CPU scheduling

Tran, Van Hoai

Faculty of Computer Science & Engineering
HCMC University of Technology

E-mail: hoai@hcmut.edu.vn
(partly based on slides of Le Thanh Van)

Outline

Outline

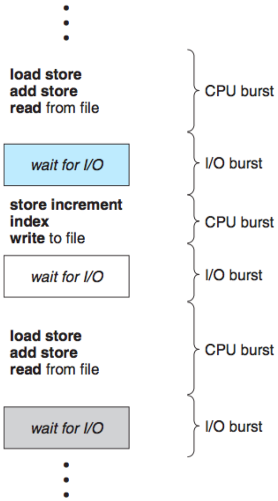
1. Basic concepts
2. Scheduling algorithms
   - Scheduling criteria
   - Scheduling algorithms
3. Multiple-processor scheduling
4. Real-time scheduling
5. Algorithm evaluation

Why do we need CPU scheduling ? (1)

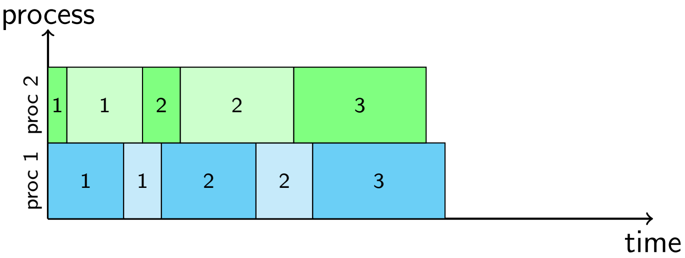
- Maximum CPU "utilization" obtained with multiprogramming

Why do we need CPU scheduling ? (1)

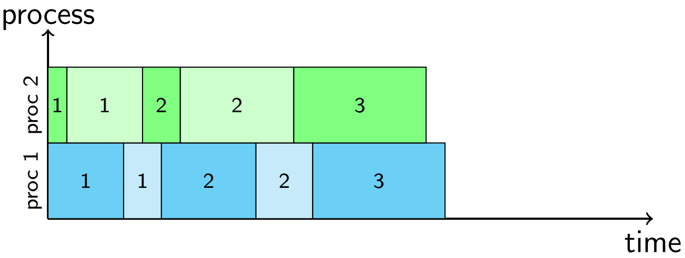
- Maximum CPU "utilization" obtained with multiprogramming
- Process execution consists of a cycle of CPU execution (CPU bound) and I/O wait (I/O bound)
  - CPU burst
  - I/O burst



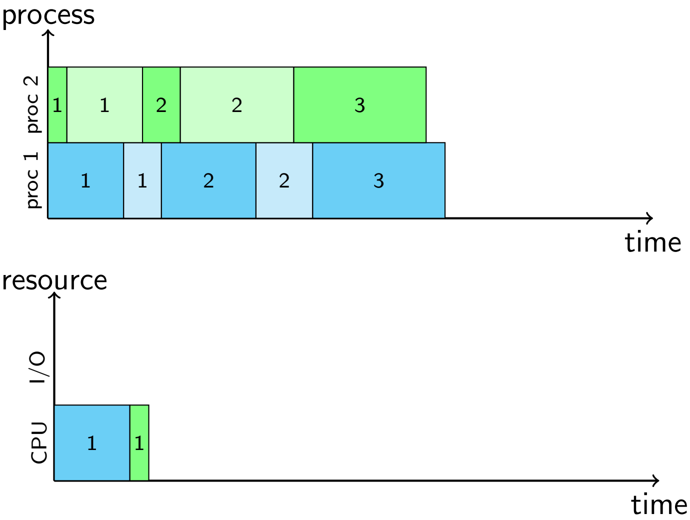
Why do we need CPU scheduling ? (2)



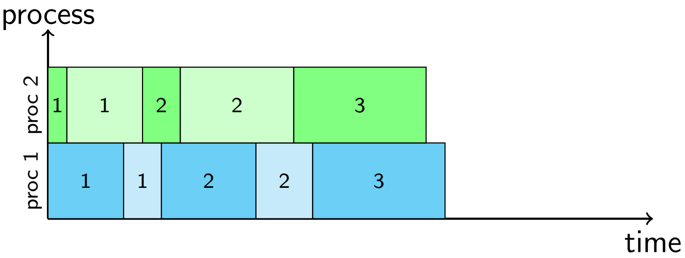
Why do we need CPU scheduling ? (2)



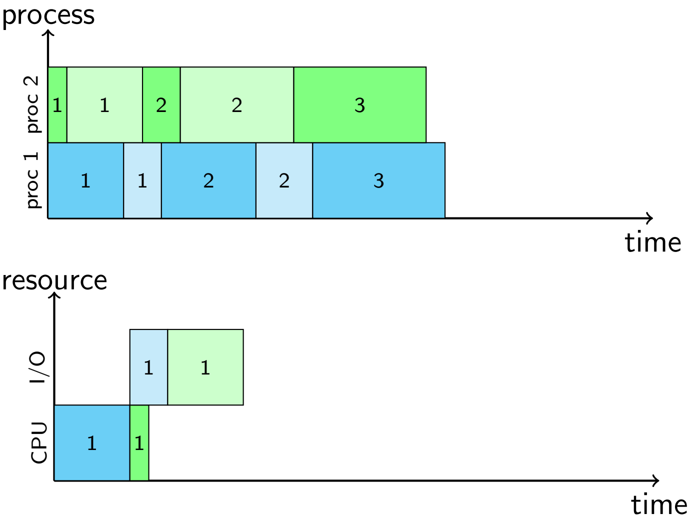
Why do we need CPU scheduling ? (2)



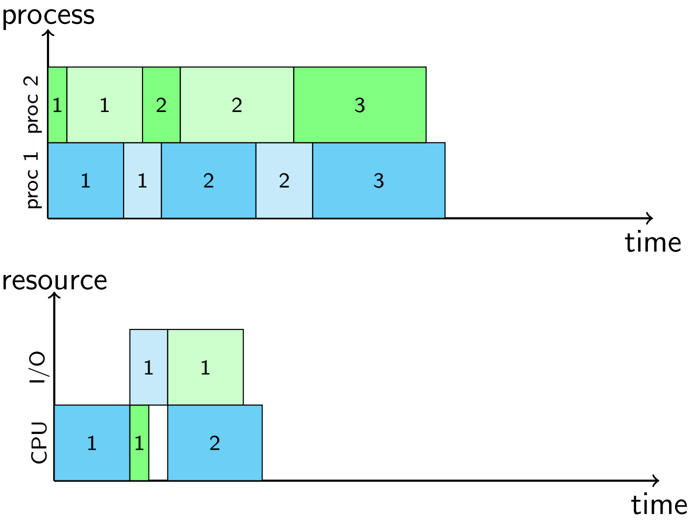
Why do we need CPU scheduling ? (2)



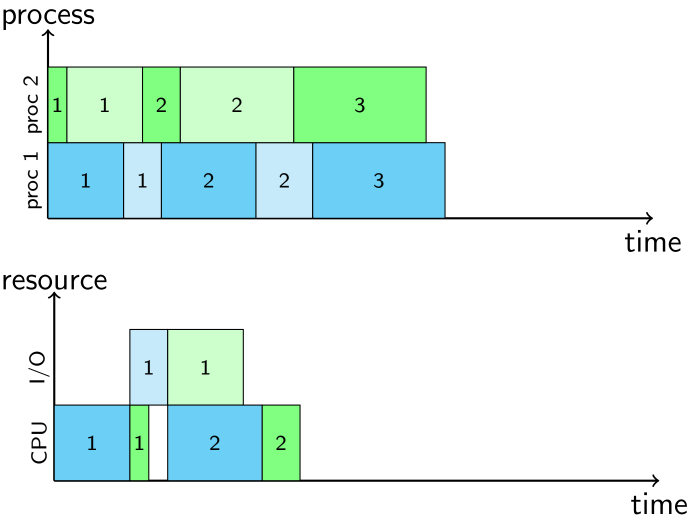
Why do we need CPU scheduling ? (2)



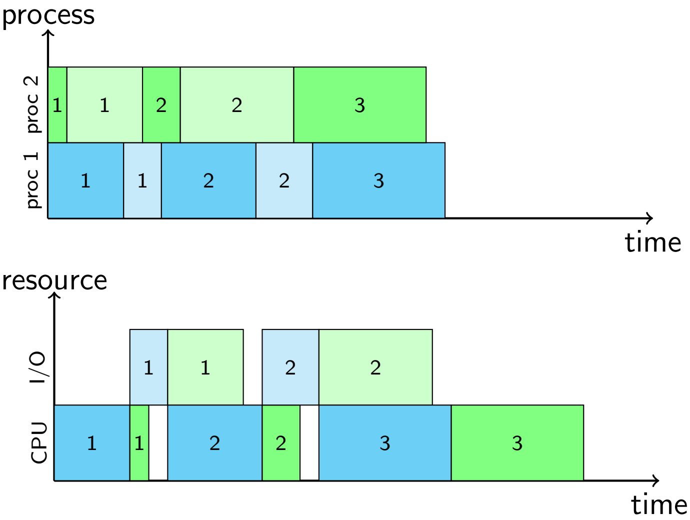
Why do we need CPU scheduling ? (2)



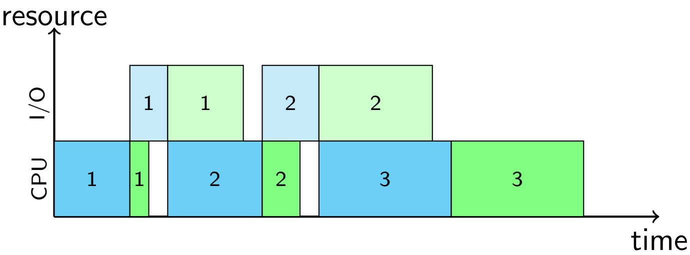
Why do we need CPU scheduling ? (2)

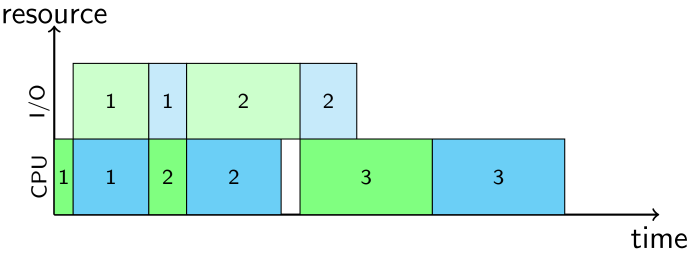


Why do we need CPU scheduling ? (2)

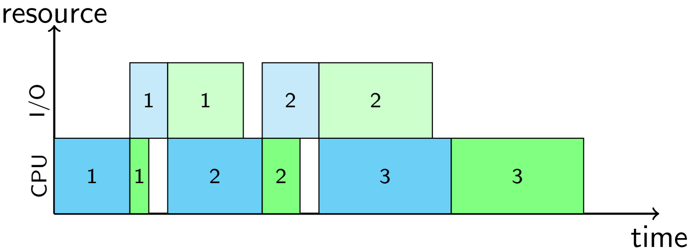


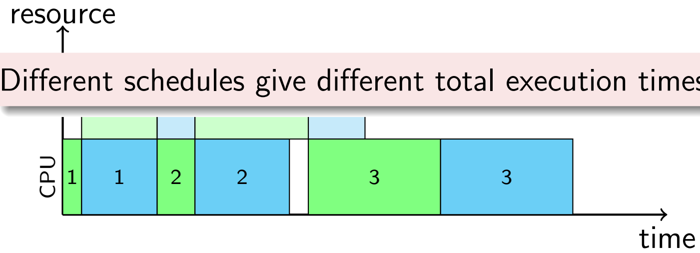
Why do we need CPU scheduling ? (3)





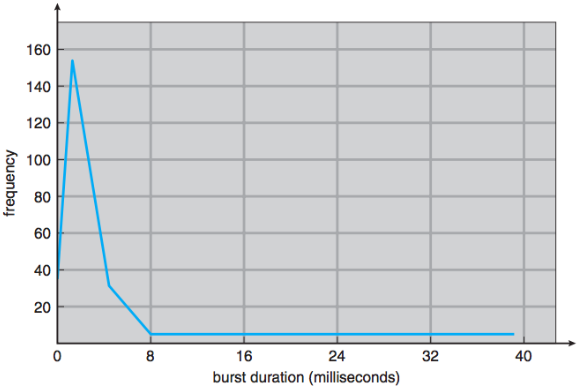
Why do we need CPU scheduling ? (3)



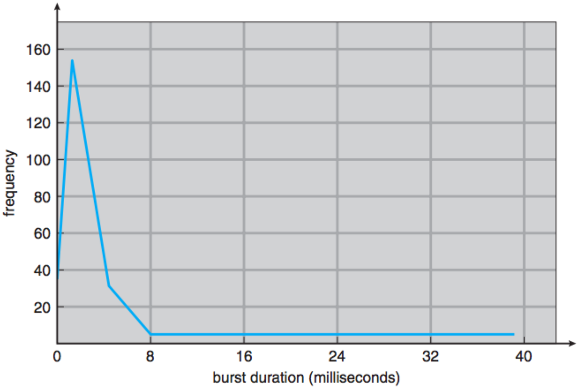


Different schedules give different total execution times

CPU burst distribution



CPU burst distribution



Distribution can be important in the selection of an appropriate CPU-scheduling algorithm

CPU scheduler

Short-term scheduler (CPU scheduler)

When CPU is idle, the scheduler selects a process in the ready queue to be executed next

- Ready queue is not necessarily first-in, first-out (FIFO)
- PCBs are used to store processes in queues

Preemptive vs. nonpreemptive scheduling

CPU scheduling decisions take place in one of 4 following cases

1. a process switches from running state to waiting state
2. a process switches from running state to ready state
3. a process switches from waiting state to ready state
4. a process terminates

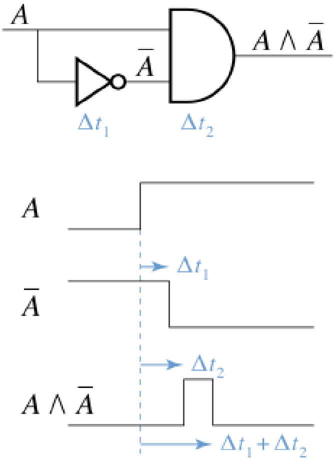
Preemptive vs. nonpreemptive scheduling

CPU scheduling decisions take place in one of 4 following cases

1. a process switches from running state to waiting state
2. a process switches from running state to ready state
3. a process switches from waiting state to ready state
4. a process terminates

- **Cases 1 & 4**: scheduler has to (no choice) choose another ready process for execution. → nonpreemptive or cooperative
- **Cases 2 & 3**: preemptive scheduling. Better, but can lead to race conditions

What is race condition ?



Dispatcher

Dispatcher

A module that gives control of CPU to the process selected by CPU scheduler

Functions of dispatcher

- Switching context
- Switching to user mode
- Jumping to proper location in the user program to restart the program

Dispatch latency should be kept small

Outline

Scheduling criteria

Different criteria suggested for comparing different CPU scheduling algorithms

- CPU utilization (max): CPU is kept as busy as possible
- Throughput (max): number of processes completed per time unit

Scheduling criteria

Different criteria suggested for comparing different CPU scheduling algorithms

- CPU utilization (max): CPU is kept as busy as possible
- Throughput (max): number of processes completed per time unit
- Turnaround time (min): interval from submission to completion of a process
- Waiting time (min): sum of periods spent waiting in the ready queue of a process
- Response time (min): time from the submission of a request until the first response is produced, not including output time

Scheduling criteria

Different criteria suggested for comparing different CPU scheduling algorithms

- CPU utilization (max): CPU is kept as busy as possible
- Throughput (max): number of processes completed per time unit
- Turnaround time (min): interval from submission to completion of a process
- Waiting time (min): sum of periods spent waiting in the ready queue of a process
- Response time (min): time from the submission of a request until the first response is produced, not including output time

- Optimize the average measure
- Optimize the minimum or maximum values

  Example: minimize the maximum response time

First come, first served (FCFS) scheduling (1)

Process	Burst time	Arrival time
P_1	24	0
P_2	3	0
P_3	3	0

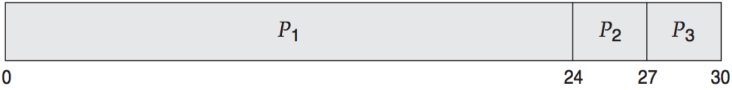
Suppose the processes arrive in the order: P_1, P_2, P_3.

First come, first served (FCFS) scheduling (1)

Process	Burst time	Arrival time
P_1	24	0
P_2	3	0
P_3	3	0

Suppose the processes arrive in the order: P_1, P_2, P_3.

Then for FCFS scheduling, Gantt chart is as follows



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

First come, first served (FCFS) scheduling (2)

Suppose the processes arrive in the order: P_2, P_3, P_1.
Then, Gantt chart is as follows

- Waiting time: $P_1 = 6, P_2 = 0, P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$ (much better)
- convoy effect: all other processes wait for one big process gets off the CPU

Shortest-Job-First (SJF) scheduling

- Each job associated with a time length of its next CPU burst
- Idea: Firstly choosing the job which has shortest time length
- SJF is optimal - giving a minimum average waiting time for a given set of jobs

Shortest-Job-First (SJF) scheduling

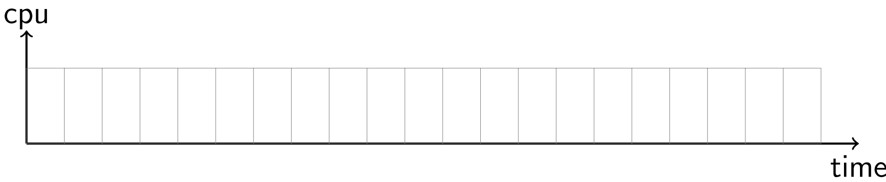
- Each job associated with a time length of its next CPU burst
- Idea: Firstly choosing the job which has shortest time length
- SJF is optimal - giving a minimum average waiting time for a given set of jobs
- 2 schemes:
  - Nonpreemptive: once given to the CPU, the process cannot be preempted until completing its CPU burst
  - Preemptive: if a new process arrives with smaller CPU burst length than remaining time of current executing process, preempt.
    $\Rightarrow$ Its new name is Shortest-Remaining-Time-First (SRTF).

Nonpreemptive Shortest-Job-First (SJF) scheduling

Example

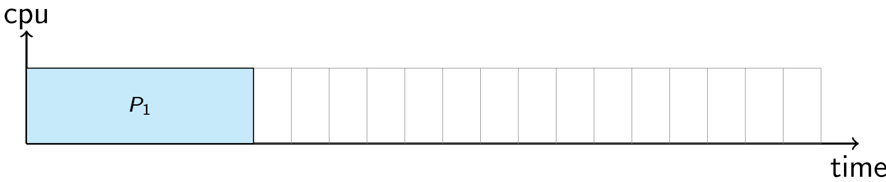
Process	Arrival time	Burst time
P_1	0	6
P_2	1	3
P_3	2	7
P_4	3	4

Then for nonpreemptive SJF scheduling, Gantt chart is as follows



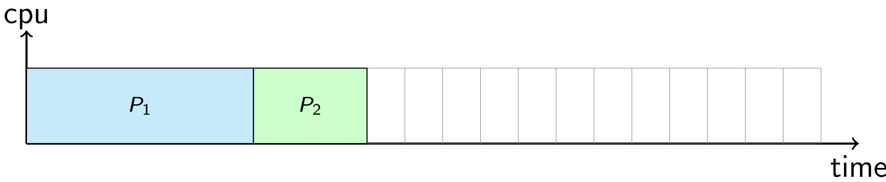
Process	Arrival time	Burst time
P_1	0	6
P_2	1	3
P_3	2	7
P_4	3	4

Then for nonpreemptive SJF scheduling, Gantt chart is as follows



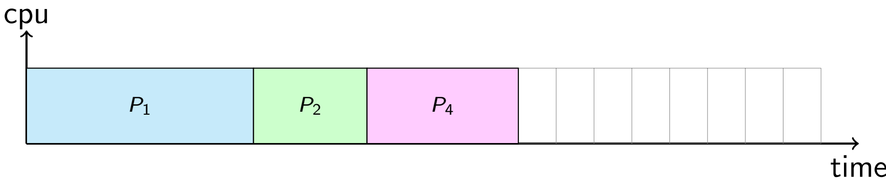
Process	Arrival time	Burst time
P_1	0	6
P_2	1	3
P_3	2	7
P_4	3	4

Then for nonpreemptive SJF scheduling, Gantt chart is as follows



Process	Arrival time	Burst time
P_1	0	6
P_2	1	3
P_3	2	7
P_4	3	4

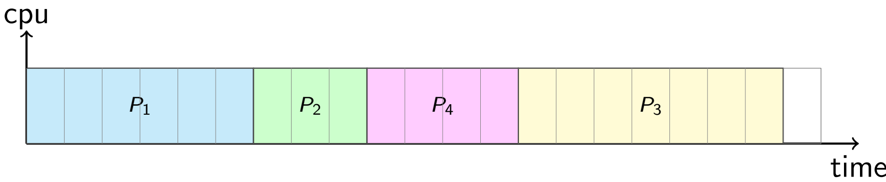
Then for nonpreemptive SJF scheduling, Gantt chart is as follows



Nonpreemptive Shortest-Job-First (SJF) scheduling

Process	Arrival time	Burst time
P_1	0	6
P_2	1	3
P_3	2	7
P_4	3	4

Then for nonpreemptive SJF scheduling, Gantt chart is as follows



- Average waiting time: $((0-0)+(6-1)+(13-2)+(9-3))/4 = 5.5$

Preemptive Shortest-Job-First scheduling (SRTF)

Example

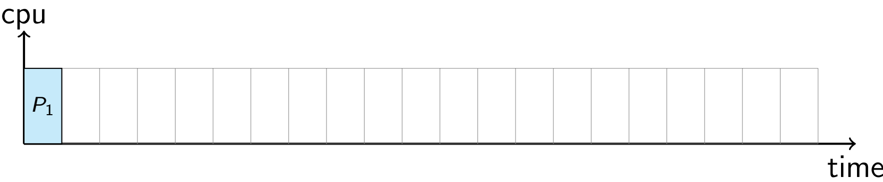
Process	Arrival time	Burst time
P_1	0	6
P_2	1	3
P_3	2	7
P_4	3	4

Then for nonpreemptive SJF scheduling, Gantt chart is as follows

- Average waiting time: $((8-1)+(1-1)+(13-2)+(4-3))/4 = 4.75$

Process	Arrival time	Burst time
P_1	0	6
P_2	1	3
P_3	2	7
P_4	3	4

Then for nonpreemptive SJF scheduling, Gantt chart is as follows



- Average waiting time: $((8-1)+(1-1)+(13-2)+(4-3))/4 = 4.75$

Process	Arrival time	Burst time
P_1	0	6
P_2	1	3
P_3	2	7
P_4	3	4

Then for nonpreemptive SJF scheduling, Gantt chart is as follows

- Average waiting time: $((8-1)+(1-1)+(13-2)+(4-3))/4 = 4.75$

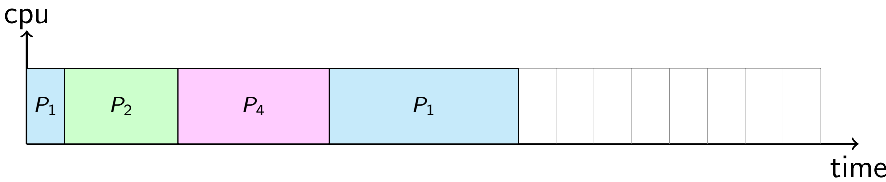
Process	Arrival time	Burst time
P_1	0	6
P_2	1	3
P_3	2	7
P_4	3	4

Then for nonpreemptive SJF scheduling, Gantt chart is as follows

- Average waiting time:
  $((8-1)+(1-1)+(13-2)+(4-3))/4 = 4.75$

Process	Arrival time	Burst time
P_1	0	6
P_2	1	3
P_3	2	7
P_4	3	4

Then for nonpreemptive SJF scheduling, Gantt chart is as follows



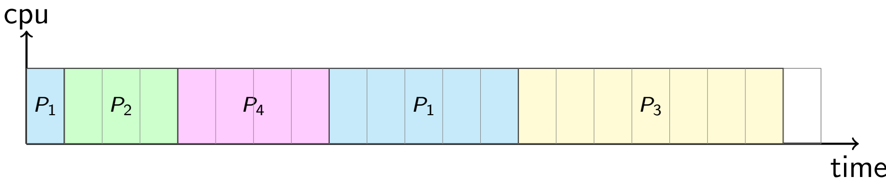
- Average waiting time: $((8-1)+(1-1)+(13-2)+(4-3))/4 = 4.75$

Preemptive Shortest-Job-First scheduling (SRTF)

Example

Process	Arrival time	Burst time
P_1	0	6
P_2	1	3
P_3	2	7
P_4	3	4

Then for nonpreemptive SJF scheduling, Gantt chart is as follows



- Average waiting time: $((8-1)+(1-1)+(13-2)+(4-3))/4 = 4.75$

Length of burst time

SJF is only optimal if burst time (or remaining time) of all jobs must be known in advance for scheduling.

Length of burst time

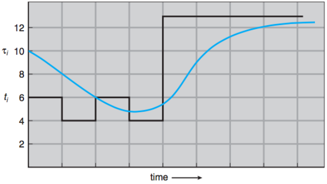
SJF is only optimal if burst time (or remaining time) of all jobs must be known in advance for scheduling.

- Length of next CPU burst can be predicted (approximate SJF scheduling)

Length of burst time

SJF is only optimal if burst time (or remaining time) of all jobs must be known in advance for scheduling.

- Length of next CPU burst can be predicted (approximate SJF scheduling)
- Next CPU burst = exponential average of measured lengths of previous CPU bursts



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

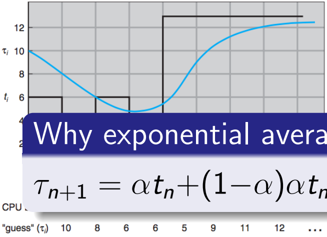
$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$

in which

- $0 \leq \alpha \leq 1$
- t_n: length of nth CPU burst (recent history)
- τ_n: predicted length of nth CPU burst (past history)

SJF is only optimal if burst time (or remaining time) of all jobs must be known in advance for scheduling.

- Length of next CPU burst can be predicted (approximate SJF scheduling)
- Next CPU burst = exponential average of measured lengths of previous CPU bursts



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

in which

- $0 \le \alpha \le 1$

Why exponential average ?

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

history)

Priority scheduling (1)

- Can we generalize SJF by using a number instead of CPU burst time ?

Priority scheduling (1)

- Can we generalize SJF by using a number instead of CPU burst time ?

Priority scheduling

- A priority (a number) is associated with a process
- CPU is allocated to the process with the highest priority
- Assumption: lowest number = highest priority

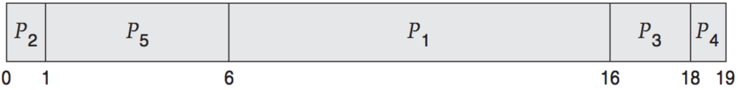
Priority scheduling (1)

- Can we generalize SJF by using a number instead of CPU burst time ?

Priority scheduling

- A priority (a number) is associated with a process
- CPU is allocated to the process with the highest priority
- Assumption: lowest number = highest priority

Process	Burst time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2



Priority scheduling (1)

- Can we generalize SJF by using a number instead of CPU burst time ?

Priority scheduling

- A priority (a number) is associated with a process
- CPU is allocated to the process with the highest priority
- Assumption: lowest number

Process	Burst time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

SJF as a special case of priority scheduling

Priority $=$ CPU burst time

Priority scheduling (2)

- Priority can be preemptive or nonpreemptive

Priority scheduling (2)

- Priority can be preemptive or nonpreemptive
- Priorities can be defined internally or externally
  - *Internally defined*: using some measurable quantities
    Example: time limit, memory requirement, ratio of I/O burst to CPU burst
  - *Externally defined*: set by criteria outside OS
    Example: importance of process, type/amount of funds for computer usage

Priority scheduling (2)

- Priority can be preemptive or nonpreemptive
- Priorities can be defined internally or externally
  - *Internally defined*: using some measurable quantities
    Example: time limit, memory requirement, ratio of I/O burst to CPU burst
  - *Externally defined*: set by criteria outside OS
    Example: importance of process, type/amount of funds for computer usage

Indefinite blocking (starvation)

- Low-priority processes can wait indefinitely
- Solution to starvation is aging which involves gradually increasing the priority of processes that waiting too long.

Round-robin (RR) scheduling

Round-robin scheduling

- Each process gets a small unit of CPU time (time quantum or time slice)
- After quantum has elapsed, the process is preempted and added to the end of the ready queue

Round-robin (RR) scheduling

Round-robin scheduling

- Each process gets a small unit of CPU time (time quantum or time slice)
- After quantum has elapsed, the process is preempted and added to the end of the ready queue

- Time quantum is usually 10-100 milliseconds
- n processes in ready queue,
  time quantum $= q$ (time units)
  $\rightarrow$ no process waits more than $(n-1)q$ time units

Round-robin (RR) scheduling

Round-robin scheduling

- Each process gets a small unit of CPU time (time quantum or time slice)
- After quantum has elapsed, the process is preempted and added to the end of the ready queue

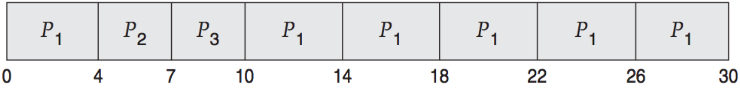
- Time quantum is usually 10-100 milliseconds
- n processes in ready queue,
  time quantum $= q$ (time units)
  $\rightarrow$ no process waits more than $(n-1)q$ time units
- Performance
  - q large $\rightarrow$ FCFS
  - q small $\rightarrow$ overhead is too high due to context switching

Round-robin (RR) scheduling

Example

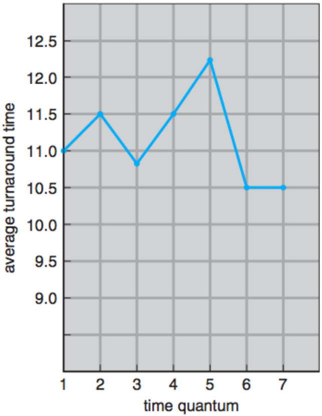
Process	Burst time
P_1	24
P_2	3
P_3	3

Time quantum $= 4$



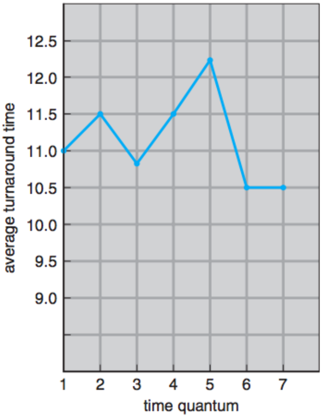
- Average waiting time:
  $((10 - 4) + (4 - 0) + (7 - 0))/3 = 5.66$

On turnaround time



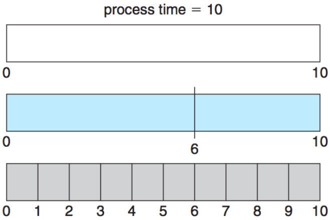
process	time
P_1	6
P_2	3
P_3	1
P_4	7

On turnaround time



process	time
P_1	6
P_2	3
P_3	1
P_4	7

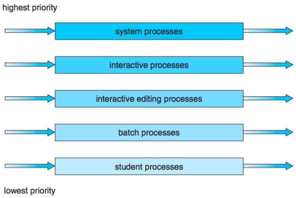
On context switches



quantum	context switches
12	0
6	1
1	9

Multilevel queue scheduling

- Ready queue is partitioned separate queues according to different response-time requirement: foreground (interactive) processes and background (batch) processes.



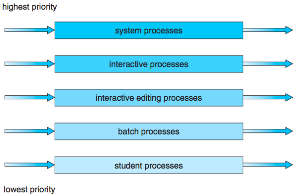
Multilevel queue scheduling

Multilevel queue scheduling

- Ready queue is partitioned separate queues according to different response-time requirement: foreground (interactive) processes and background (batch) processes.

- Each queue has its own scheduling algorithm. For example
  - RR for foreground queue
  - FCFS for background queue
- Scheduling is needed between the queues
  - Fixed preemptive scheduling $\rightarrow$ starvation
  - Time slice: each queue is assigned an amount of time

    For example: 80% for foreground queue; 20% for background queue

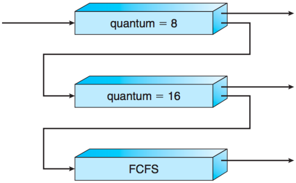


Multilevel feedback queue scheduling

Multilevel feedback

A multilevel queue scheduling but allowing a process to move between queues.

- Separating processes according to their CPU burst times
  - Process with much CPU time moved to low-priority queue
  - Process waiting too long in low-priority queue moved to higher priority queue



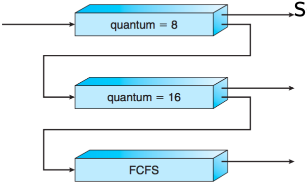
Multilevel feedback queue scheduling

Multilevel feedback

A multilevel queue scheduling but allowing a process to move between queues.

- Separating processes according to their CPU burst times
  - Process with much CPU time moved to low-priority queue
  - Process waiting too long in low-priority queue moved to higher priority queue

Parameters of multilevel feedback queue scheduling:



- number of queues
- scheduling algorithm for each queue
- method to upgrade a process to higher queue; method to demote (giáng cấp) a process to lower queue
- method to choose which queue to put a process for service

- Three queues
  - Q_0 - quantum $= 8$
  - Q_1 - quantum $= 16$
  - 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

Outline

Multiple-processor scheduling

Challenges

Scheduling on multiple-processor systems are more complex

- Homogeneous processors within a multiprocessor
- Load sharing among multiple CPUs
- Asymmetric multiprocessing: only one processor accesses the system data structures, alleviating the need for data sharing
  Most OSs are using symmetric multiprocessing
- Processor affinity (thân thuộc): a process is not migrated to another one than currently running
  Migration could lead to cache invalidation or repopulation.

Load balancing

Load balancing

Keeping the workload evenly distributed across all processors

- Only necessary on systems where each processor has its own private ready queue
- Load balancing vs. processor affinity
- Load balancing can be performed in 2 forms
  - push migration
  - pull migration

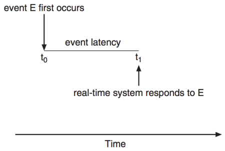
Outline

Real-time scheduling

Real-time scheduling

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

- Event latency kept small in order to increase responsiveness to events (minimizing latency)



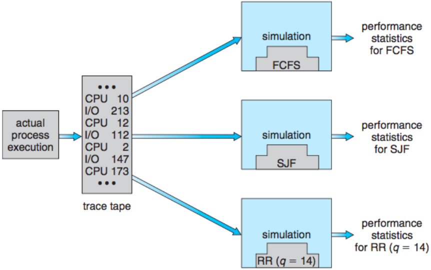
- Real-time system must have priority-based scheduling with preemption

Outline

1. Basic concepts
2. Scheduling algorithms
   - Scheduling criteria
   - Scheduling algorithms
3. Multiple-processor scheduling
4. Real-time scheduling
5. Algorithm evaluation

Algorithm evaluation

- Deterministic modeling: takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Queueing models
- Computer simulation (e.g., by discrete-event simulation)



Homeworks

1. Read materials on Multiple processor scheduling & Realtime-scheduling: textbook, slides (Nguyen Thanh Son)