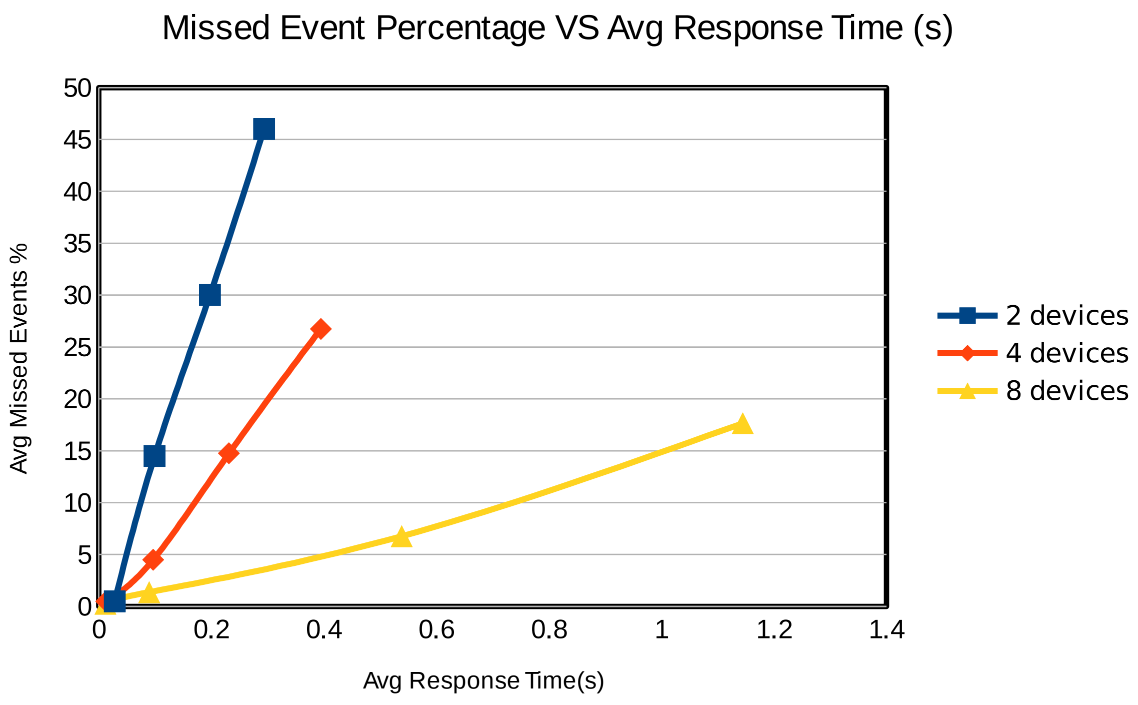

Figure 2

In the case of Round Robin in a single threaded environment, turnaround time exhibits the same trend as response time. As shown in figure 3, longer turnaround time leads to more missed events because the program is unable to react to new events for a longer period of time.

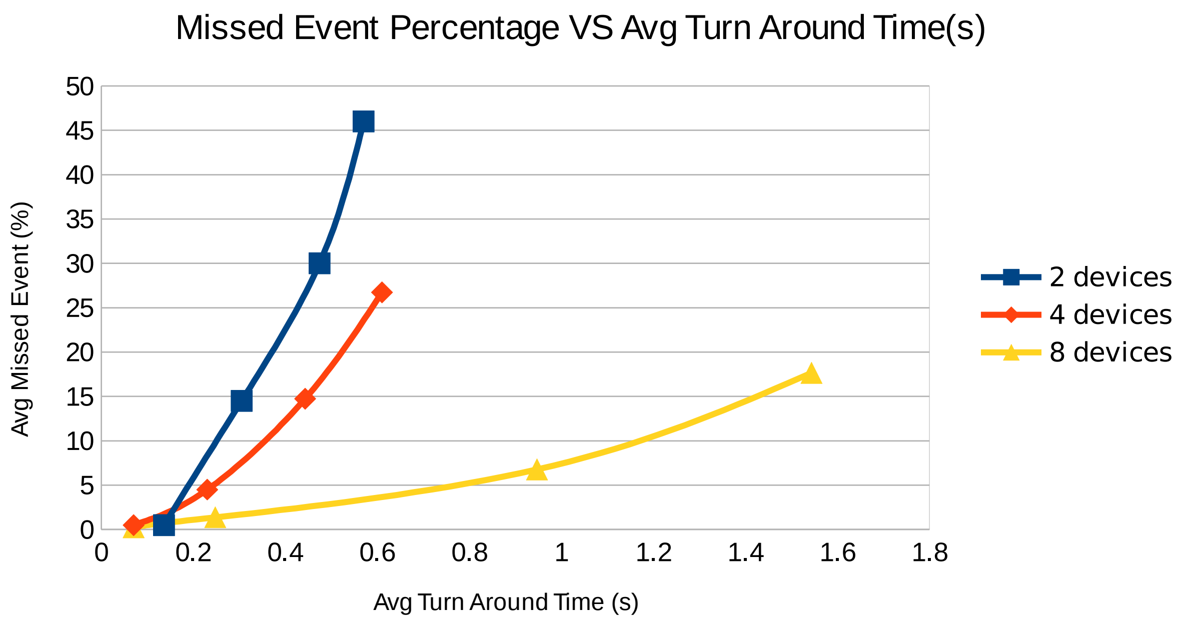

Figure 3

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. If we are limited by Round Robin, we can only decrease the number of missed events by reducing turnaround time for each event in a single threaded environment. If we implement multi threading with Round Robin, we could reduce response time since we will be able to respond to events while other events are being served. These are all strategies in addition to improving program efficiency and reduce the time it takes to complete a cycle.