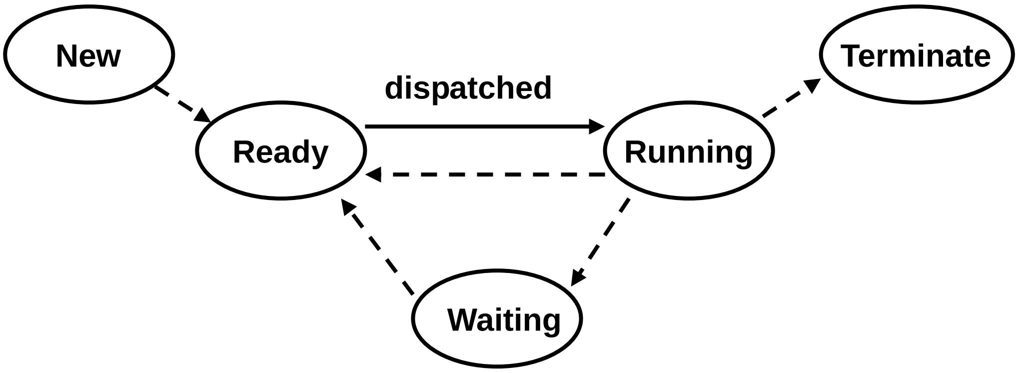
6 CPU Scheduling

The Objective of Multiprogramming

Maximize CPU utilization (based on some criteria)

CPU Scheduling

-- The selection process ～ CPU scheduler, i.e., short-term scheduler



-- Nonpreemptive scheduling
A running process keeps CPU until it volunteers to release CPU
Adv. Easy to implement (at the cost of better resource sharing)
∗ Adopted by Windows 3.1

-- Preemptive scheduling (強者暫停弱者)
CPU scheduling occurs whenever some process become ready or the running process leaves running state！

Issues involved：

-- Synchronization & protection of resources such as I/O queues

＊ 何時需 Scheduling？

1. process need I/O
2. running → ready → timeout (time-sharing) by interrupt
3. waiting state → ready state (I/O 完了) by interrupt
4. process 結束

情況 1, 4 下，scheduling → non-preemptive scheduling；否則為 preemptive

-- Dispatcher (the module that gives control of the CPU to the process selected by the short-term scheduler)

Functionalities：

. Switch context

. Switching to user mode

. Restarting a user program

Dispatch Latency：(def.)

Stop a process ←— **Must be fast** —→ **Star a process**

Process Scheduling

∗ Choose one of processes which are in the ready state to be the running state.

∗ Considerations

(a) fairness.

(b) CPU utilization

(c) Throughput：the number of processes that are completed per time unit.

(d) Turnaround time：the time the batch users must wait for output.

(* for a particular process, we care about how long it takes to execute that process; the interval from the time of submission to the time of completion – including the time waiting in memory, waiting in the ready queue, executing on CPU, doing I/O *)

(e) Waiting time：for each process, the amount of time that a process spends waiting in the ready queue.

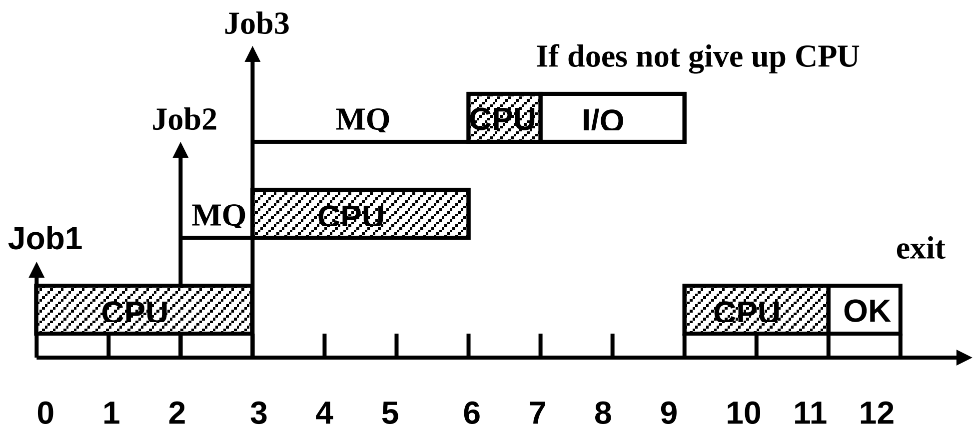
(f) Response time：in an active system, the time from the submission of a request until the first response is produced.

(* not the output *)

That is, the amount of time it takes to start responding, but not the time it takes to output the response.

EX. (round-robin; fixed time slot)

CPU utilization：9/12 = 0.75
Turnaround time：job 1, 12t
Waiting time：job1, 6t
Response time：
Throughput：(up to 12t) 1/12



Scheduling Criteria

Used to compare CPU scheduling algorithms！

∗ A CPU scheduling algorithm only has impacts on the time of a process waiting for dispatching, rather than its execution time on CPU！

1. CPU utilization (↑)
   ～ keep CPU as busy as possible！

2. Throughput (↑)
   ～maximize # of completing process / time unit
   Issues：long transactions vs short transactions

3. Turnaround time (針對某一個) (↓)

   Process submitted $\xleftrightarrow{\quad t \quad}$ **Process completes**

   (含在 disk memory，在 ready queue，等 I/O..)

   ～minimize t

4. Wait time (↓)
   ～minimize the sum of time spent waiting in the ready queue

5. Response time (in an interactive system) (↓)

   Process submitted $\xleftrightarrow{\quad t \quad}$ **Process starts responding**

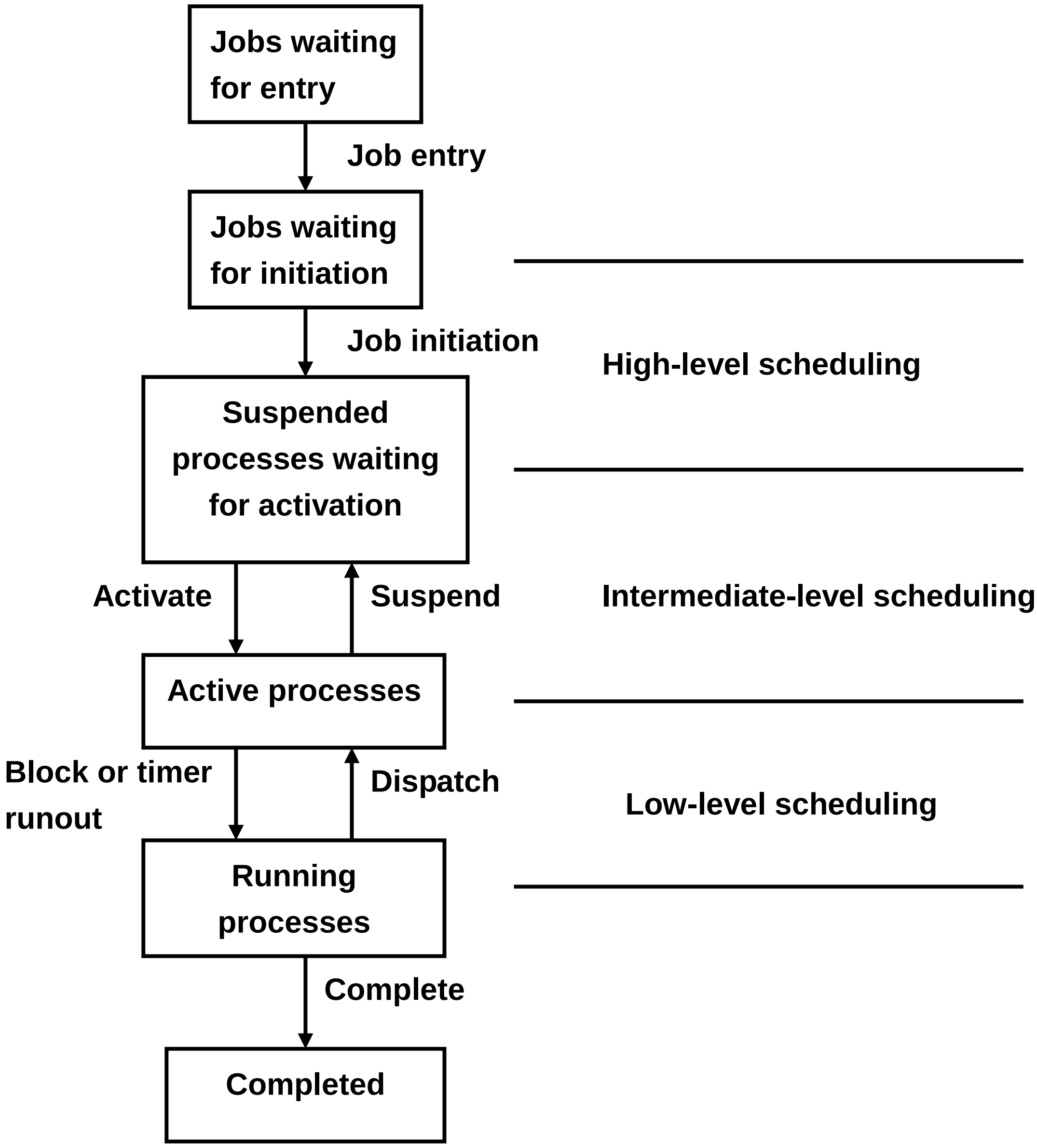
   (從 output devices)

   ～minimize t

排班層次（Scheduling level）

排班層次如下頁圖中所示，共分有：

1. high-level scheduling (或稱 long-term/job scheduling)：此 scheduler 決定哪些 job 允許載入主記憶體內以備執行，其特點有：
   (1) 執行頻率較少。
   (2) 需要花較多的執行時間以決定哪些 job 可進入主記憶體中。
   (3) 決定 degree of multiprogramming。
   (4) 必須精確地排班使得在主記憶體的 processes 性質是 CPU-bound 與 I/O-bound 的比例能平均。
   (5) 此排班程式通常是由 halt-state （即完成一個 process）所驅動執行的。


Jobs waiting for entry
Job entry
Jobs waiting for initiation
Job initiation
High-level scheduling
Suspended processes waiting for activation
Activate
Suspend
Intermediate-level scheduling
Active processes
Block or timer runout
Dispatch
Low-level scheduling
Running processes
Complete
Completed

2. Intermediate-level scheduling (或稱 medium-term scheduling)

為了改進 CPU 與 I/O 間的負載平衡，平均 CPU-bound 與 I/O-bound 程序的比例或由於某些因素致使主記憶體中的 process 被交換出（swap out）至磁碟上，或由磁碟上交換入(swap in）至主記憶體中，因此藉著暫時性的暫停（suspending）或啟動（activating）程序以達上述目的。這些暫停與啟動的排班工作即是由中程排班程式（medium-term scheduler）所負責。

3. Low-level scheduling (或稱 short-term/process scheduling)：

從主記憶體中等待執行的 processes 選擇其中之一以交由 CPU 執行，此排班程式之特點如下：

(1) 執行頻率較多。
(2) 由於每執行一次 CPU 控制權轉換一次，因此造成使用者時間上的浪費，故 process scheduler 決定 CPU 的使用效益。
(3) 為了減少 process scheduler 的執行所花費的 overhead，故須減少 process scheduler 的執行時間。

排班佇列圖

圖 a 為 CPU 排班的佇列圖，其中等待佇列 (ready queue)內存由 job scheduler 所選入的 processes 以等待 CPU 的執行。而 process scheduler 則從等待佇列中選擇其中一個 process 以供 CPU 執行。若執行過程中有 I/O 需求時，則此 process 會進入 I/O 佇列 (I/O queue)以等待 I/O 的執行。圖 b 為圖 a 的簡化圖。若加入中程排班 (即有 swap in 與 swap out 的功能) 時，則可如圖 c 所示。圖 d 圍等待佇列與 I/O 佇列的結構。

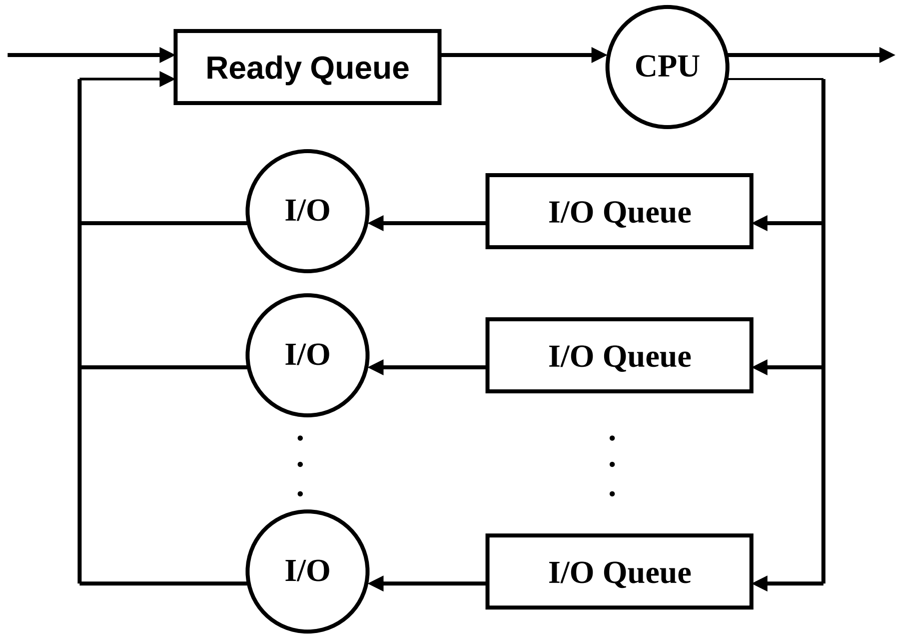


圖 a　CPU 排班的佇列圖

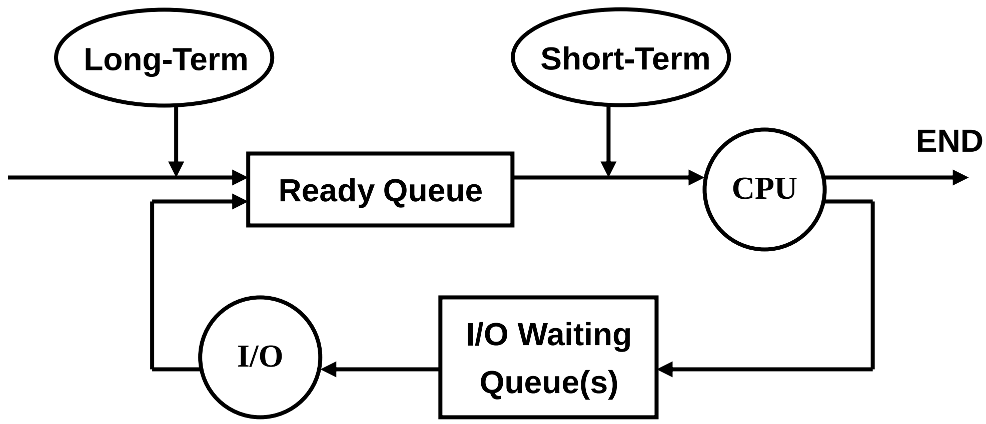


圖 b　簡化佇列圖

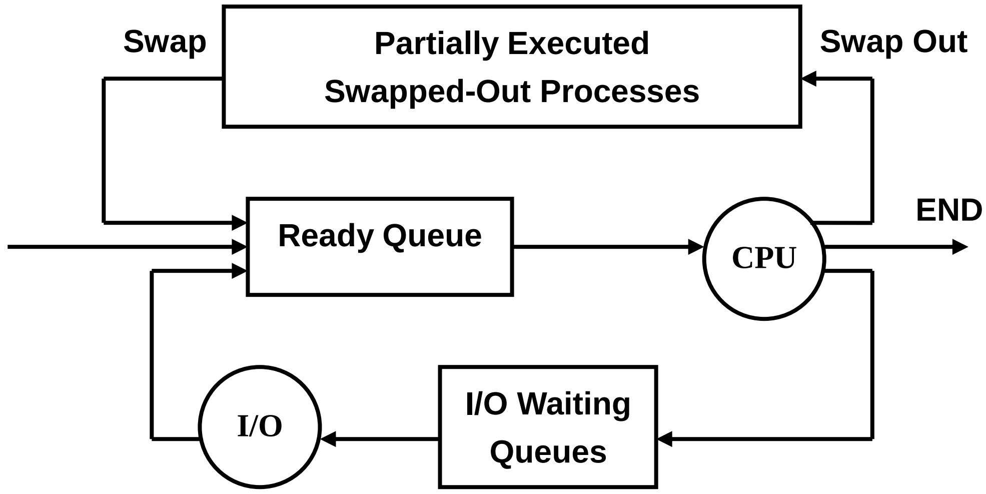


圖 c　備有中程排班的佇列圖

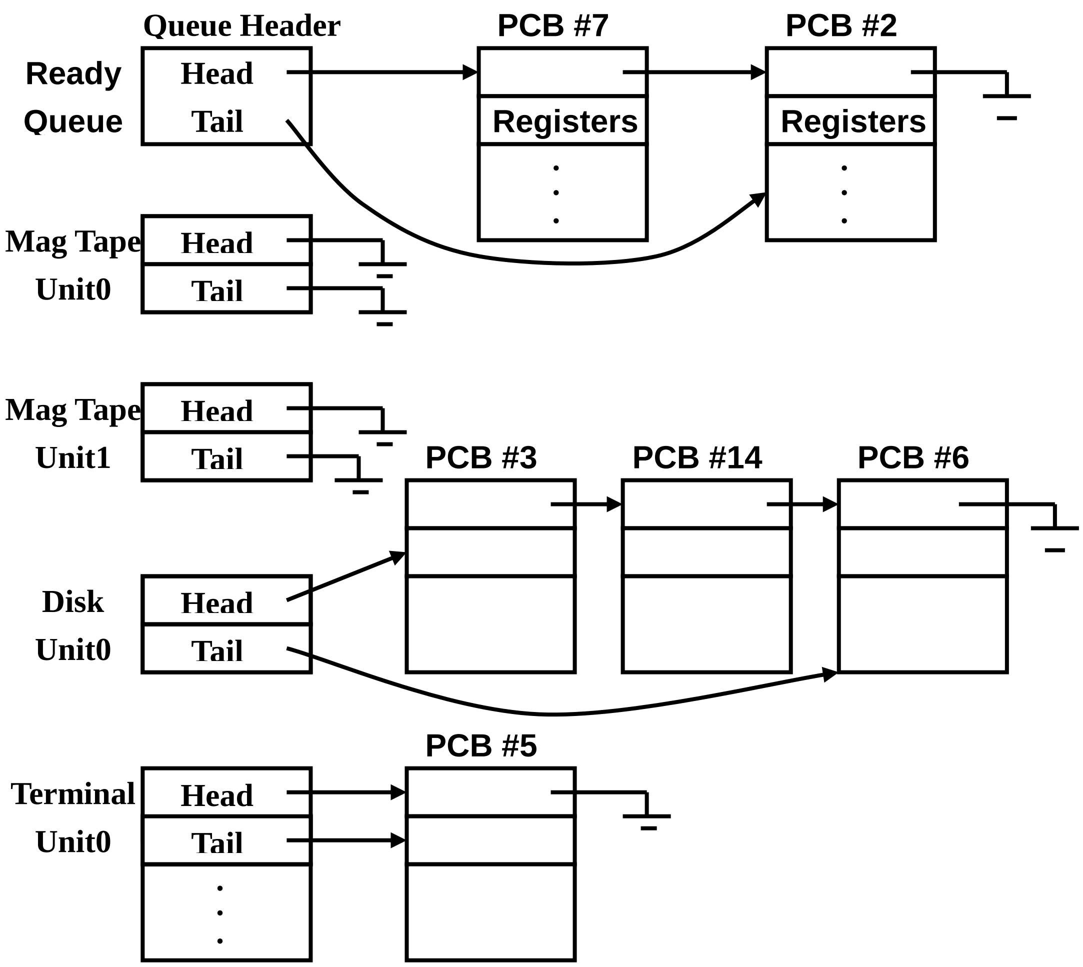
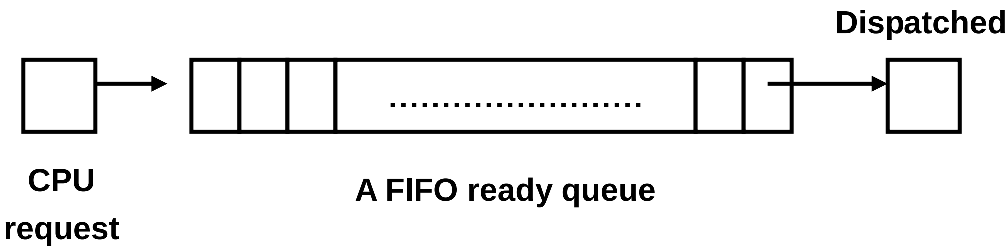


圖 d　等待佇列與 I/O 佇列的結構

Scheduling Algorithms

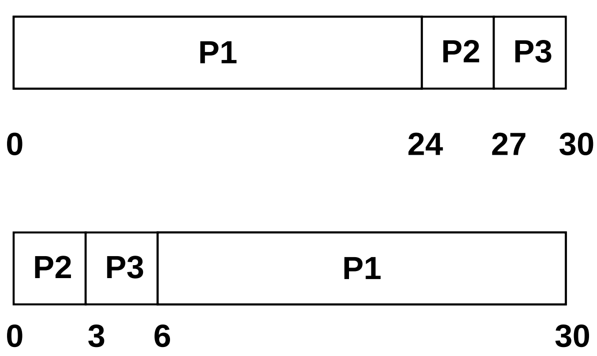
(1) First-Come, First Served Scheduling (FCFS)



-- Non-preemptive
-- CPU might be hold for an extended period

Example1：

Process	CPU Burst time
P1	24
P2	3
P3	3



Average waiting time = (0+24+27)/3 = 17

Average waiting time = (6+0+3)/3 = 3

Gantt Chart (甘蔗圖)

＊Average waiting time is highly affected by process CPU burst times！

Classification

Preemptive vs. nonpreemptive (run-to-complete) scheduling

- the average waiting time under a FCFS policy is generally not minimal.
-
- Not good for time-sharing systems.

Job2	Job3	Job1

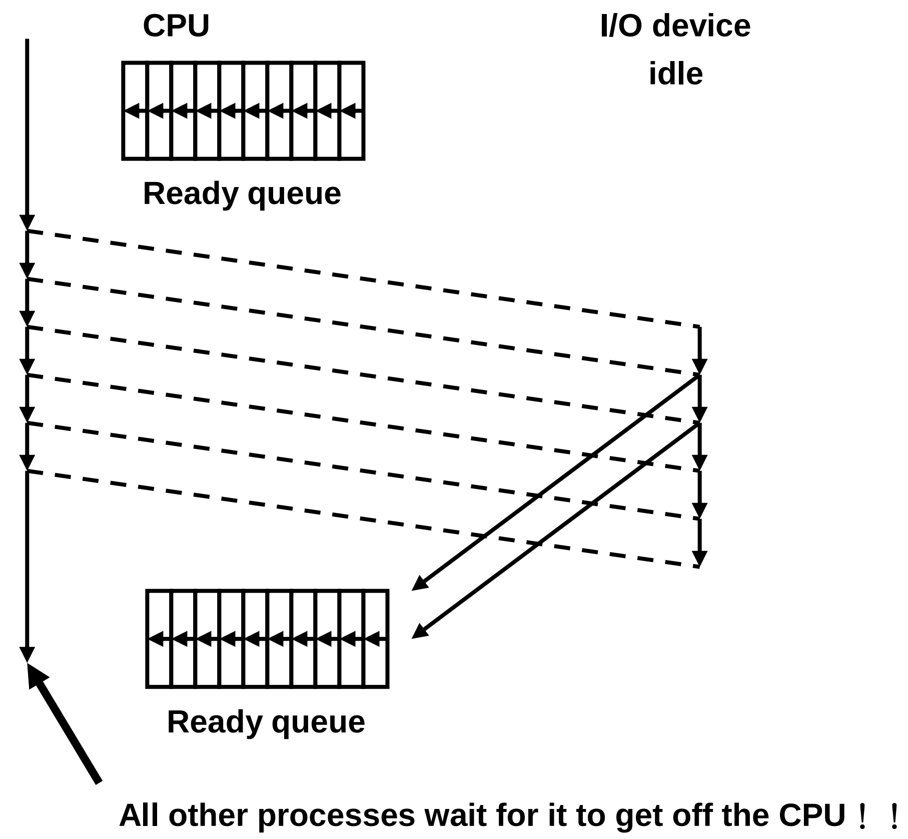
0 3 6 30

因此 average turnaround time = (30+3+6)/3 = 13
故可知 FCFS 排班法的效益和程序進入 ready queue 的順序有關。

Example2：Convoy Effect (島群)

Process set = One CPU-bound process / Many I/O-bound process

A scenario：



＊ **CPU utilization** ↓ **(** 很不好 **)**

(2) Shortest-Job-First Scheduling (SJF)

-- Nonpreemptive SJF
shortest next CPU burst first (而非 its total length)

Process	CPU Burst time
P1	6
P2	8
P3	7
P4	3

P4	P1	P3	P2

0 3 9 16 24

AWT = 1/4 * (3 + 9 + 16) = 28/4 = 7

(∗ compared to FCFS 的 10.25 ∗)

∗ optimal in that it gives the minimum average waiting time (When processes are all ready at time ?!) (Batch System)

∗ Prediction of the next CPU burst time！(難處)
～exponential average

SJF：常用於 long-term Scheduling
不適用於 short-term CPU Scheduling

- Good for batch systems. (run time is known in advance)
- But not good for interactive systems. (how do we know the run time of all the processes in advance？)

-- Preemptive SJF

shortest-remaining-time-first

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

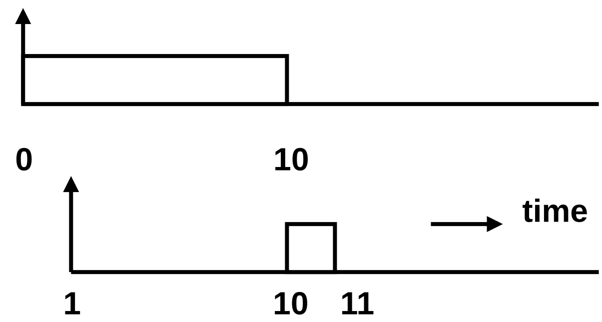
P1	P2	P4	P1	P3

0 1 5 10 17 26

Average Waiting Time = ((10-1) + (1-1) + (17-2) + (5-3))/4 = 26/4 = 6.5

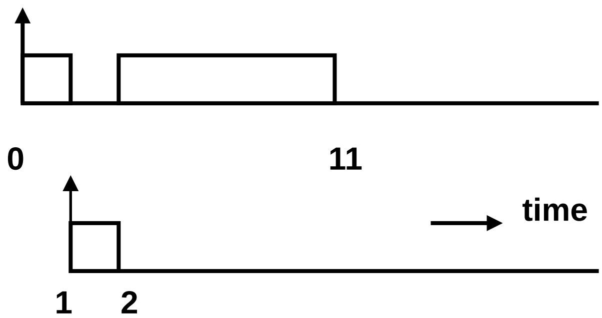
-- Preemptive or Nonpreemptive ?

∗ criteria such as AWT (Average Waiting Time)



Nonpreemptive
AWT = (0 + (10-1))/2
= 9/2
= 4.5

or

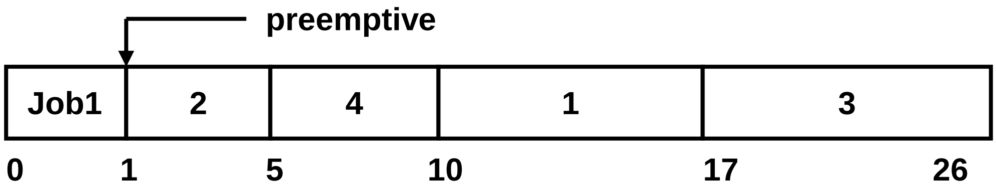


Preemptive AWT
= ((2-1) + 0)
= 0.5

∗ context switching cost

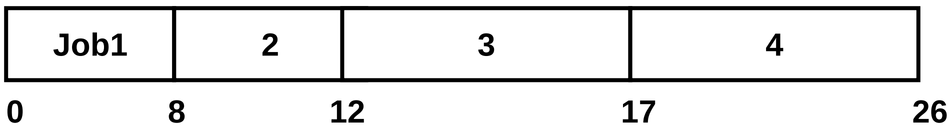
～modeling & analysis

若採用 preemptive SJF 的排班法，則其 Gantt chart 為



其 average turnaround time （完成時間）
= ((17-0) + (5-1) + (26-2) + (10-3))/4 =52/4 = 13

但若採用 non-preemptive SJF 的排班法，則其 Gantt chart 為



其 average turnaround time
= ((8-0) + (12-1) + (17-2) + (26-3))/4
= 57/4
= 14.25

(3) Priority Scheduling

A framework that always schedules the process with the highest priority

Equal-priority, tie-breaking by FCFS → FCFS priority

$$\frac{1}{\text{Next CPU burst length}} \rightarrow \text{SJF}$$

Avg. waiting time is 8.2.

process	CPU Burst time	Priority
P1	10	3
P2	1	1(highest)
P3	2	3
P4	1	4
P5	5	2

Gantt graph

P2	P5	P1	P3	P4

0 1 6 16 18 19

-- Priority Assignment
. internally defined – use some measurable quality such as # as open files, $\frac{\text{Average I/O Burst}}{\text{Average CPU Burst}}$

(time limits, memory requirement, the number of open files, the ratio of average I/O..)

.externally defined – set by criteria external to the O.S., such as the criticality of jobs

-- Preemptive or not？

. Preemptive scheduling –

CPU scheduling is invoked whenever a process arrives at the ready queue, or the running process relinquishes the CPU

.Nonpreemptive scheduling –

CPU scheduling is invoked only when the running process relinquishes the CPU

-- Major problem

Indefinite blocking (starvation)

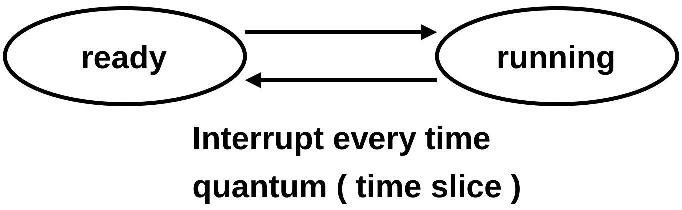
～ low priority process starves to death！

A solution：Aging

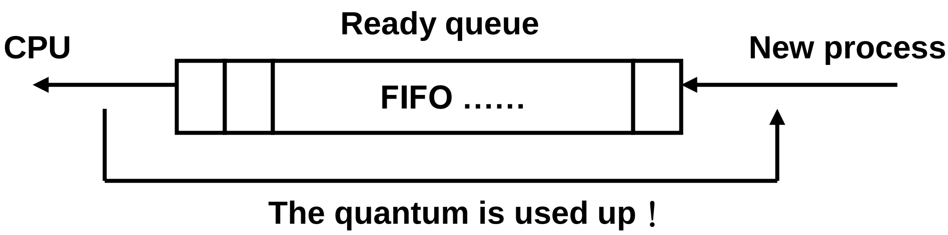
A technique that increases the priority of processes waiting in the system for a long time

(4) Round-Robin Scheduling (RR)

Similar to FCFS except that preemption is added to switch between processes.



Goal：Fairness～for time sharing system



Process	CPU Burst time
P1	24
P2	6
P3	3

Time slice = 4

P1	P2	P3	P1	P2	P1	P1	P1	P1

0 4 8 11 15 17 21 25 29 33

AWT = ((7+2) + (4+7) + 8))/3 = 28/3 = 9.3

Average waiting time ➔ long

-- Service size & interval

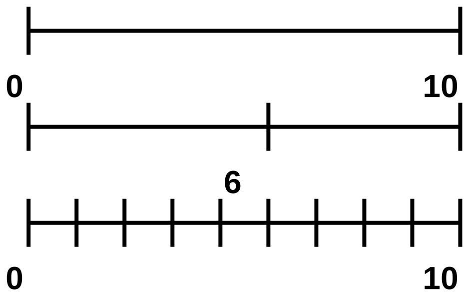
Time quantum：q

Service interval ≦ (N-1) * q if n processes are ready & …

If q = ∞, RR → FCFS

If q = ↓0, RR → processor sharing

# of context switching ↑



process	quantum	Context switch # only approximated
0 – 10	12	0
0 – 6 – 10	6	1
0 – 10	1	9

$$\frac{\text{If context switching cost}}{\text{Time quantum}} = 10\ \%$$

➔ | 10 | 1 | ➔ 1/11 of CPU is wasted !

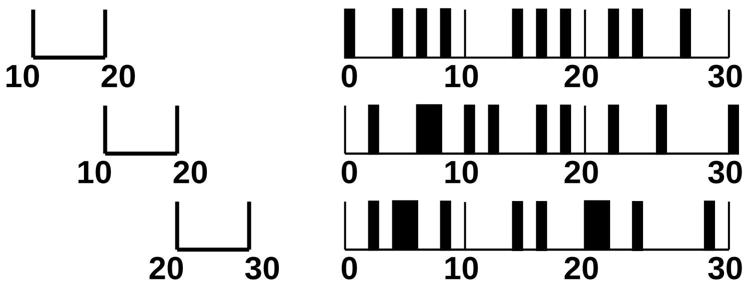
-- Turnaround Time (與 time Q 有關)

Process(10 ms) **Quantum = 10** **Quantum = 1**

P1

P2

P3



Average Turnaround Time = (10+20+30)/3 = 20 ATT = (28+29+30)/3 = 29

➔ 80% CPU burst < time slice

- Good for time-sharing systems
- <u>Preemptive</u>
- The performance of the RR algorithm depends on the size of the time quantum
- If the time quantum is very large (infinite) ➔ as FCFS.

(* cause poor response to short interactive requests *)

- If it is very small ➔ called processor sharing

(* too many processes switches, lower CPU efficiency *)

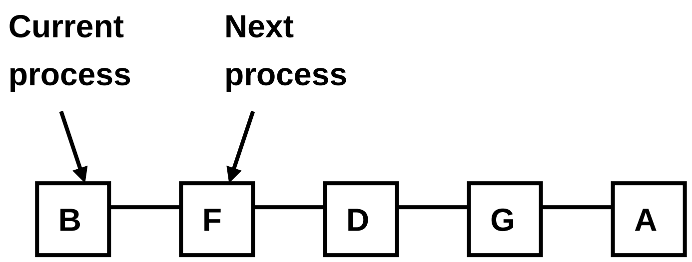
EX. Time quantum = 4

The average waiting time is 17/3 = 5.66

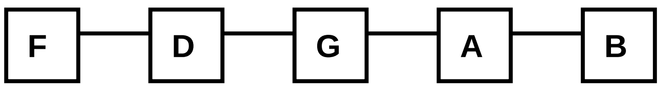
Process	**Burst Time**
P1	**24**
P2	**3**
P3	**3**

P1	P2	P3	P1	P1	P1	P1	P1

0 4 7 10 14 18 22 26 30



(a)

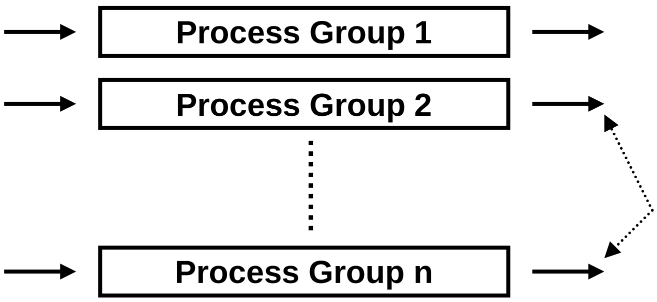


(b)

(5) Multilevel Queue Scheduling

Partition the ready queue into several separate queues

➔ processes can be classified into different groups and permanently assigned to one queue



Ex：
Foreground (interactive)；
background (batch)

各用各的scheduling方法！！

-- Intra-queue scheduling (互不相關)
Independent choice of scheduling algorithm

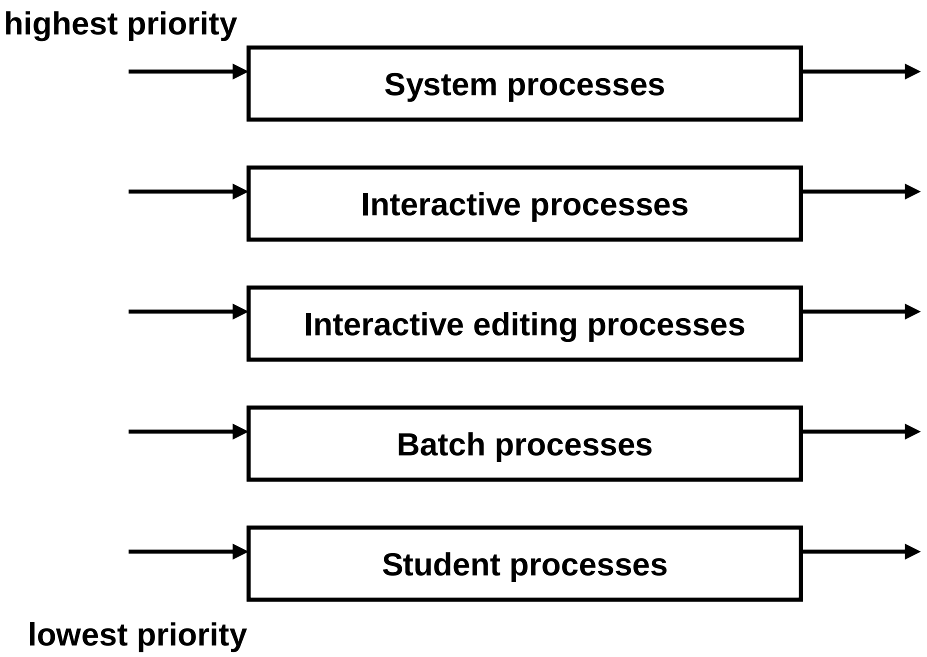
-- Inter-queue scheduling

a. Fixed-priority preemptive scheduling, e.g., foreground queues always have absolute priorities over background queues.
b. Time slice between queue, e.g., 80% CPU to give foreground processes and 20% CPU to give to background processes (queues)
c. And more ?!!!

共(n+1) scheduling 方法

- Processes are classified into different groups.
- Foreground (interactive), background (batch) processes.
- Processes are permanently assigned to one queue.
  (* based on some priority *)
  (* processes do not move between queues. *)

- Each queue has its own scheduling algorithm.
  (N queues with (N+1) scheduling algorithms)
- There must be scheduling between the queues.
  (* a fixed-priority preemptive scheduling *)
- Another possibility is to time slice between the queues.
  (* 80% CPU time；foreground queue with RR algorithm *)
  (* 20% CPU time；background queue with FCFS *)



(6) Multilevel Feedback Queue Scheduling

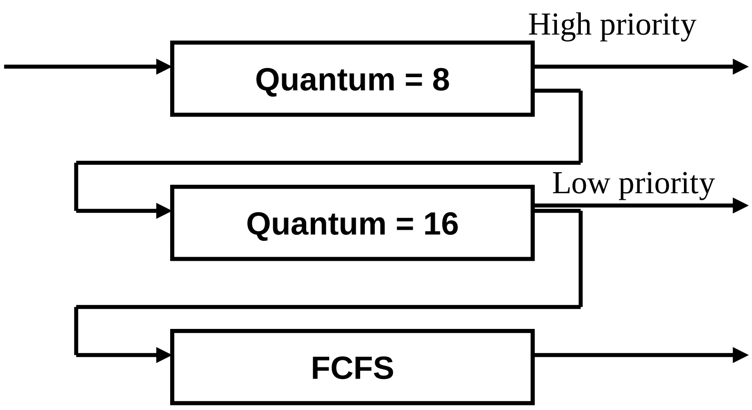
Different from Multilevel Queue Scheduling by allowing processes to migrate among queues.

-- Parameters (configurable !)

a. # of queues
b. The scheduling algorithm for each queue
c. The method to determine when to upgrade a process to a higher priority queue
d. The method to determine when to demote a process to a lower priority queue
e. The method to determine which queue a newly ready process will enter

∗ Inter-queue scheduling：Fixed-priority preemptive ?!

Example



∗ **Idea：separate processes with different CPU-burst characteristics！**

- allow a process to move between queues.
- separate processes with different CPU-burst characteristics.
- if a process uses too much CPU time => move to a lower-priority queue.
- if a process waits too long in a lower-priority queue => move to a higher-priority queue.

(7)Multiple-Processor-scheduling
(Load Balancing)

CPU scheduling in a system with multiple CPUs.

-- A homogeneous environment

processes are identical in terms of their functionality

=> Can processes run on any processor？

Any libations if special peripheral devices exist in certain nodes

-- A heterogeneous environment

processors must be compile to the compiled codes of programs

-- Load sharing～load balancing！！

A common ready queue for a number of processes

=>

1. Self-scheduling～symmetric multiprocessing

   Need synchronization to access common data structures, e.g., queues

2. Master-slave structure～asymmetric multiprocessing

   One processor as scheduler