University of Newcastle
School of Electrical Engineering and Computer Science

COMP2240 - Operating Systems
Workshop 3 - Solution
Topics: Scheduling

1. Consider a variant of the RR scheduling algorithm where the entries in the ready queue are pointers to the PCBs.
   a) What would be the effect of putting two pointers to the same process in the ready queue?
   b) What would be the major advantage/disadvantage of this scheme?
   c) How could you modify the basic RR algorithm to achieve the same effect without the duplicate pointers?

Answer:
   a) The process would be run twice as many times.
   b) The advantage is that more important jobs could be given more time, in other words, higher priority in treatment. The consequence, of course, is that shorter jobs will suffer.
   c) Allot a longer amount of time to processes deserving higher priority, on other words, have two or more quantums possible in the RR scheme.

2. An interactive system using round-robin scheduling and swapping tries to give guaranteed response to trivial requests using the following algorithm: After completing a round-robin cycle among all Ready processes, the system determines the time slice to allocate to each Ready process for the next cycle by dividing the maximum response time by the number of processes requiring service. Will this work in practice?

Answer:
It will work only as long as there are comparatively few users in the system. When the quantum is decreased to satisfy more users rapidly two things happen:
   a) processor utilisation decreases, and
   b) at a certain point, the quantum becomes too small to satisfy most trivial requests.
Users will then experience a sudden increase of response times because their requests must pass through the round-robin queue several times.

3. The following processes are being scheduled using a preemptive, round robin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority.
In addition to the processes listed below, the system also has an ***idle task*** (which consumes no CPU resources and is identified as $\mathbf{P}_{\mathbf{idle}}$). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units.

Thread	Priority	Burst	Arrival
P1	40	20	0
P2	30	25	25
P3	30	25	30
P4	35	15	60

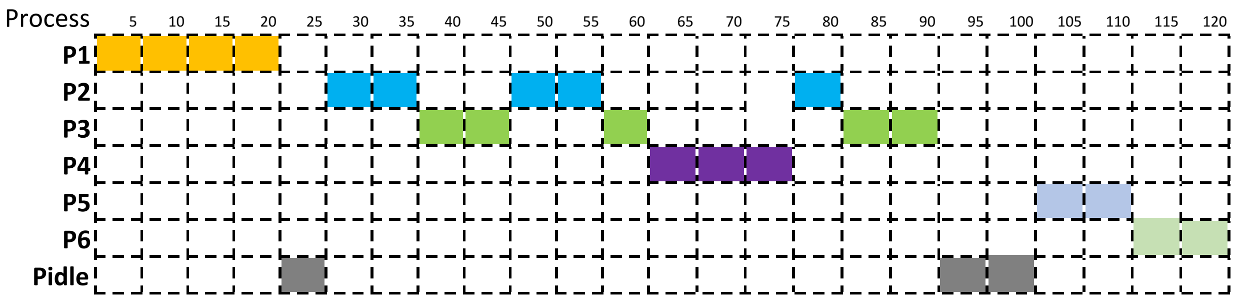
P5	5	10	100
P6	10	10	105

If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

a) Show the scheduling order of the processes using a Gantt chart.
b) What is the turnaround time for each process?
c) What is the waiting time for each process?
d) What is the CPU utilization rate?

Answer:

a) Gantt Chart:



b) Turnaround time:
P1: 20-0 = 20
P2: 80-25 = 55
P3: 90 - 30 = 60
P4: 75-60 = 15
P5: 120-100 = 20
P6: 115-105 = 10

c) Waiting Time:
P1: 0
P2: 40
P3: 35
P4: 0
P5: 10
P6: 0

d) CPU Utilisation Rate: 105/120 = 87.5 %

4. A variation of preemptive priority scheduling has dynamically changing priorities. A new process is assigned a priority of 0. While a process is in the ready queue, its priority changes at the rate of α; and while it is executing, its priority changes at the rate of β. The different values of α and β give different algorithms. If larger values imply higher priorities, state with reasons the type of algorithm that will result if:

a) $\beta > \alpha > 0$
b) $\alpha < \beta < 0$

Answer:

a) Given β > α > 0, when α and β have positive values, the priorities of processes increase with the time they have been in the system. Thus, the process entering the system first will get the CPU first. Also, since the rate of increase of priority of a running process is greater than the rate of increase of priority of a waiting process, a process that has substantial CPU time will get the CPU again because of its increased priority. Hence, eventually, first come-first-served scheduling will occur.

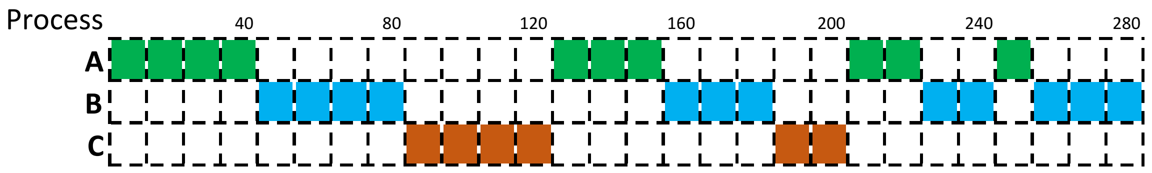
b) Given α < β < 0, since both α and β are negative, the priorities decrease with time. The new process entering the system has the highest priority and so it preempts the running process and starts executing. Further, the rate of decrease of priority in the ready queue is more than the rate of decrease of priority for the running process. Hence, the processes that have just arrived in the ready queue have greater priority over processes that were in the queue earlier. Processes that have got the CPU time have greater priority over processes that have not. All these observations imply that last-in-first-out scheduling will be observed in this case.

5. Consider a variant of Round Robin Scheduling, say NRR scheduling. In NRR scheduling, each process can have its own time quantum, *q*. The value of *q* starts out at 40 ms and decreased by 10 ms each time it goes through the round robin queue, until it reaches a minimum of 10 ms. Thus, long jobs get decreasingly shorter time slices.

Analyse this scheduling algorithm for three jobs A, B and C that arrive in the system having estimated burst times of 100 ms, 120 ms, and 60 ms respectively. Also identify some advantages and disadvantages that are associated with this algorithm.

Answer:

Burst time of process A = 100 ms
Burst time of process B = 120 ms
Burst time of process C = 60 ms
Analysis of system with NRR scheduling:
GANTT chart:



Turnaround time for A = 240 ms
Turnaround time for B = 280 ms
Turnaround time for C = 200 ms
Average turnaround time = 720 / 3 = 240 ms

Waiting time for A = 240 − 100 = 140 ms
Waiting time for B = 280 − 120 = 160 ms
Waiting time for C = 200 − 60 = 140msec
Average waiting time = 440 / 3 = 146.67 ms

In this system, the CPU time of newer processes entering the system is greater. Thus, processes that are interactive and I/O bound get more CPU time than CPU bound processes. Hence, a relative priority that prefers processes with less CPU time is automatically set. The turnaround time of these processes is reduced as they waste less time in context switches, which is an advantage over the normal round-robin scheduling.

The two disadvantages of this system are: **1**. it requires the overhead of an added logic in each process to maintain its time quantum; **2**. if there is a steady influx of short processes in the system, a long job may have to wait for a very long time for its completion.

6. Suppose that a scheduling algorithm (at the level of short-term CPU scheduling) favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs?

Answer:
It will favor the I/O-bound programs because of the relatively short CPU burst request by them; however, the CPU-bound programs will not starve because the I/O-bound programs will relinquish the CPU relatively often to do their I/O.

7. Android operating system uses the CFS (Completely Fair Scheduler) algorithm implemented at its Linux Kernel. The default scheduling algorithm in Linux 2.6.23 is the completely fair scheduler (CFS). As the documentation to the kernel states, 80% of CFS design can be summed up in a single sentence: CFS basically models an "ideal, precise multitasking CPU" on real hardware.

The general principle behind this scheduler is to provide maximum fairness to each task in the system in terms of the computation power it is given. More precisely, this means that when the CFS makes a scheduling decision, it always selects processes that have been waiting for the longest amount of time. In android system, with *n* running processes, each process would be having *1/n* amount of CPU-time while running constantly – in any measurable period all the tasks would have run for the same amount of wall-clock time.

A nice value in the range from -20 to +19 is assigned to each task of android process. Lower nice value indicates a higher relative priority and processes with lower nice values receive a higher proportion of CPU processing time than tasks with higher nice values. The default nice value is 0. A nice value is mapped to a weight value (W_i).

CFS identifies a targeted latency (TL), which is an interval of time during which every runnable task should run at least once.

$$\text{Time slice for a task i} = TL * W_i/(\text{Sum of all } W_i)$$

Calculate the time slice of two android processes t1 and t2 with nice values of 0 (weight value is 1024) and 5 (weight value is 335) respectively. Where the targeted latency of the system is 20 ms.

Answer:

a) Targeted latency = 20 ms

The share for task t1 is 1024/(1024+335) = 0.75
Task t1 will have a "time slice" of 0.75*20 = 15 ms

The share for task t2 is 335/(1024+335) = 0.25
Task t2 will have a "time slice" of 20*.25 = 5 ms

Supplementary problems:

S1 Which type of process is generally favoured by a multi-level feedback queuing scheduler: a processor-bound process or an I/O-bound process? Briefly explain why?

Answer:
Multi-level feedback quieting scheduler favours I/O bound processes over processor-bound process.
If a process uses too much processor time it will be moved to a lower-priority queue. This leaves I/O-bound processes in the higher-priority queue. UNIX System V (Chapter 9 Section 9.3) is an example of this scheme.

S2 Contrast the scheduling policies you might use when trying to optimize a time-sharing system with those you would use to optimize a *multiprogrammed* batch system.
Answer:
With time sharing, the concern is response time. Time-slicing is preferred because it gives all processes access to the processor over a short period of time. In a batch system, the concern is with throughput, and the less context switching, the more processing time is available for the processes. Therefore, policies that minimize context switching are favored.

S3. Explain how time quantum value and context switching time affect each other, in a round-robin scheduling algorithm.

Answer:
If the context switching time is large, then the time quantum value has to be proportionally large. Otherwise, the overhead of context switching can be quite high. Choosing large time quantum values can lead to an inefficient system if the typical CPU burst times are less than the time quantum.
If context switching is very small or negligible, then the time quantum value can be chosen with more freedom.