Ethan Crawford

CSS 430

10/23/2008

Program 2

# Part 1: Round-Robin Scheduler

## Execution output:

Thread[e]: response time = 6005 turnaround time = 6514 execution time = 509

Thread[b]: response time = 2999 turnaround time = 10012 execution time = 7013

Thread[c]: response time = 4001 turnaround time = 21034 execution time = 17033

Thread[a]: response time = 1998 turnaround time = 29052 execution time = 27054

Thread[d]: response time = 5003 turnaround time = 33062 execution time = 28059

# Part 2: Multilevel Feedback Queue Scheduler

## Execution output:

Thread[e]: response time = 2582 turnaround time = 3092 execution time = 510

Thread[b]: response time = 1032 turnaround time = 5576 execution time = 4544

Thread[c]: response time = 1549 turnaround time = 13732 execution time = 12183

Thread[a]: response time = 516 turnaround time = 21386 execution time = 20870

Thread[d]: response time = 2065 turnaround time = 28021 execution time = 25956

# Part 3: Report

## Algorithm

1. On construction, the Scheduler creates a List of 3 JobQueue objects, assigning a job quantum (maximum execution time) to each.
2. When the Kernel adds a job to the Scheduler, the remainingQuantum is assigned by the JobQueue to the Job.
3. Scheduler runs in an endless loop, checking each queue, in priority order, for jobs.
4. When a job is found, it is executed and the Scheduler sleeps for sleepTime. If, after sleeping, the job has not completed, the Scheduler decrements the job’s remainingQuantum by sleepTime.
5. When a Job’s remainingQuantum goes to 0, the job is demoted to a lower-priority queue. When no lower-priority queues are available, the job is added to the end of the lowest-priority queue (see figure 1)

500 MS quantum

JobQueue 0

1000 MS quantum

JobQueue 1

2000 MS quantum

JobQueue 2

Figure 1: Multilevel feedback queue

## Comparison of part 1 and part 2 test results

### Response Time

Round-robin (RR) and Multilevel Feedback Queue (MFQ) response times are comparable. Each switches on a sleep interval, defined in RR as 1000 MS and in MFQ as 500 MS. RR scheduling shows a delay of 2000 MS prior to starting execution on thread A, while MFQ shows a delay of 500 MS.

### Turnaround Time

Compared to RR, MFQ scheduling shows consistently better turnaround time. As expected, the best MFQ turnaround time is seen in threads with shorter CPU bursts (about twice as fast as RR). Longer burst threads show turnaround times more comparable to RR, although even in the worst case, there is a 5 second improvement.

### Execution Time

Overall execution time for MFQ scheduling is faster, owing to the algorithm’s superior turnaround time.

### Summary

MFQ appears to be a significantly more efficient scheduling algorithm in cases where jobs have variable CPU burst increments. By biasing execution time to jobs that consume fewer CPU cycles, the scheduler is able to more efficiently allocate CPU cycles-per-context-switch.

## Consideration of implementing final queue as FCFS

Implementing the final queue as first-come first-served (FCFS) increases the risk of thread starvation caused by a long-running job refusing to relinquish the CPU . It also potentially provides for maximum efficiency, due to the elimination of Scheduler-driven context switching. In the case of ThreadOS, the Kernel, via threads that will never exit, provides services which are scheduled by the Scheduler. This means that FCFS scheduling for the final queue is not a viable alternative for this project, since only the first non-exiting thread in the queue will ever get any attention, leading to starvation for the rest.