Department of Electrical Engineering, Feng-Chia University

Chapter 6: CPU Scheduling

王振傑 (Chen-Chieh Wang)
ccwang@mail.ee.ncku.edu.tw

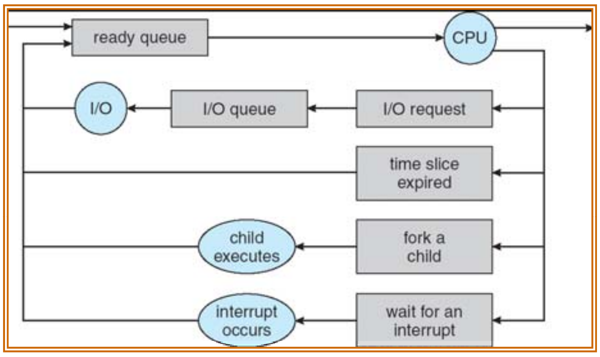
Department of Electrical Engineering, Feng-Chia University

Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Other Issues

Department of Electrical Engineering, Feng-Chia University

CPU Scheduling



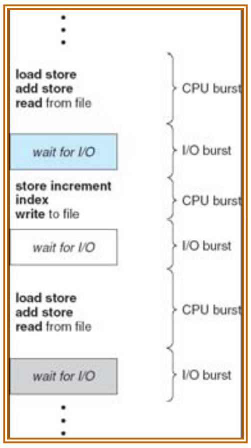
- Earlier, we talked about the life-cycle of a thread
  - Active threads work their way from Ready queue to Running to various waiting queues.
- Question: How is the OS to decide which of several tasks to take off a queue?
  - Obvious queue to worry about is ready queue
  - Others can be scheduled as well, however
- Scheduling: deciding which threads are given access to resources from moment to moment

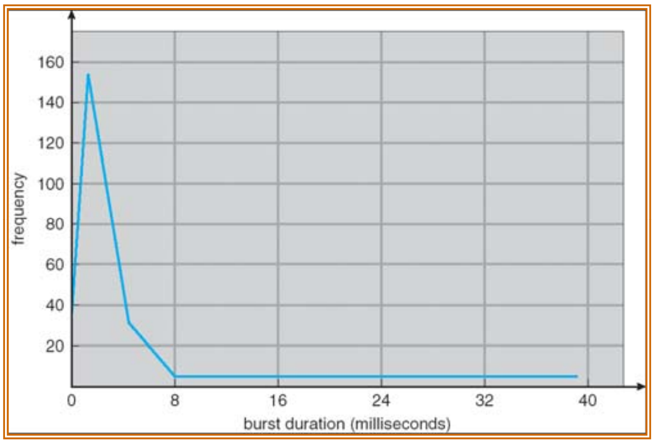
Department of Electrical Engineering, Feng-Chia University

Scheduling Assumptions

- CPU scheduling big area of research in early 70's
- Many implicit assumptions for CPU scheduling:
  - One program per user
  - One thread per program
  - Programs are independent
- Clearly, these are unrealistic but they simplify the problem so it can be solved
  - For instance: is "fair" about fairness among users or programs?
    - If I run one compilation job and you run five, you get five times as much CPU on many operating systems
- The high-level goal: Dole out CPU time to optimize some desired parameters of system

CPU Bursts

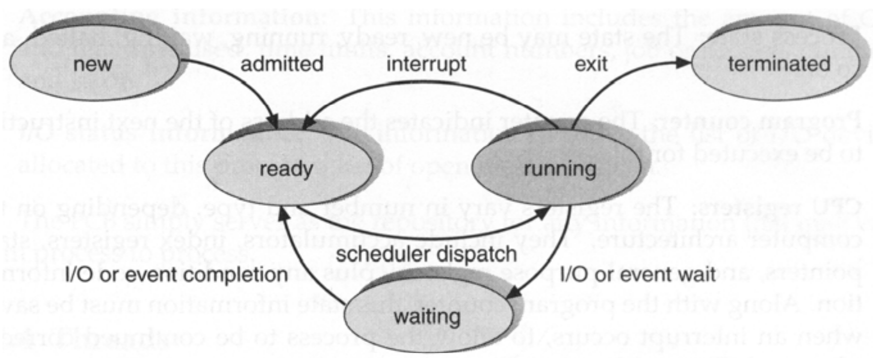




- Execution model: programs alternate between bursts of CPU and I/O
  - Program typically uses the CPU for some period of time, then does I/O, then uses CPU again
  - Each scheduling decision is about which job to give to the CPU for use by its next CPU burst

Department of Electrical Engineering, Feng-Chia University

CPU Scheduler



- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- 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
- Scheduling under 1 and 4 is ***nonpreemptive***
- All other scheduling is ***preemptive***

Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Other Issues

Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process
- 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
- Maximize throughput
  - Throughput related to response time, but not identical:
    - Minimizing response time will lead to more context switching than if you only maximized throughput
  - Two parts to maximizing throughput
    - Minimize overhead (for example, context-switching)
    - Efficient use of resources (CPU, disk, memory, etc)
- Minimize turnaround time
- Minimize waiting time
- Minimize response time

Outline

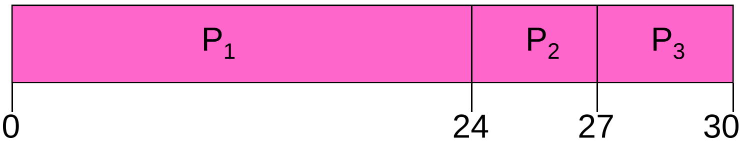
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Other Issues

First-Come, First-Served (FCFS) Scheduling

- **First-Come, First-Served (FCFS)**
  - Also "First In, First Out" (FIFO) or "Run until done"
- Example:

Process	Burst Time
P_1	24
P_2	3
P_3	3

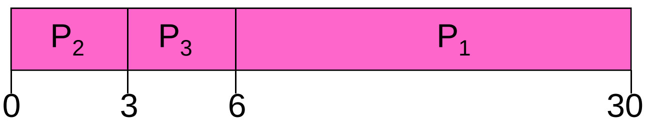
  - Suppose processes arrive in the order: P_1 , P_2 , P_3
    The Gantt Chart for the schedule is:



  - Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
  - Average waiting time: (0 + 24 + 27)/3 = 17
  - Average turnaround time: (24 + 27 + 30)/3 = 27
- *Convoy effect:* short process behind long process

FCFS Scheduling (Cont.)

- Example continued:
  - Suppose that processes arrive in order: P_2 , P_3 , P_1
    Now, the Gantt chart for the schedule is:



  - Waiting time for P_1 = 6; P_2 = 0; P_3 = 3
  - Average waiting time: (6 + 0 + 3)/3 = 3
  - Average turnaround time: (3 + 6 + 30)/3 = 13
- In second case:
  - average waiting time is much better (before it was 17)
  - Average turnaround time is better (before it was 27)
- FIFO Pros and Cons:
  - Simple (+)
  - Short jobs get stuck behind long ones (-)

Round Robin (RR)

- FCFS Scheme: Potentially bad for short jobs!
  - Depends on submit order
  - If you are first in line at supermarket with milk, you don't care who is behind you, on the other hand…
- Round Robin Scheme
  - Each process gets a small unit of CPU time (*time* ***quantum***), usually **10-100** milliseconds
  - After quantum expires, the process is preempted and added to the end of the ready queue.
  - n processes in ready queue and time quantum is $q \Rightarrow$
    - Each process gets $1/n$ of the CPU time
    - In chunks of at most q time units
    - No process waits more than $(n-1)q$ time units

- Performance
  - q large $\Rightarrow$ FCFS
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high (all overhead)

Example of RR with Time Quantum = 20

- Example:

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

  - The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 28 48 68 88 108 112 125 145 153

  - Waiting time for
    P_1=(68-20)+(112-88)=72
    P_2=(20-0)=20
    P_3=(28-0)+(88-48)+(125-108)=85
    P_4=(48-0)+(108-68)=88
  - Average waiting time = (72+20+85+88)/4=$66\frac{1}{4}$
  - Average turnaround time = (125+28+153+112)/4 = $104\frac{1}{2}$
- Thus, Round-Robin Pros and Cons:
  - Better for short jobs, Fair (+)
  - Context-switching time adds up for long jobs (-)

Comparisons between FCFS and Round Robin

- Assuming zero-cost context-switching time, is RR always better than FCFS?
- Simple example: 10 jobs, each take 100s of CPU time
  RR scheduler quantum of 1s
  All jobs start at the same time
- Completion Times:

Job #	FCFS	RR
1	100	991
2	200	992
…	…	…
9	900	999
10	1000	1000

  - Both RR and FCFS finish at the same time
  - Average completion time is much worse under RR!
    - Bad when all jobs same length
- Also: Cache state must be shared between all jobs with RR but can be devoted to each job with FIFO
  - Total time for RR longer even for zero-cost switch!

Department of Electrical Engineering, Feng-Chia University

What if we Knew the Future?

- Shortest Job First (SJF):
  - Run whatever job has the least amount of computation to do
  - Sometimes called "Shortest Time to Completion First" (STCF)
- Shortest Remaining Time First (SRTF):
  - Preemptive version of SJF: if job arrives and has a shorter time to completion than the remaining time on the current job, immediately preempt CPU
  - Sometimes called "Shortest Remaining Time to Completion First" (SRTCF)
- These can be applied either to a whole program or the current CPU burst of each program
  - Idea is to get short jobs out of the system
  - Big effect on short jobs, only small effect on long ones
  - Result is better average response time

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

- Average waiting time = (0 + 6 + 3 + 7) / 4 = 4
- Average turnaround time = (7+10+4+11) / 4 = 8

Example of Preemptive SJF

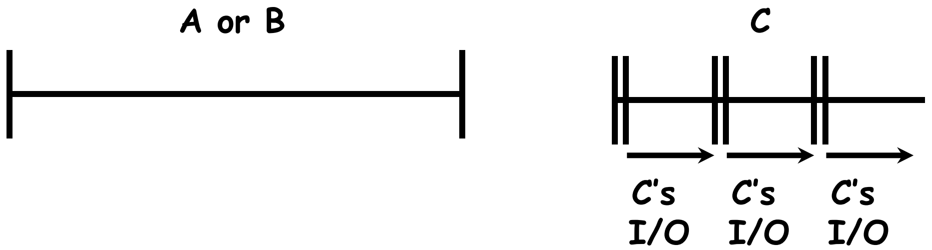
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)

- Average waiting time = (9 + 1 + 0 +2) / 4 = 3
- Average turnaround time = (16+5+1+6) / 4 = 7

Department of Electrical Engineering, Feng-Chia University

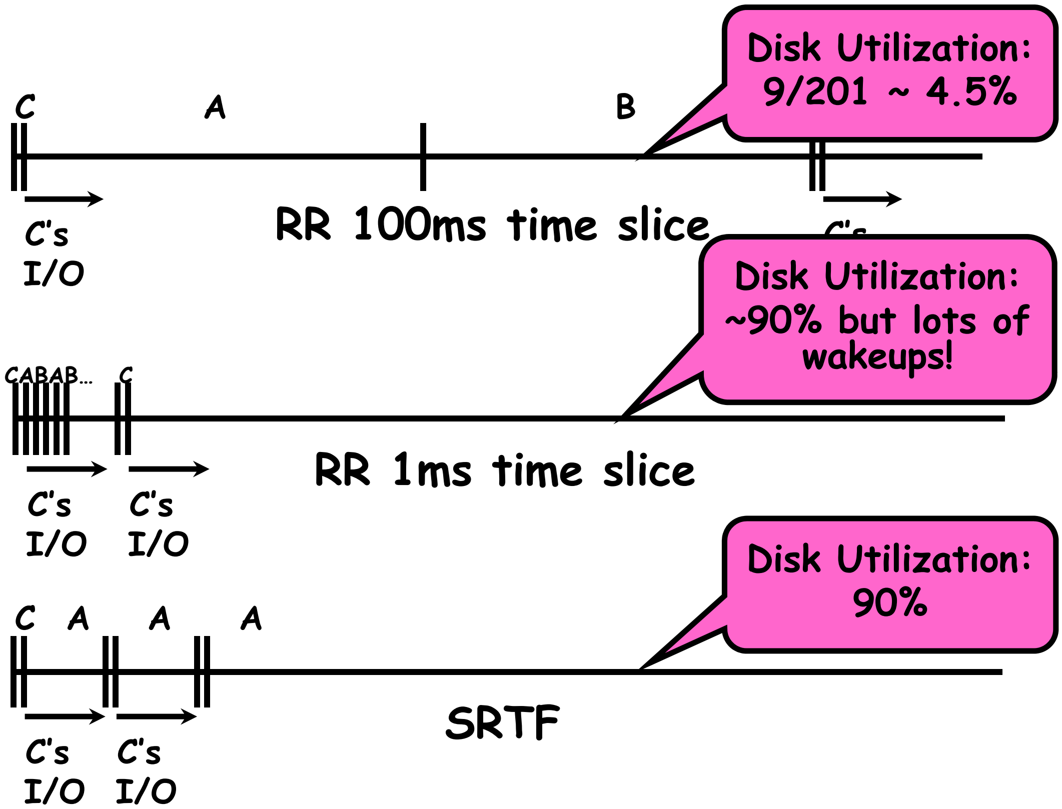
Example to illustrate benefits of SRTF



- Three jobs:
  - A,B: both CPU bound, run for week
    C: I/O bound, loop 1ms CPU, 9ms disk I/O
  - If only one at a time, C uses 90% of the disk, A or B could use 100% of the CPU
- With FIFO:
  - Once A or B get in, keep CPU for two weeks
- What about RR or SRTF?
  - Easier to see with a timeline

Department of Electrical Engineering, Feng-Chia University

SRTF Example continued:

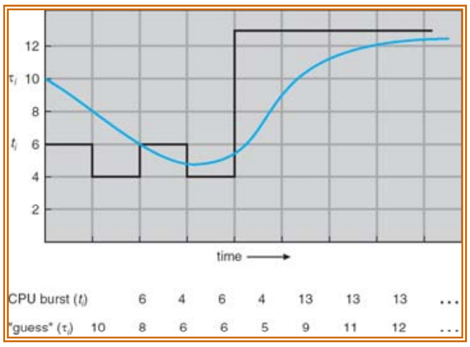


Determining Length of Next CPU Burst

- 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. $\alpha, 0 \le \alpha \le 1$
4. Define :

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



Department of Electrical Engineering, Feng-Chia University

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 the actual last CPU burst counts
- If we expand the formula, we get:

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

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

Department of Electrical Engineering, Feng-Chia University

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
  - nonpreemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem ≡ Starvation – low priority processes may never execute
- Solution ≡ Aging – as time progresses increase the priority of the process

Department of Electrical Engineering, Feng-Chia University

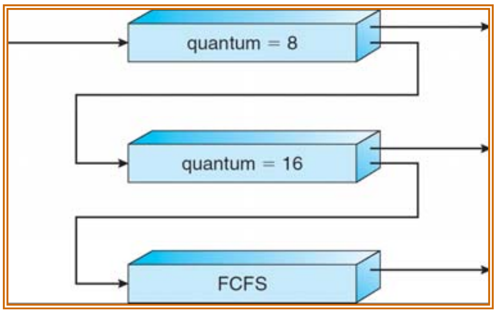
Multilevel Queue

- 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.
  - 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 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
- 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.

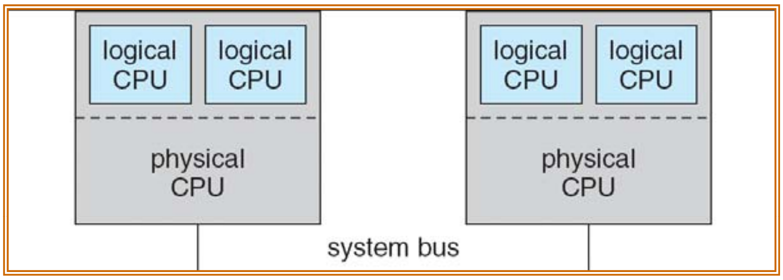
Summary (Scheduling)

- Scheduling:
  - Selecting a waiting process from the ready queue and allocating the CPU to it
- FCFS Scheduling:
  - Run threads to completion in order of submission
  - Pros: Simple
  - Cons: Short jobs get stuck behind long ones
- Round-Robin Scheduling:
  - Give each thread a small amount of CPU time when it executes; cycle between all ready threads
  - Pros: Better for short jobs
  - Cons: Poor when jobs are same length
- Shortest Job First (SJF)/Shortest Remaining Time First (SRTF):
  - Run whatever job has the least amount of computation to do/least remaining amount of computation to do
  - Pros: Optimal (average response time)
  - Cons: Hard to predict future, Unfair

Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- <u>Other Issues</u>

Multiple-Processor Scheduling



- CPU scheduling more complex when multiple CPUs are available
- Homogeneous *processors* within a multiprocessor
- Load sharing
- Asymmetric multiprocessing – only one processor accesses the system data structures, alleviating the need for data sharing

Real-Time Scheduling

- Hard real-time systems – required to complete a critical task within a guaranteed amount of time
  - Resource reservation
- Soft real-time computing – requires that critical processes receive priority over less fortunate ones

Exercises : CPU Scheduling

1. Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Processes	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

The processes are assumed to have arrived in the order P_1, P_2, P_3, P_4, P_5, all at time 0.

(a) Draw four **Gantt charts** illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum=1) scheduling.

(b) What is the **turnaround time** of each process for each of the scheduling algorithms in part(a)?

(c) What is the **waiting time** of each process for each of the scheduling algorithm in part(a)?

(d) Which of the schedules in part(a) results in the minimal average waiting time (over all processes)?

2. There are four processes that arrived at a computer at different time. The arrival time, burst time, and the priority of each process is as the following table: (the time unit is millisecond and a lower priority number means higher priority)

Process	Arrival time	Burst time	Priority
A	2	6	4
B	0	9	3
C	3	10	2
D	5	5	1

(a) Draw five **Gantt charts** illustrating the execution of these processes using FCFS, SJF, SRTF, nonpreemptive priority, and preemptive priority scheduling.

(b) What is the **turnaround time** of each process for each of the scheduling algorithms in part(a)?

(c) What is the **waiting time** of each process for each of the scheduling algorithm in part(a)?

(d) Which of the schedules in part(a) results in the minimal average waiting time (over all processes)?