CS341: Operating System

Process Scheduling

Lect12 : 27th Aug 2014

Dr. A. Sahu
Dept of Comp. Sc. & Engg.
Indian Institute of Technology Guwahati

1

Outline

- Process Concepts Recap
  - Process States, PCB, Context Switch
  - **Job Queue, Ready Queue, De**
- Scheduling
  - **System Oriented and Theoretical Analysis**
  - **Introduction to Scheduling Algorithms**
  - **Real Time Scheduling Algorithms**
  - **Multiprocessor Scheduling Algorithms**
  - **Distributed and Power Aware Scheduling**

2

Scheduling Algorithm Optimization Criteria

- Maximize
  - CPU utilization
  - Throughput
- Minimize
  - Turnaround time
  - Waiting time
  - Response time

Assumption

- Task/Process/Job used interchangbly
- Let all tasks in memory
- Let all tasks don't do any IPC
- All task are independent
- All tasks don't require any other resource other then CPU

Five Popular Scheduling

- First Come First Serve
- Shortest Job First
- Shortest Remaining Time First
  - SJF-I
- Round-Robin Scheduler
- Priority Scheduler
- Priority-I
- Multi-Level Priority Queue
- Feed Back Priority Queue

5

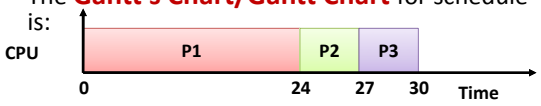
Problem Cases

- Blindness about job types
  - I/O goes idle
- Optimization involves favoring jobs of type "A" over "B".
  - Lots of A's? B's starve
- Interactive process trapped behind others.
  - Response time sucks for no reason
- Priority Inversion: A depends on B. A's priority > B's.
  - B never runs

FCFS Scheduling

Process	Burst Time/ Execution Time
P1	24
P2	3
P3	3

- Assume: processes arrive in the order: P_1 , P_2 , P_3 The **Gantt's Chart/Gantt Chart** for schedule is:



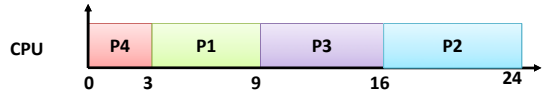
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- **SJF is optimal** – gives minimum average waiting time for a given set of processes
  - **The difficulty is knowing the length of the next CPU request**
  - Could ask the user

Example of SJF

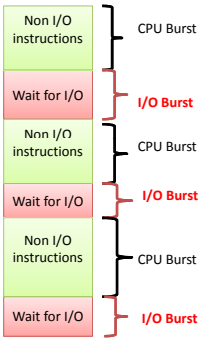
Process	Burst Time
P1	6
P2	8
P3	7
P4	3



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging



Prediction: Next CPU Burst

- Can be done by using the length of previous CPU bursts, using exponential averaging
  - t_n is actual CPU Burst of nth CPU Burst
  - τ_{n+1} = predicted values of next cpu burst
  - α, 0 ≤ α ≤ 1

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\, \tau_n$$

- Commonly, α set to ½
- Preemptive version called **shortest-remaining-time-first**

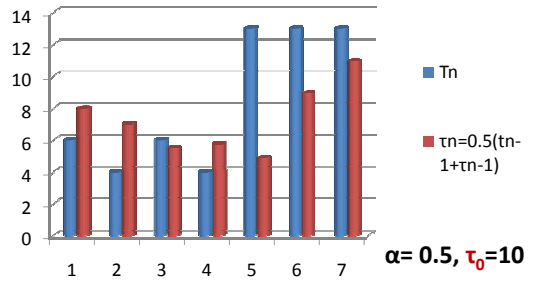
Examples of Exponential Averaging

- $\alpha = 0$: $\tau_{n+1} = \tau_n$, Recent history does not count
- $\alpha = 1$: $\tau_{n+1} = \alpha\, t_n$, Only actual last CPU burst counts
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\alpha\, t_{n-1} + \ldots$$
$$+ (1-\alpha)^j \alpha\, t_{n-j} + \ldots$$
$$+ (1-\alpha)^{n+1} \tau_0$$

- Since both α and (1 - α) are less than or equal to 1, each successive term has less weight than its predecessor

$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\, \tau_n$



$\alpha = 0.5,\ \tau_0 = 10$

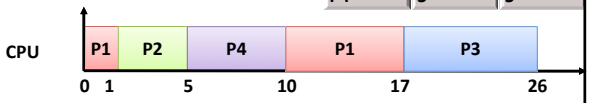
$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\, \tau_n$ with $\alpha = 0.5$

13

SJF-Preemptive: Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis
- **Dynamic Decision @Runtime**

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5



- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

Priority Scheduling

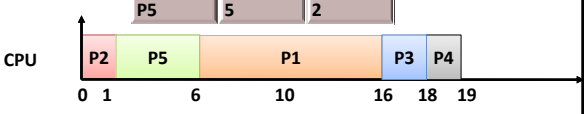
- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority
  - Smallest integer == > highest priority
  - Preemptive, Non-preemptive
- SJF is priority scheduling
  - Where priority is 1/CPU Burst time

Starvation: Priority Scheduling

- **Starvation Problem**
  - low priority processes may never execute
  - Starve to execute…
- **Solution**
  - **Aging** – as time progresses increase the priority of the process

Example of Priority Scheduling

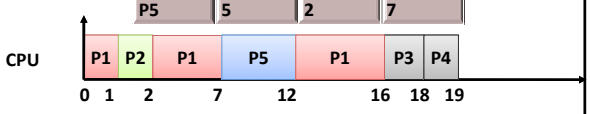
Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2



- Assume all arrived@ time 0
- Average waiting time (6+0+16+18+1=41)/ =8.2 msec

Priority Scheduling Preemptive

Process	Burst Time	Priority	Arrival Time
P1	10	3	0
P2	1	1	1
P3	2	4	6
P4	1	5	6
P5	5	2	7



- Assume all arrived@ time 0
- Average waiting time (6+0+16+18+7=47)/5 =9.4 msec

Round Robin (RR)

- Each process gets a small unit of CPU time (**time quantum *q***)
  - –Usually 10-100 milliseconds.
  - –After this time has elapsed, the process is preempted and added to the end of the ready queue.

Round Robin (RR)

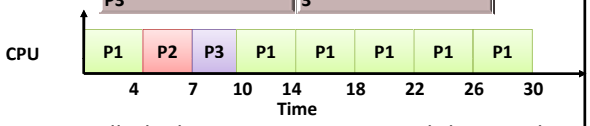
- If there are ***n*** processes in the ready queue and the time quantum is ***q***, then
  - –Each process gets **1/*n*** of the CPU time in chunks of at most ***q*** time units at once.
- No process waits more than **(*n*-1)*q*** time units.

Round Robin (RR)

- Timer interrupts every quantum to schedule next process
  - **Timer: Hardware unit, similar to Alarm**
- Performance
  - – *q* large ⇒ FIFO
  - – *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high

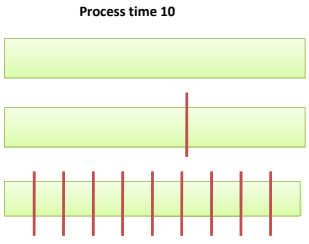
Example : RR (with q = 4)

Process	Burst Time
P1	24
P2	3
P3	3



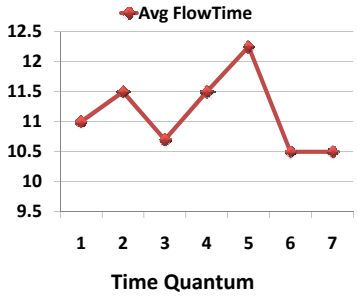
- Typically, higher Average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Time Quantum and Context Switch Time



Quantum	Context Switches
12	0
6	1
1	9

Turnaround Time Varies With The Time Quantum



Process	Time
P1	6
P2	3
P3	1
P4	7

80% of CPU bursts should be shorter than q

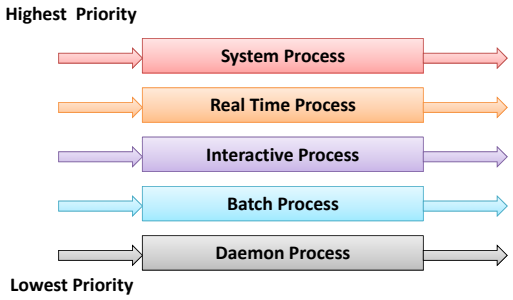
Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
  - foreground (interactive)
  - background (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS

Multilevel Queue

- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). **Possibility of starvation.**
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes;
  - i.e., 80% to foreground in RR, 20% to background in FCFS

Multilevel Queue Scheduling

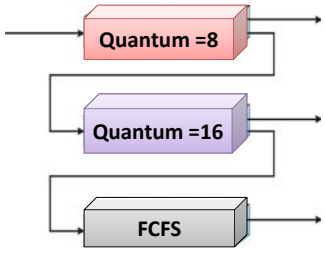


Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

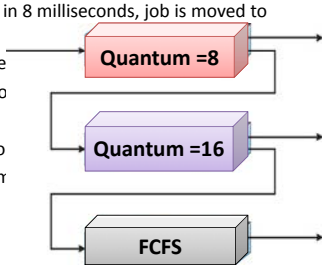
Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS



Example of Multilevel Feedback Queue

- Scheduling
  - A new job enters queue Q_0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again serve and receives 16 additio milliseconds
    - If it still does not co it is preempted and m to queue Q_2



Introduction to Scheduling Algorithms

A bit of Theoretical View

31

Parallel Machine Problems

- For **identical machines** $M_1, \dots, M_m$
  - The processing time for ***j*** is the same on each machine.
- For **unrelated machines**
  - The processing time p_{jk} depends on the machine M_k on which j is processed.

Parallel Machine Problems

- For **uniform machine**
  - if $p_{jk} = p_j/r_k$.
- For **multi-purpose machines**
  - A set of machines μ_j is is associated with each job j indicating that j can be processed on one machine in μ_j only.

Example: Machine Environment

	M1	M2	M3
P1	5	5	5
P2	8	8	8
P3	6	6	6

Identical

	M1	M2	M3
P1	5	4	6
P2	9	8	4
P3	3	18	4

Unrelated

	M1	M2	M3
P1	5	5/1.5	5/2
P2	9	9/1.5	9/2
P3	9	9/1.5	9/2

Uniform

34

Classification of Scheduling Problems

Classes of scheduling problems can be specified in terms of the three-field classification

$$\alpha \mid \beta \mid \gamma$$

where

- α specifies the **machine environment**,
- β specifies the **job characteristics**, and
- γ describes the **objective function(s).**

Machine Environment

- 1 single machine
- P parallel identical machines
- Q uniform machines
- R unrelated machines
- MPM multipurpose machines, J job-shop,
- F flow-shop O open-shop

If the number of machines is fixed to m we write **Pm, Qm, Rm**, MPMm, Jm, Fm, Om.