操作系统原理

第六章：CPU调度

洪明坚

重庆大学软件学院

February 19, 2016

目录

Outline

Basic concepts

Basic concepts

- The multiprogramming is used by the operating system to overlap the operations of CPU and I/O devices in order to maximize CPU utilization.

Basic concepts

- The multiprogramming is used by the operating system to overlap the operations of CPU and I/O devices in order to maximize CPU utilization.
  - When current running process can not proceed, the operating system must schedule one of the ready processes to run.

Basic concepts

- The multiprogramming is used by the operating system to overlap the operations of CPU and I/O devices in order to maximize CPU utilization.
  - When current running process can not proceed, the operating system must schedule one of the ready processes to run.
- Scheduling is a fundamental operating system function.

Basic concepts

- The multiprogramming is used by the operating system to overlap the operations of CPU and I/O devices in order to maximize CPU utilization.
    - When current running process can not proceed, the operating system must schedule one of the ready processes to run.
- Scheduling is a fundamental operating system function.
    - Almost all computer resources must be scheduled before use.

Basic concepts

- The multiprogramming is used by the operating system to overlap the operations of CPU and I/O devices in order to maximize CPU utilization.
  - When current running process can not proceed, the operating system must schedule one of the ready processes to run.
- Scheduling is a fundamental operating system function.
  - Almost all computer resources must be scheduled before use.
  - The CPU scheduling is central to operating system design.

Basic concepts

- The multiprogramming is used by the operating system to overlap the operations of CPU and I/O devices in order to maximize CPU utilization.
    - When current running process can not proceed, the operating system must schedule one of the ready processes to run.
- Scheduling is a fundamental operating system function.
    - Almost all computer resources must be scheduled before use.
    - The CPU scheduling is central to operating system design.
- Operating systems supporting threads at the kernel level must schedule threads (NOT processes) for execution.

CPU-I/O burst cycle (1/2)

CPU-I/O burst cycle (1/2)

- The success of CPU scheduling depends on the following observed property of processes:

CPU-I/O burst cycle (1/2)

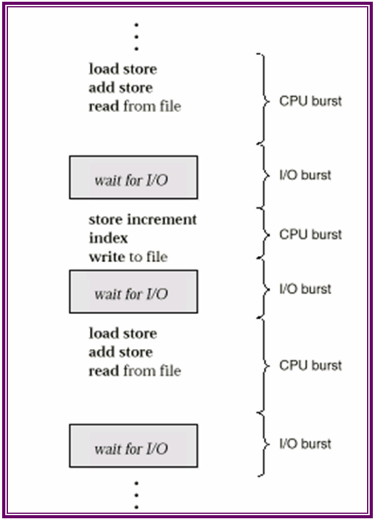
- The success of CPU scheduling depends on the following observed property of processes:
    - Process execution consists of a cycle of **CPU execution** and **I/O wait**.

CPU-I/O burst cycle (1/2)

- The success of CPU scheduling depends on the following observed property of processes:
  - Process execution consists of a cycle of **CPU execution** and **I/O wait**.
  - Processes alternate between these two states.

CPU-I/O burst cycle (1/2)

- The success of CPU scheduling depends on the following observed property of processes:
  - Process execution consists of a cycle of **CPU execution** and **I/O wait**.
  - Processes alternate between these two states.



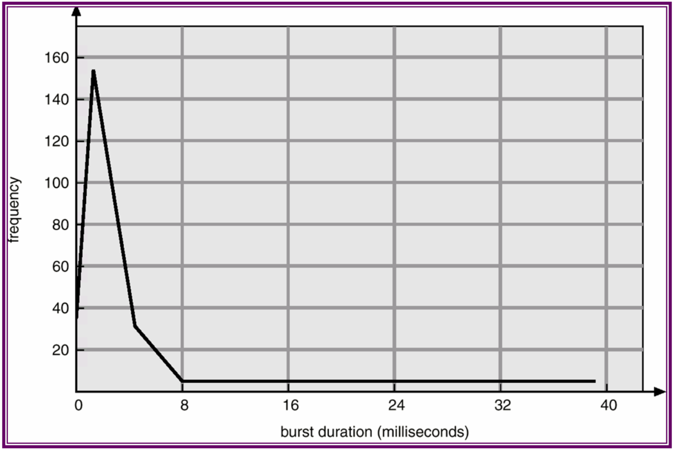
CPU-I/O burst cycle (2/2)

CPU-I/O burst cycle (2/2)

- CPU burst distribution

CPU-I/O burst cycle (2/2)

- CPU burst distribution



CPU scheduler

CPU scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

CPU scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
    - Also known as **short-term scheduler**.

CPU scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
  - Also known as **short-term scheduler**.
- CPU scheduling decisions may take place when a process:

CPU scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
  - Also known as **short-term scheduler**.
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state.

CPU scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
  - Also known as **short-term scheduler**.
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state.
  2. Switches from running to ready state.

CPU scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
    - Also known as **short-term scheduler**.
- 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.

CPU scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
  - Also known as **short-term scheduler**.
- 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.

CPU scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
  - Also known as **short-term scheduler**.
- 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 **non-preemptive**. All other scheduling is **preemptive**.

Dispatcher

Dispatcher

- *Dispatcher* module gives control of the CPU to the process selected by the short-term scheduler; this involves:

Dispatcher

- *Dispatcher* module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context

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

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

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*

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.

Scheduling criteria

Scheduling criteria

- CPU utilization

Scheduling criteria

- CPU utilization –keep the CPU as busy as possible

Scheduling criteria

- CPU utilization –keep the CPU as busy as possible
- Throughput

Scheduling criteria

- CPU utilization –keep the CPU as busy as possible
- Throughput –# of processes that complete their execution per time unit

Scheduling criteria

- CPU utilization –keep the CPU as busy as possible
- Throughput –# of processes that complete their execution per time unit
- Turnaround time

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

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

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

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

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

Outline

1. CPU scheduling
   - Basic concepts
   - CPU or I/O burst
   - CPU scheduler
   - Dispatcher
   - Scheduling criteria
2. Scheduling algorithms
   - First-come, first-served
   - Shortest Job First
   - Priority scheduling
   - Round-Robin scheduling
   - Multilevel queue
   - Multilevel feedback queue
   - Conclusion

First-come, first-served (FCFS)

First-come, first-served (FCFS)

- Example

First-come, first-served (FCFS)

- Example

Process	Burst time
P_1	24
P_2	3
P_3	3

First-come, first-served (FCFS)

- Example

Process	Burst time
P_1	24
P_2	3
P_3	3

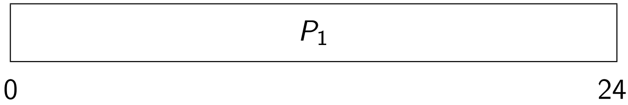
- Suppose that the processes arrive in the order: P_1, P_2, P_3. The *Gantt Chart* for the schedule is:

First-come, first-served (FCFS)

- Example

Process	Burst time
P_1	24
P_2	3
P_3	3

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

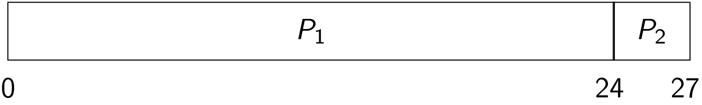


First-come, first-served (FCFS)

- Example

Process	Burst time
P_1	24
P_2	3
P_3	3

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

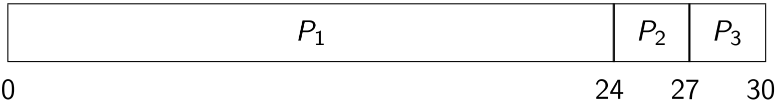


First-come, first-served (FCFS)

- Example

Process	Burst time
P_1	24
P_2	3
P_3	3

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



First-come, first-served (FCFS)

- Example

Process	Burst time
P_1	24
P_2	3
P_3	3

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

P_1	P_2	P_3
0 – 24	24 – 27	27 – 30

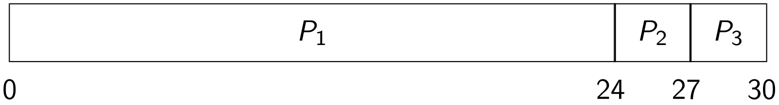
- Waiting time for P1 = 0; P2 = 24; P3 = 27

First-come, first-served (FCFS)

- Example

Process	Burst time
P_1	24
P_2	3
P_3	3

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



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

Shortest-Job-First (SJF)

Shortest-Job-First (SJF)

- Associate with each process the length of its next CPU burst.

Shortest-Job-First (SJF)

- Associate with each process the length of its next CPU burst.
  - Use these lengths to schedule the process with the shortest time.

Shortest-Job-First (SJF)

- Associate with each process the length of its next CPU burst.
  - Use these lengths to schedule the process with the shortest time.
- Two schemes:

Shortest-Job-First (SJF)

- Associate with each process the length of its next CPU burst.
  - Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - Non-preemptive

Shortest-Job-First (SJF)

- Associate with each process the length of its next CPU burst.
    - Use these lengths to schedule the process with the shortest time.
- Two schemes:
    - Non-preemptive - once CPU given to the process it cannot be preempted until completes its CPU burst.

Shortest-Job-First (SJF)

- Associate with each process the length of its next CPU burst.
  - Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - Non-preemptive - once CPU given to the process it cannot be preempted until completes its CPU burst.
  - Preemptive

Shortest-Job-First (SJF)

- Associate with each process the length of its next CPU burst.
  - Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - Non-preemptive - 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.

Shortest-Job-First (SJF)

- Associate with each process the length of its next CPU burst.
  - Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - Non-preemptive - 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 known as the *Shortest-Remaining-Time-First (SRTF)*.

Shortest-Job-First (SJF)

- Associate with each process the length of its next CPU burst.
  - Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - Non-preemptive - 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 known as the *Shortest-Remaining-Time-First (SRTF)*.
- SJF is optimal gives minimum average waiting time for a given set of processes.

Example: Non-preemptive SJF

Example: Non-preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

Example: Non-preemptive SJF

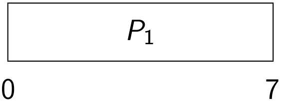
Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- Non-preemptive SJF

Example: Non-preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

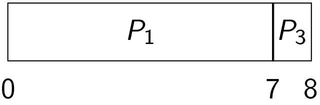
- Non-preemptive SJF



Example: Non-preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

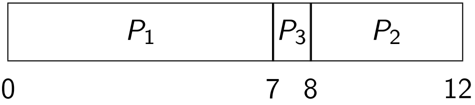
- Non-preemptive SJF



Example: Non-preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

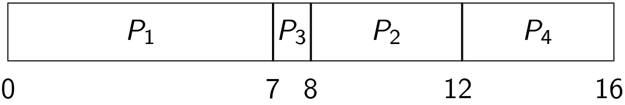
- Non-preemptive SJF



Example: Non-preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

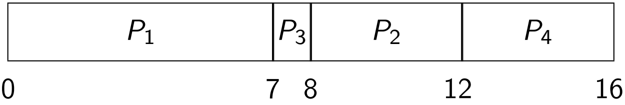
- Non-preemptive SJF



Example: Non-preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- Non-preemptive SJF



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

Example: Preemptive SJF

Example: Preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

Example: Preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- Preemptive SJF

Example: Preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- Preemptive SJF

P_1

0 2

Example: Preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- Preemptive SJF

P_1	P_2

0 2 4

Example: Preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- Preemptive SJF

P_1	P_2	P_3

0 2 4 5

Example: Preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- Preemptive SJF

P_1	P_2	P_3	P_2

0 2 4 5 7

Example: Preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

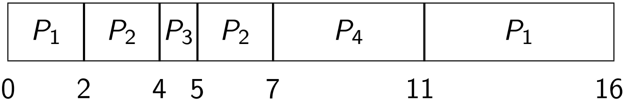
- Preemptive SJF

P_1	P_2	P_3	P_2	P_4

0 2 4 5 7 11

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

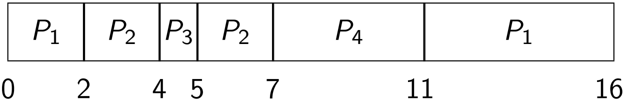
- Preemptive SJF



Example: Preemptive SJF

Process	Arrival time	Burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- Preemptive SJF



- Average waiting time $= (9 + 1 + 0 + 2)/4 = 3$.

Priority scheduling

Priority scheduling

- A priority number (integer) is associated with each process

Priority scheduling

- A priority number (integer) is associated with each process
  - SJF is a priority scheduling where priority is the next CPU burst time.

Priority scheduling

- A priority number (integer) is associated with each process
  - SJF is a priority scheduling where priority is the next CPU burst time.
- The CPU is allocated to the process with the highest priority.

Priority scheduling

- A priority number (integer) is associated with each process
  - SJF is a priority scheduling where priority is the next CPU burst time.
- The CPU is allocated to the process with the highest priority.
  - It can be preemptive or non-preemptive
- Problems

Priority scheduling

- A priority number (integer) is associated with each process
  - SJF is a priority scheduling where priority is the next CPU burst time.
- The CPU is allocated to the process with the highest priority.
  - It can be preemptive or non-preemptive
- Problems
  1. *Starvation*

Priority scheduling

- A priority number (integer) is associated with each process
  - SJF is a priority scheduling where priority is the next CPU burst time.
- The CPU is allocated to the process with the highest priority.
  - It can be preemptive or non-preemptive
- Problems
  1. *Starvation* - low priority processes may never execute.

Priority scheduling

- A priority number (integer) is associated with each process
  - SJF is a priority scheduling where priority is the next CPU burst time.
- The CPU is allocated to the process with the highest priority.
  - It can be preemptive or non-preemptive
- Problems
  1. *Starvation* - low priority processes may never execute.
  2. *Priority inversion*

Priority scheduling

- A priority number (integer) is associated with each process
  - SJF is a priority scheduling where priority is the next CPU burst time.
- The CPU is allocated to the process with the highest priority.
  - It can be preemptive or non-preemptive
- Problems
  1. *Starvation* - low priority processes may never execute.
  2. *Priority inversion* - higher-priority process need to access a resource being held by another lower-priority process.

Priority scheduling

- A priority number (integer) is associated with each process
  - SJF is a priority scheduling where priority is the next CPU burst time.
- The CPU is allocated to the process with the highest priority.
  - It can be preemptive or non-preemptive
- Problems
  1. *Starvation* - low priority processes may never execute.
  2. *Priority inversion* - higher-priority process need to access a resource being held by another lower-priority process.
- Solution

Priority scheduling

- A priority number (integer) is associated with each process
  - SJF is a priority scheduling where priority is the next CPU burst time.
- The CPU is allocated to the process with the highest priority.
  - It can be preemptive or non-preemptive
- Problems
  1. *Starvation* - low priority processes may never execute.
  2. *Priority inversion* - higher-priority process need to access a resource being held by another lower-priority process.
- Solution
  - *Aging*

Priority scheduling

- A priority number (integer) is associated with each process
  - SJF is a priority scheduling where priority is the next CPU burst time.
- The CPU is allocated to the process with the highest priority.
  - It can be preemptive or non-preemptive
- Problems
  1. *Starvation* - low priority processes may never execute.
  2. *Priority inversion* - higher-priority process need to access a resource being held by another lower-priority process.
- Solution
  - *Aging* - increasing the priority of the process with time going on.

Round-Robin scheduling (RR)

Round-Robin scheduling (RR)

- Each process gets a small unit of CPU time (time *quantum* or *slice*), usually 10-100 milliseconds.

Round-Robin scheduling (RR)

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

Round-Robin scheduling (RR)

- Each process gets a small unit of CPU time (time *quantum* or *slice*), usually 10-100 milliseconds.
    - After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q,

Round-Robin scheduling (RR)

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

Round-Robin scheduling (RR)

- Each process gets a small unit of CPU time (time *quantum* or *slice*), usually 10-100 milliseconds.
  - After this time has elapsed, the process is preempted and added to the end of the ready queue.
- 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 $(n\text{-}1)q$ time units.

Round-Robin scheduling (RR)

- Each process gets a small unit of CPU time (time *quantum* or *slice*), usually 10-100 milliseconds.
  - After this time has elapsed, the process is preempted and added to the end of the ready queue.
- 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 $(n\text{-}1)q$ time units.
- Performance

Round-Robin scheduling (RR)

- Each process gets a small unit of CPU time (time *quantum* or *slice*), usually 10-100 milliseconds.
  - After this time has elapsed, the process is preempted and added to the end of the ready queue.
- 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 $(n\text{-}1)q$ time units.
- Performance
  - q large

Round-Robin scheduling (RR)

- Each process gets a small unit of CPU time (time *quantum* or *slice*), usually 10-100 milliseconds.
  - After this time has elapsed, the process is preempted and added to the end of the ready queue.
- 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 $(n\text{-}1)q$ time units.
- Performance
  - q large - FCFS

Round-Robin scheduling (RR)

- Each process gets a small unit of CPU time (time *quantum* or *slice*), usually 10-100 milliseconds.
  - After this time has elapsed, the process is preempted and added to the end of the ready queue.
- 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 $(n\text{-}1)q$ time units.
- Performance
  - q large - FCFS
  - q small

Round-Robin scheduling (RR)

- Each process gets a small unit of CPU time (time *quantum* or *slice*), usually 10-100 milliseconds.
  - After this time has elapsed, the process is preempted and added to the end of the ready queue.
- 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 $(n\text{-}1)q$ time units.
- Performance
  - q large - FCFS
  - q small - q must be large with respect to context switch, otherwise overhead is too high.

Multilevel queue (1/2)

Multilevel queue (1/2)

- Ready queue is further partitioned into separate queues:

Multilevel queue (1/2)

- Ready queue is further partitioned into separate queues:
  - A priority number is associated with each queue.

Multilevel queue (1/2)

- Ready queue is further partitioned into separate queues:
  - A priority number is associated with each queue.
  - Example: foreground (for interactive processes) and background (for batch processes) queues.

Multilevel queue (1/2)

- Ready queue is further partitioned into separate queues:
  - A priority number is associated with each queue.
  - Example: foreground (for interactive processes) and background (for batch processes) queues.
- Each queue has its own scheduling algorithm,

Multilevel queue (1/2)

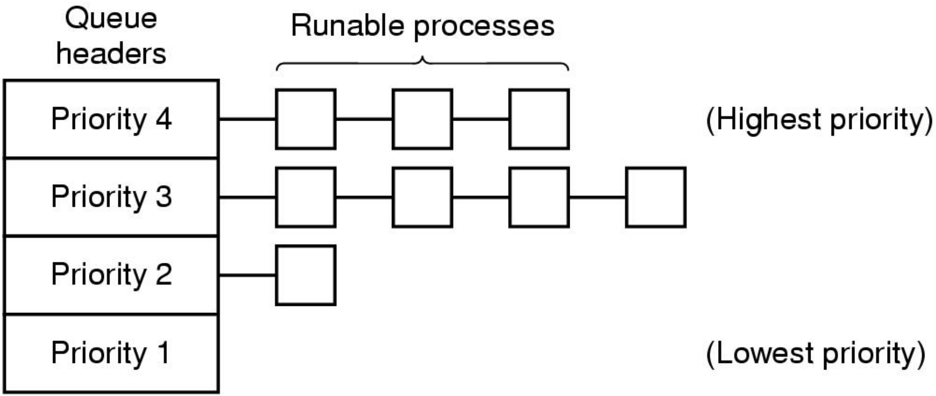
- Ready queue is further partitioned into separate queues:
  - A priority number is associated with each queue.
  - Example: foreground (for interactive processes) and background (for batch processes) queues.
- Each queue has its own scheduling algorithm,
  - foreground uses RR; background uses FCFS, for example.

Multilevel queue (1/2)

- Ready queue is further partitioned into separate queues:
  - A priority number is associated with each queue.
  - Example: foreground (for interactive processes) and background (for batch processes) queues.
- Each queue has its own scheduling algorithm,
  - foreground uses RR; background uses FCFS, for example.

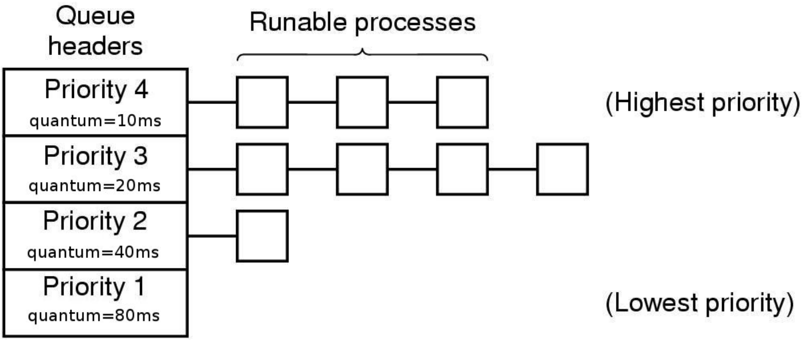
Multilevel queue (2/2)

Multilevel queue (2/2)

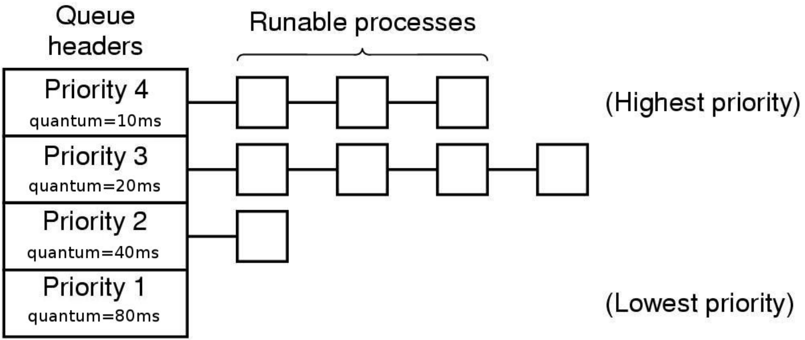


Multilevel feedback queue scheduling

Multilevel feedback queue scheduling

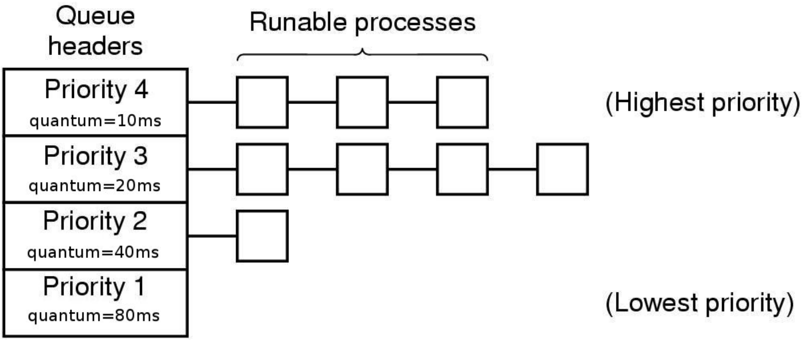


Multilevel feedback queue scheduling



- A process can move between the various queues.

Multilevel feedback queue scheduling



- A process can move between the various queues.
  - For example, whenever a process used up the quantum allocated to it, it sank into a lower priority queue.

Conclusion

Conclusion

- *Getting it right in practice is much harder than getting it right in principle.*

Conclusion

- *Getting it right in practice is much harder than getting it right in principle.*
    - As a result, the scheduler rarely makes the best choice.

Conclusion

- *Getting it right in practice is much harder than getting it right in principle.*
    - As a result, the scheduler rarely makes the best choice.
    - The solution:

Conclusion

- *Getting it right in practice is much harder than getting it right in principle.*
  - As a result, the scheduler rarely makes the best choice.
  - The solution: separation of scheduling policy and scheduling mechanism.

Conclusion

- *Getting it right in practice is much harder than getting it right in principle.*
  - As a result, the scheduler rarely makes the best choice.
  - The solution: separation of scheduling policy and scheduling mechanism.
    - That is, the scheduling algorithm is parameterized in some way, but the parameters can be filled in by the user.

Questions

- Any questions?