操作系统原理与设计

第5章 CPU Scheduling (1)

陈香兰

中国科学技术大学计算机学院

2009年09月01日

CPU scheduling is the basis of multiprogrammed OSes.

提纲——CPU scheduling (1)

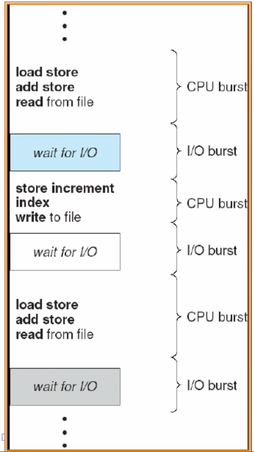
Objective of multiprogramming

- Maximum CPU utilization

Outline

CPU-I/O Burst Cycle I

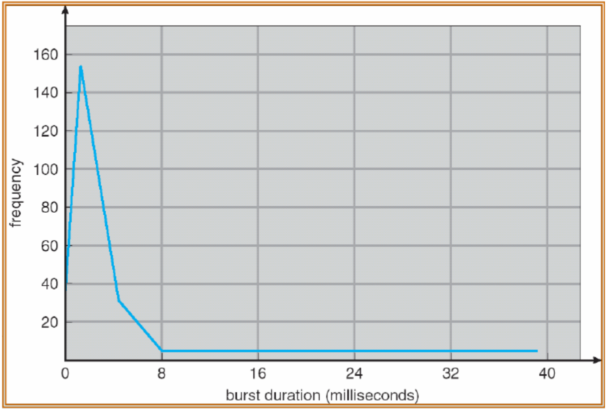
- A property of process :
  CPU-I/O Burst Cycle
  - Process execution consists of a cycle of CPU execution and I/O wait
    - Process execution
      = n (CPU execution + I/O wait)
- Alternating Sequence of CPU And I/O Bursts



CPU-I/O Burst Cycle II

- CPU burst distribution
  - Histogram of CPU-burst Times

CPU-I/O Burst Cycle III



Outline

CPU Scheduler

- CPU scheduler (Short-term Scheduler)
  - Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- Ready Queue:
  - FIFO Queue?
  - by priority?
  - tree?
  - unordered linked list?

Outline

Preemptive Scheduling I

- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- The scheduling scheme:
  - nonpreemptive: only 1 & 4
    - Windows 3.x
    - before Mac OS X
  - otherwise preemptive (usually needs a hardware timer, synchronization overhead)
    - Windows 95 & ...

Preemptive Scheduling II

- Mac OS X

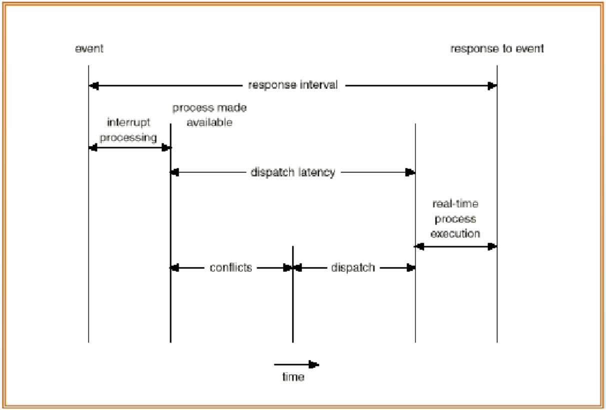
preemptive kernel VS. nonpreemptive kernel?
Interrupt affected code VS normal kernel code?

Outline

Dispatcher

- Dispatcher module **gives control of the CPU to the process selected by the short-term scheduler**; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running
  - must be short

Dispatch latency



Outline

Scheduling Criteria I

- **CPU utilization** （CPU 利用率）– keep the CPU as busy as possible
  - conceptually: $0\% \sim 100\%$
  - in a real system: $40\% \sim 90\%$
- **Throughput** （吞吐率）– $\#$ of processes that complete their execution per time unit
  - different from one process set to another process set
  - for long processes: may be 1 process per hour
  - for short transactions: may be 10 processes per second

Scheduling Criteria II

- **Turnaround time** (周转时间)– amount of time to execute a particular process
  - from the time of submission of a process to the time of completion
    = the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Optimization Criteria

- Maximize?
  - CPU utilization
  - throughput
- Minimize?
  - turnaround time
  - waiting time
  - response time
- Average?
- Stability?

different from system to system.

Outline

FCFS Scheduling

- First-Come, First-Served
  - nonpreemptive
- Implementation:
  - Normal Queue: FIFO Queue
  - ordered by request time
  - linked list
  - Insert: linked to the tail of the queue
  - scheduling: removed from the head of the queue

Example of FCFS Scheduling

- Suppose that the processes arrive in the order:

Process	BurstTime(ms)
P1	24
P2	3
P3	3

P1 , P2 , P3

The Gantt Chart for the schedule is:

- Waiting time for P1 $= 0$; P2 $= 24$; P3 $= 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

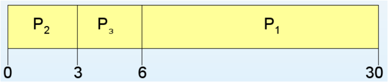
Example of FCFS Scheduling II

- Suppose that the processes arrive in the order

P2 , P3 , P1

Process	BurstTime(ms)
P1	24
P2	3
P3	3

- The Gantt chart for the schedule is:



- Waiting time for P1 = 6; P2 = 0; P3 = 3
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case

Convoy effect

- example situation:
  - one CPU-bound process
  - many I/O-bound processes
- **Convoy effect （护航效应；护卫效应）：**
  - all the other processes wait for the one big process to get off the CPU
  - ≡short process behind long process

Outline

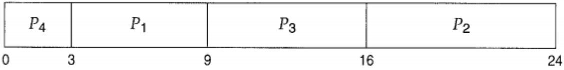
SJF Scheduling

- Shortest-Job-First (shortest-next-CPU-burst algorithm)
  - Associate with each process **the length of its next CPU burst**.
  - Use these lengths to **schedule the process with the shortest time**.

SJF scheduling example

Process	BurstTime(ms)
P1	6
P2	8
P3	7
P4	3

- The Gantt chart for the schedule is:



- Waiting time for P1 = 3; P2 = 16; P3 = 9; P4 = 0
- Average waiting time: $(3 + 16 + 9 + 0)/4 = 7$
- If FCFS, average waiting time would be: $(0 + 6 + 14 + 21)/4{=}10.25$

- SJF is **optimal** – gives **minimum average waiting time** for a given set of processes

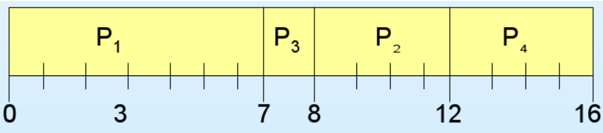
SJF scheduling schemes

- Two schemes:
  - **nonpreemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst
  - **preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as **the Shortest-Remaining-Time-First (SRTF)**

Example of Non-Preemptive SJF

Process	ArrivalTime	BurstTime(ms)
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

- SJF (non-preemptive)

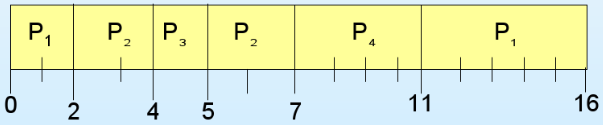


- Average waiting time $= (0 + 6 + 3 + 7)/4 = 4$

Example of Preemptive SJF

Process	ArrivalTime	BurstTime(ms)
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

- SJF (preemptive)



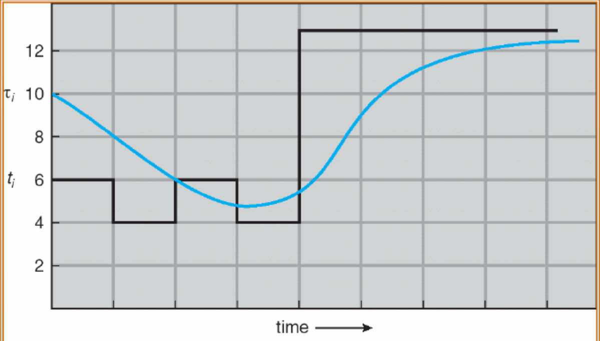
- Average waiting time = ((11 - 2) + (5 - 4) + 0 +(7 - 5))/4 = 3

Determining Length of Next CPU Burst

- For job scheduling
  - depend on user?
- For CPU scheduling
  - **Can only estimate** the length
- Can be done by using the length of previous CPU bursts, using exponential averaging
  1. t_n =actual length of n^{th} CPU burst
  2. τ_{n+1}= predicted value for the next CPU burst
  3. α, $0 \leq \alpha \leq 1$
  4. Define:
     $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$

Prediction of the Length of the Next CPU Burst

- $\alpha = 1/2;\ \tau_0 = 10$



CPU burst (t_i)		6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\tau_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:
  $\tau_{n+1} =$
  $\alpha\tau_n + (1-\alpha)\,\alpha\tau_{n-1} + \cdots + (1-\alpha)^j\,\alpha\tau_{n-j} + \cdots + (1-\alpha)^{n+1}\,\tau_0$
- Since both α and $(1-\alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Outline

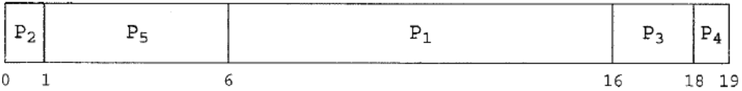
Priority Scheduling

- A priority number (integer) is associated with each process
  - priority number VS. priority
- The CPU is allocated to the process with the highest priority (smallest integer $\equiv$ highest priority)
  - Preemptive
  - nonpreemptive
- **SJF is a special case** of general priority scheduling where priority is the predicted next CPU burst time

Example

Process	BurstTime(ms)	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	4	2

- The Gantt chart for the schedule is:



- Average waiting time $= (6 + 0 + 16 + 18 + 1)/5 = 8.2$

- The determination of priority
  - **internally**, for example:
    - time limits, memory requirement, the number of open files, ...
  - **externally**, for example:
    - the importance, the type and amount of funds, the department, ...
- Priority Scheduling:
  - Problem $\equiv$ **Starvation** (indefinite blocking) – low priority processes may never execute
  - Solution $\equiv$ **Aging** – as time progresses increase the priority of the process

Outline

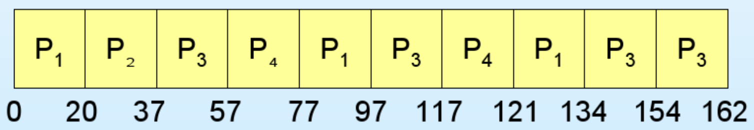
Round Robin (RR)

- Time quantum, time slice
  - a small unit of CPU time
  - usually 10-100 ms
- Implementation
  - Ready queue: a FIFO circular queue
  - Each process gets 1 time quantum
  - Insert: to the tail of the queue
  - Scheduling: pick the first process; set timer; and dispatch
  - two situation:
    - CPU burst $\leq$1 time quantum
    - CPU burst $>$ 1 time quantum. After this time has elapsed, the process is **preempted** and added to the end of the ready queue.

Example of RR with Time Quantum = 20

Process	BurstTime
P1	53
P2	17
P3	68
P4	24

- The Gantt chart is:



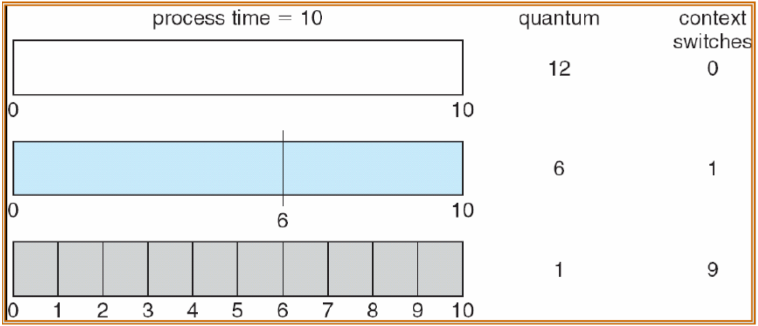
- Typically, higher average turnaround than SJF, but better response

Performance

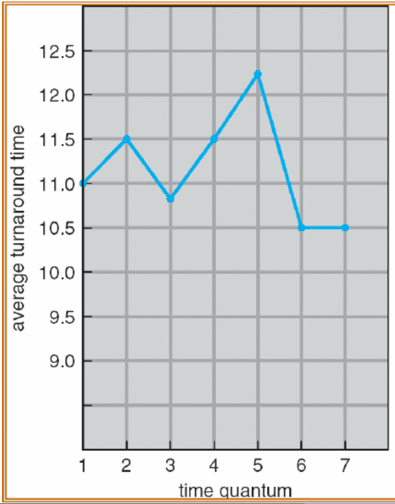
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than (n-1)q time units.
- Performance
  - q large $\Rightarrow$FIFO
  - q small $\Rightarrow$q must be large with respect to context switch, otherwise overhead is too high

Time Quantum and Context Switch Time

- The effect of context switching on the performance of RR scheduling



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Outline

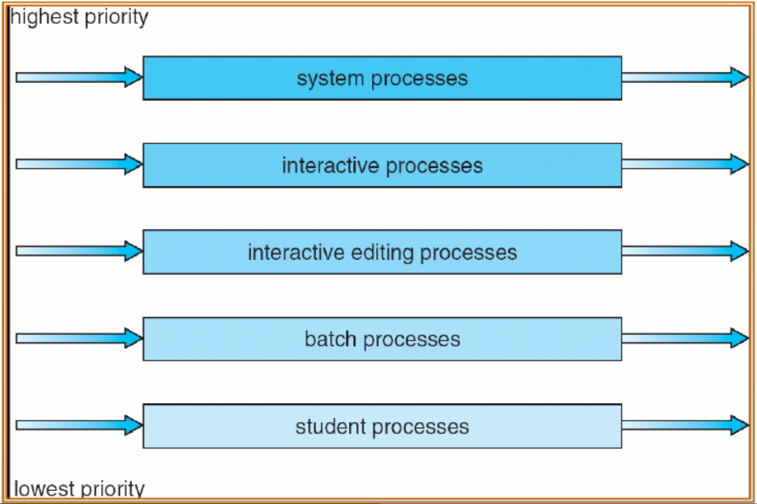
Multilevel Queue Scheduling I

- Ready queue is partitioned into separate queues:
  - foreground (interactive)
  - background (batch)
- Each queue has its own scheduling algorithm
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues
  - **Fixed priority scheduling**;
    - i.e., serve all from foreground then from background
    - **Possibility of starvation**.

Multilevel Queue Scheduling II

- **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

example


highest priority
system processes
interactive processes
interactive editing processes
batch processes
student processes
lowest priority

Outline

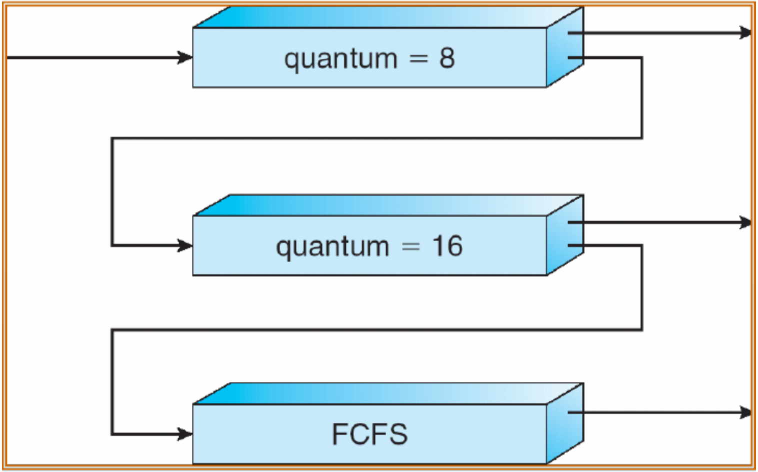
Multilevel Feedback Queue Scheduling

- 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 I

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- 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 served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.

Example of Multilevel Feedback Queue II



小结

1. Basic Concepts
   - CPU-I/O Burst Cycle
   - CPU Scheduler
   - Preemptive Scheduling
   - Dispatcher
2. Scheduling Criteria
   - Scheduling Criteria
3. Scheduling Algorithms
   - FCFS Scheduling
   - SJF Scheduling
   - Priority Scheduling
   - Round Robin Scheduling
   - Multilevel Queue Scheduling
   - Multilevel Feedback Queue Scheduling
4. 小结和作业

作业

- ?

谢谢！