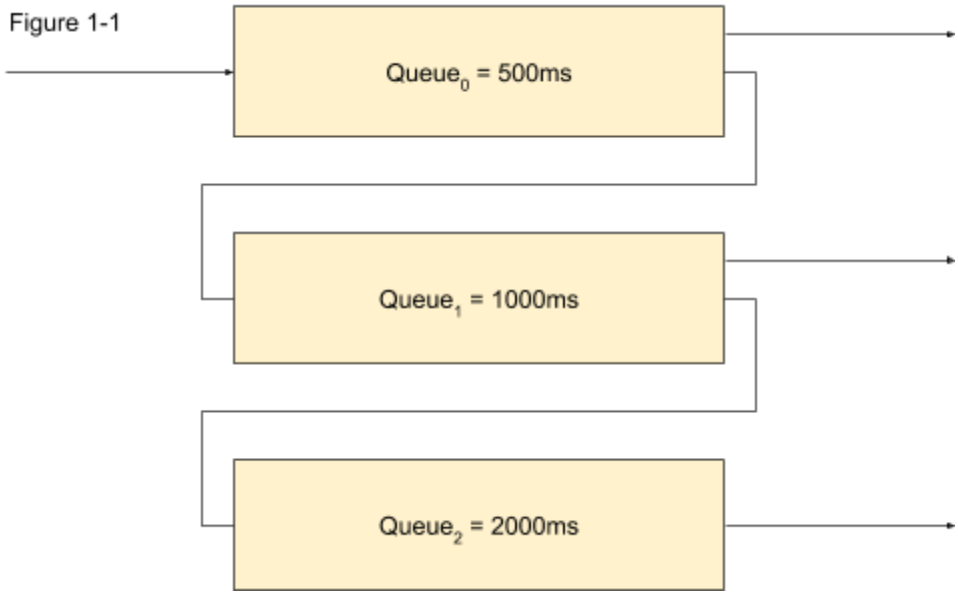
Margaret Connor
CSS 430 Spring 2019
P2 Report

To achieve a multilevel feedback queue I first created 3 vectors to act as individual queues and keep track of which thread a queue is in. At the beginning of the run() function it checks each queue beginning with $Queue_0$ if it contained a thread to be run it would select the first thread in the queue, otherwise it would move on to the next queue. The program would then resume() the process and enter a for loop that would sleep for 500ms. The loop looped $2^{<queue\ num>}$ times, so in case $Queue_0$ it would loop $2^0 = 1 * 500ms = 500ms$ in the case of $Queue_2$ it would loop $2^2 = 4 * 500ms = 2000ms$. After each loop (every 500ms) it would check the other queues. If the queue has a job in it it would break out of the loop, suspend the current job, and beginning the job in the highest queue. Figure 1-1 illustrates this movement of thread.

Figure 1-1



Bellow in table 1-1 and 1-2 are the results of my testing on the schelder using round robin and multilevel feedback queues . As we can see from the tables the average response time, turn around time, and execution time is better in the multilevel feedback queue. The multilevel feedback queue is better for a few reasons. To begin less context switches occur in the Multilevel feedback queue, since context switching is expensive this helps save time by limiting the number of switches, by counting the number of times a letter changes in figure 2-1 and 2-2 we see that MFQ (Multilevel feedback queue) performs one less context switch then the round robin model. MFQ also gives preference to shorter jobs, making those jobs more likely to complete earlier than in

round robin scheduling. Finally the MFQ is pre-emptive allowing all the processes to get at least a little but of processing time at the beginning, this also favors short jobs because it allows them to get in and out faster.

Table 1-1: Round Robin

	Response Time	Turn around Time	Execution Time
Trial 1			
Thread[e]:	5986	6486	500
Thread[b]:	2985	9986	7001
Thread[c]:	3985	20988	17003
Thread[a]:	1984	28989	27005
Thread[d]:	4986	32989	28003
Trail 1 Avg.	3985.2	19887.6	15902.4
Trail 2			
Thread[e]:	6020	6537	517
Thread[b]:	3009	10033	7024
Thread[c]:	4012	21073	17061
Thread[a]:	2006	29095	27089
Thread[d]:	5017	33101	28084
Trial 2 Avg.	4012.8	19967.8	15955
Total Avg	**3999**	**19927.7**	**15928.7**

Table 1-2: Multilevel feedback queue

	Response Time	Turn around Time	Execution Time
Trial 1			
Thread[b]:	980	5482	4502
Thread[e]:	2481	7982	5501
Thread[c]:	1480	15985	14505
Thread[a]:	480	23987	23507
Thread[d]:	1981	30987	29006
Trial 1 Avg	1480.4	16884.6	15404.2
Trial 2			
Thread[b]:	1003	5543	4540
Thread[e]:	2515	8053	5538
Thread[c]:	1506	16137	14631
Thread[a]:	500	24182	23682
Thread[d]:	2010	31173	29163
Trail 2 Avg.	1506.8	17017.6	15510.8
Total Avg.	**1493.6**	**16951.1**	**15457.5**

If my multilevel feedback queue's final queue ($Queue_2$) was first come first serve, instead of a 2000ms pre-emptive queue, I've illustrated how the processes would get CPU time in figure 2-3. Counting the number of context switches, the MFQFCFS (Multilevel Feedback Queue First Come First Serve) would have significantly less than the round robin scheduling system and a few less than the normal MFQ. However, the scheduler would lose its pre-emptive quality. In the example of figure 2 the MFQFCFS would perform faster because of the less context switch, in practice you may get cases where it does not perform faster because a process in the final queue hogs the CPU. This can be mitigated by using a pre-emtive FCFS queue instead.

Figure 2-1: Round Robin Gantt chart

Figure 2-2: Multilevel feedback queue

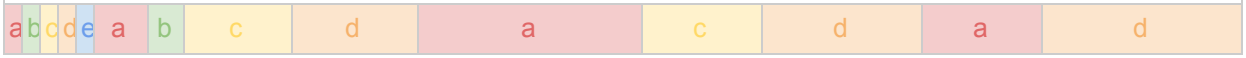


Figure 2-3: Multilevel Feedback Queue with First Come First Serve