4.Processes

* **原因：Concurrent execution. (multiprogramming)**

* **定義：**

– A program in execution. (≠ program)

– Basic unit of work in OS   (for cpu scheduling)

– 含 OS process 與 user’s process

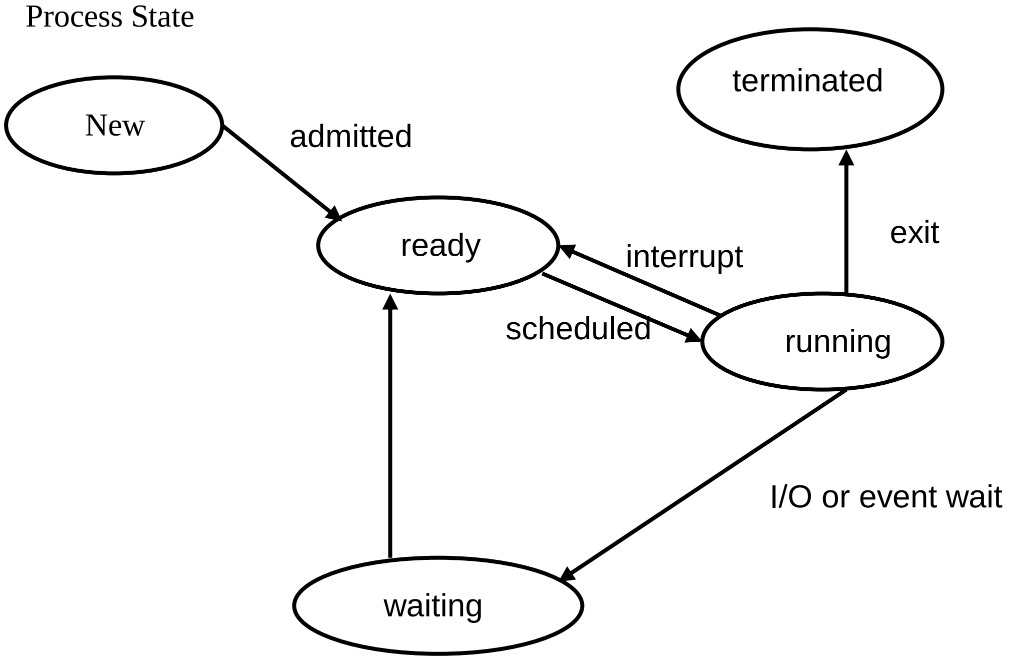
– an active entry vs. program (被動)

– need ? resources

– a “running” car!

– 記錄其 current activity,含 program counter, register, stack, data segment (open 那些 file,讀到那…)

Process State



* **Process control block (PCB)**

– Process state

– Program counter

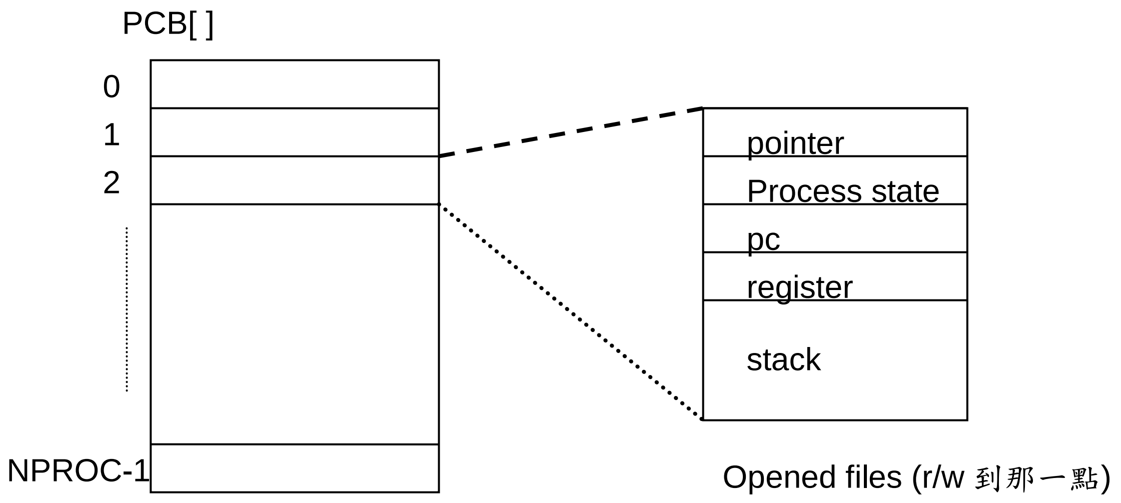
– CPU register

– CPU scheduling (ex.已執行多久了)

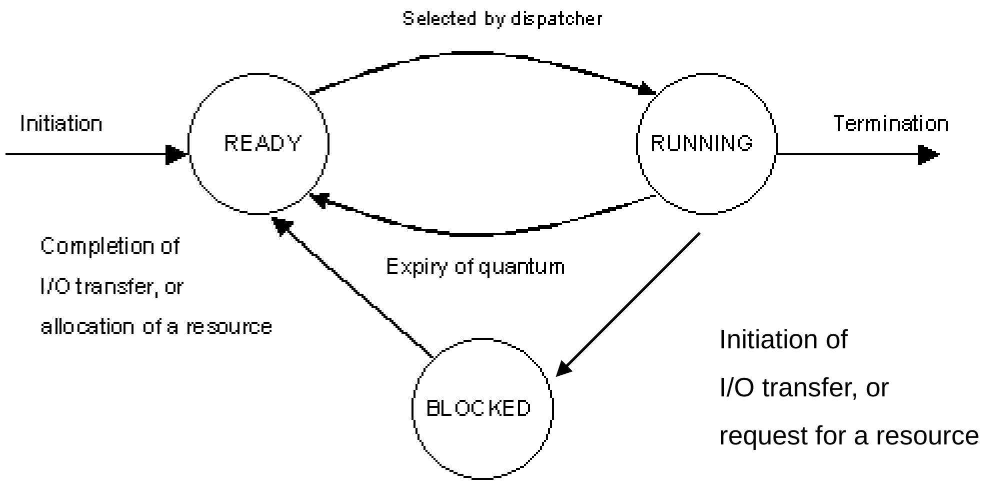
– Memory – management (上下限)

– Accounting information

– I/O status information (已 open 哪些 files)



* Process vs. thread (a light weight process)



pointer	Process state
Process number	
Program counter	
registers	
Memory limits	
List of open files	
______	
______	

Process Control Block (PCB)

*　程序定義

所謂程序(process)是指執行中的程式，由於程式中的指令是由 computation 指令與 I/O 指令所構成，因此可說程序是一序列的 CPU burst 與 I/O burst 所構成，如下圖即是程序執行中的狀況。

程序的定義可視為如下敘述：

- a program in execution
- an asynchronous activity
- the “animated spirit” of a procedure
- the “locus of control” of a procedure in execution
- that entity to which processors are assigned
- the “dispatchable” unit

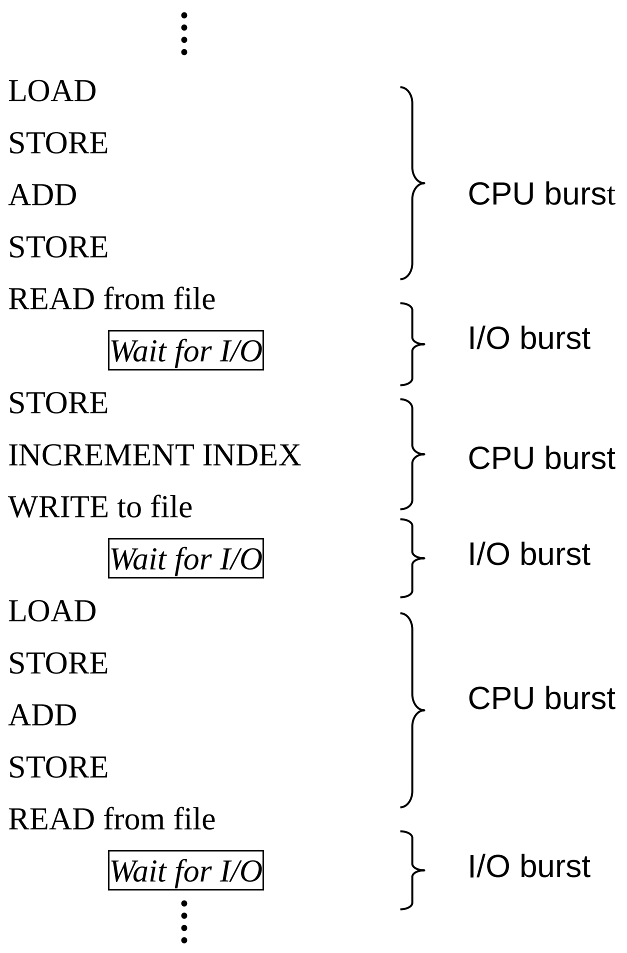


Fig. CPU 與 I/O burst 的交替執行系列

* 程序狀態(process state)

程式在執行時，可說程序經過了狀態的變化。若程序是在”running”狀態，則表示此程序正在使用 CPU。若程序是在”ready”狀態，則此程序處於可被執行的狀態，但未真正由 CPU 執行。若程序是在”blocked”（或”waited”），則此程序等待某些事件（event）的發生（如 I/O 的完成）後方能再度成為可被執行的狀態。若程序是在”new”的狀態，則表示此程序還在”job pool”中而未準備進入可執行狀態。程序狀態的變化表示於下圖中。

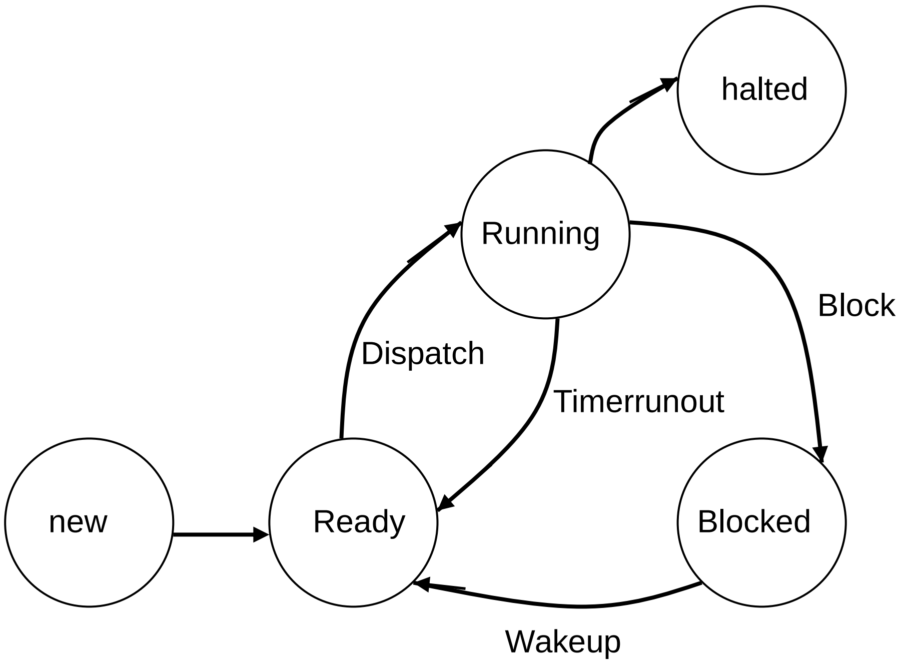


Fig.程序狀態圖

Dispatch (process name): ready→running
Timerrunout (process name): running→ready
Block (process name): running→blocked
Wakeup (process name): blocked→ready

注意：由程序自己所啟動的狀態轉移僅是 block，而其他三種狀態轉移是由程序外的 entities（如 O.S 或 timer）所啟動。

* 程序控制結構(process control block)

在作業系統中表示程序的資料結構是程序控制結構（簡稱 PCB），其內容如下圖所示，共記錄有：

1. 程序狀態：new, running, ready, waiting,或 halted。

2. 程式計數器（program counter）：指示程序的下一個執行指令。

3. CPU 暫存器內容：包括有累積器，索引暫存器，一般資料暫存器與狀態旗標暫存器。

4. 記憶管理資料：base 與 bound register 或 page table。

5. 統計資料：如 CPU 的使用時間，時間的限制值，程序的總數（process number）等。

6. I/O 狀態資料（I/O status information）：未完成的 I/O 需求（I/O request），配置到此程序的 I/O 設備，與所 open 的檔案等。

7. CPU 規劃資料（CPU scheduling information）：此程序的優先前（priority），指到 scheduling queue 的指標值與其他相關的 scheduling parameters。

pointer	Process state
Process number	
Program counter	
Registers	
Memory limits	
List of open files	
⋮	

Fig.程序控制結構

*   **Process scheduling**

Basic goal of <u>multiprogramming</u>, maximize CPU utilization!

– Scheduling queues (各種 queue)

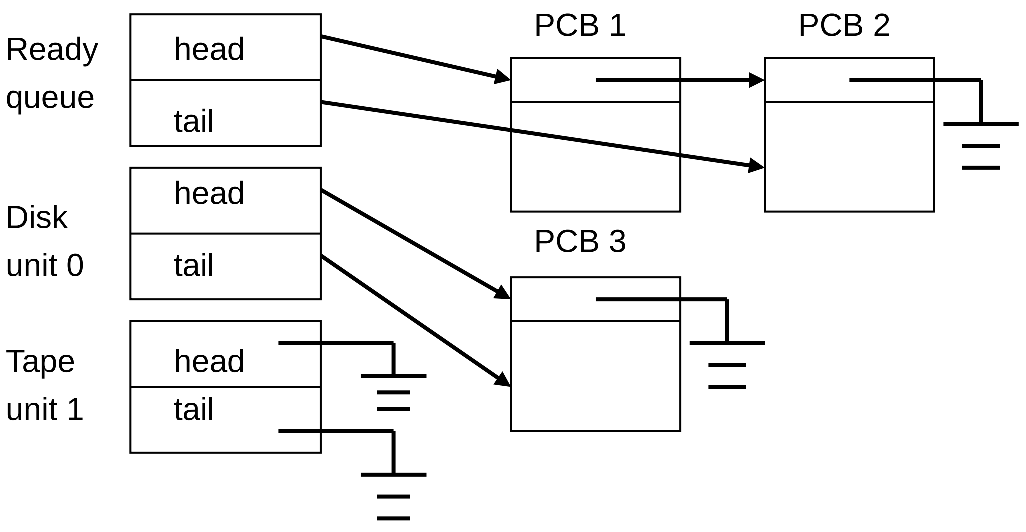
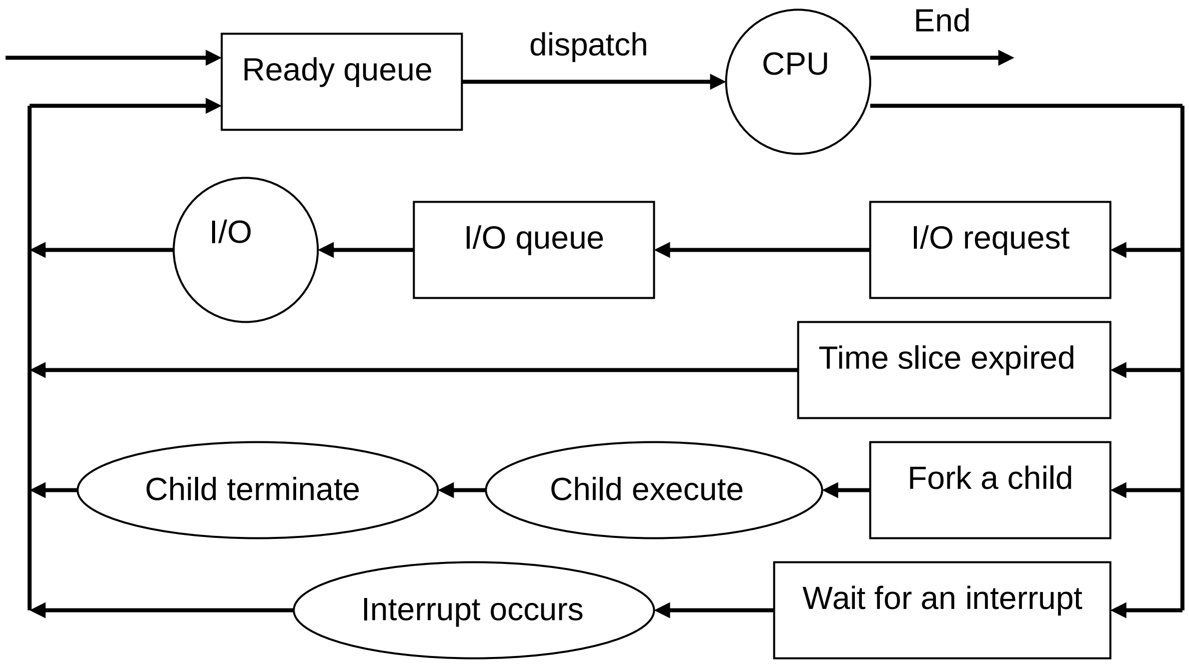


Fig. 4.4

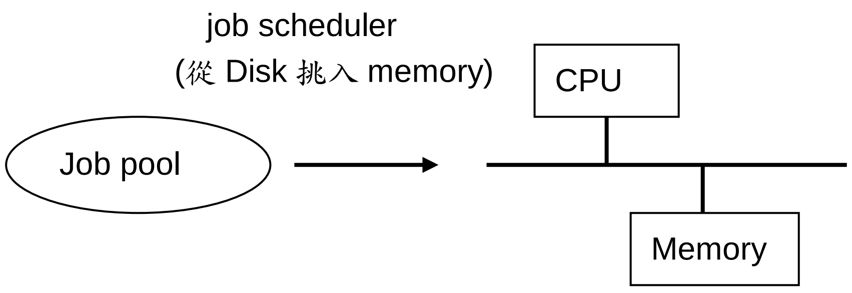


Queue Diagram

Fig. 4.5

* Schedulers (select process)

– Long-term scheduler (當 job 離開後，才需要 long-term schedule)
long-term 與 short-term 區分 ⟹ frequency of execution



Goal: select a good mix of I/O-bounded and
CPU-bounded processes

Remarks:

a. control the degree of multiprogramming.(能有多少人在 memory 中)

b. can take more time in selecting a process because of longer interval between execution.

– Short-term ~ CPU scheduler (已在 memory 中)

Goal: efficiently allocate the CPU to one of the ready
processes according to some criteria.

– mid-term scheduler
swap processes（已在執行中）in and out memory to control the degree of multiprogramming. (ex. threshing 時，必須剔一些掉)

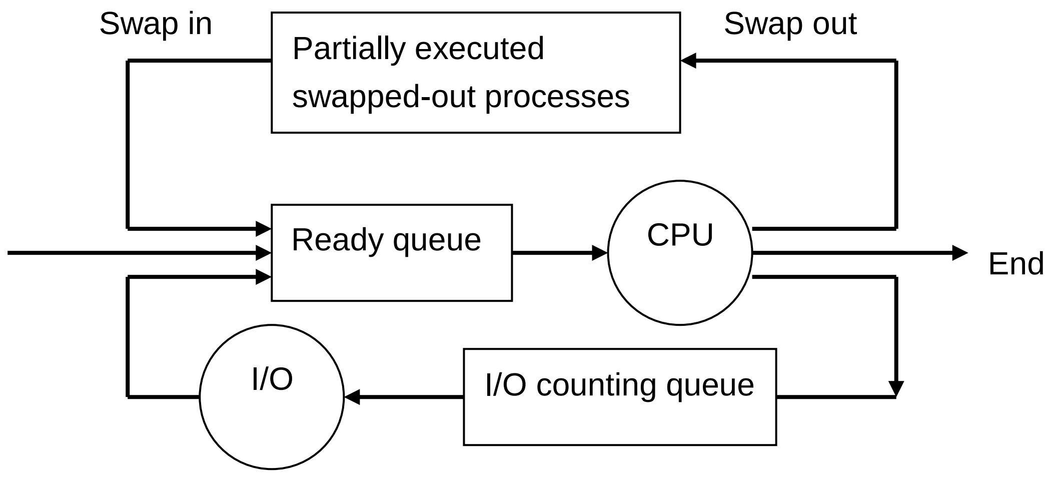
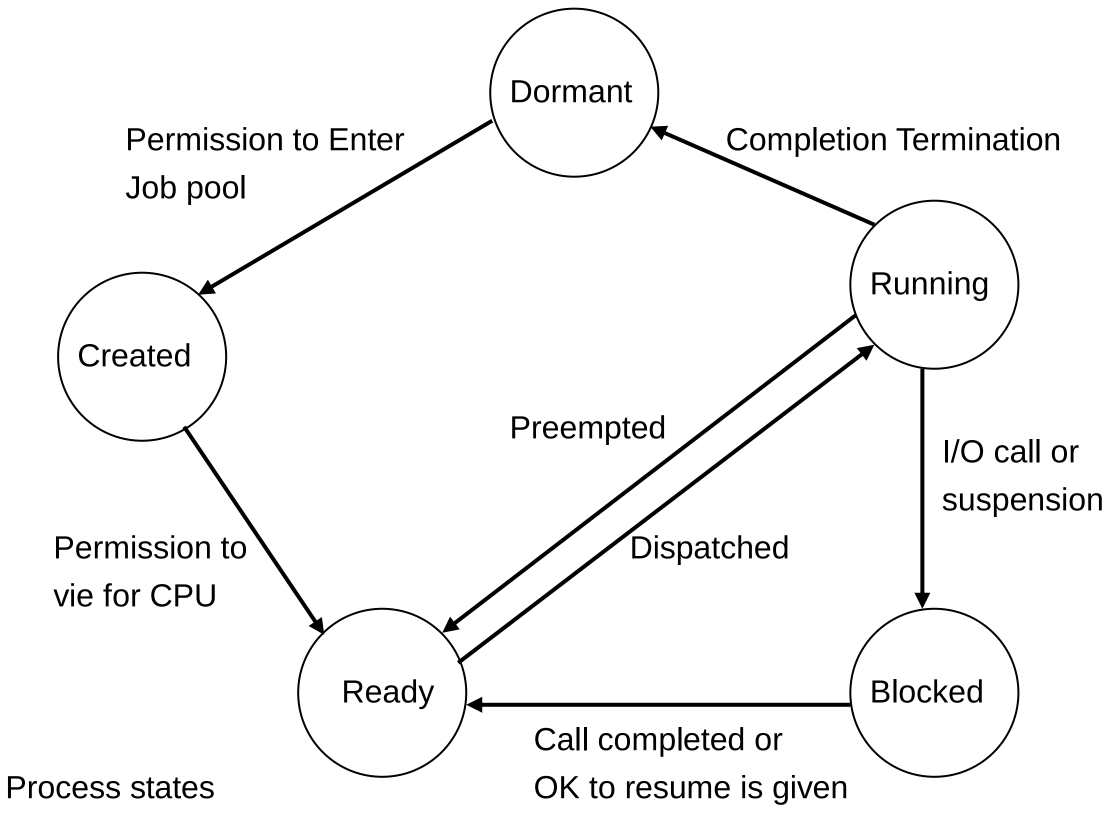


Fig. 4.6



Short-term scheduler

* **context switch ~pure overhead (switching CPU to another process) (* take time *)**
saving the state of the old process and loading the state of newly scheduling process.

Issue:

- Cost depends on hardware support
  e.g., multiple register sets

- Threads, light-weight process (LWP) are introduced to break this bottleneck!

* Operation on processes

– Process creation & termination
(restricting a child process to a subset of the parent’s resources prevents any process from overloading the system by creating too many subprocesses.)
Issue:

– Restrictions on resource usage.

– Concurrent execution.

– Process duplication or reconstruction.

– Cascading termination (VMS).
(父親 if 不存在→子就不存在，tree 觀念)

Example:

```
So;
if (proc_id=fork() )==o {
…..}
/* code executed by child process */  →a tree 馬上得到
else {                                  resources
…..
/* code executed by parent process */
/* wait() for child to die */
}
```

. a copy of parent address space +
  context is made for child. <u>Except</u>, on return from fork():
    . child returns with a value 0
    . parent returns with process <u>id of child</u>
. No shared data structures between parent and child
  ⇨ communicate via shared files + pipes
. use execve() to load a new program

UNIX

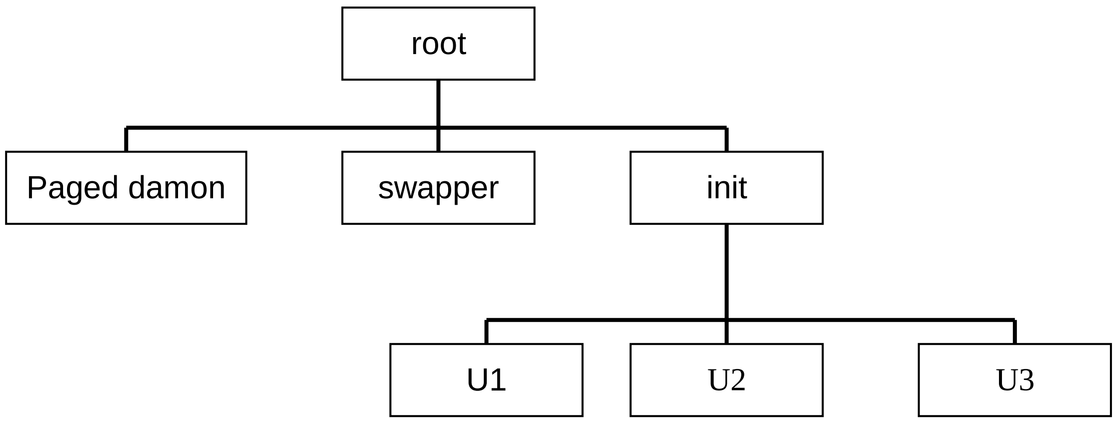


Fig. 4.7

1. 父親與子同執行 → 〈 子與父同一程式
子與父不同程式

2. 父 → **Wait** ↗

* 程序得操作

作業系統提供一些功能可以操作程序，這些操作有：

– create a process

– destroy a process

– suspend a process

– resume a process

– change a process’s priority

– block a process

– wakeup a process

– dispatch a process

建立一個 process 的步驟

– name the process

– insert it in the system’s list of known process

– determine the process’s initial priority

– create the process control block

– allocate the process’s initial resources

* 層次性建立方式

一個程序可以自己生長出一個新的程序，此新的程序稱為子輩程序（child process），而原來的程序稱為父輩程序（parent process）。而此二程序接能同時存在而執行。此種建立子輩程序的方式（下圖所示）稱為層次性程序結構（hierarchical process structure）。

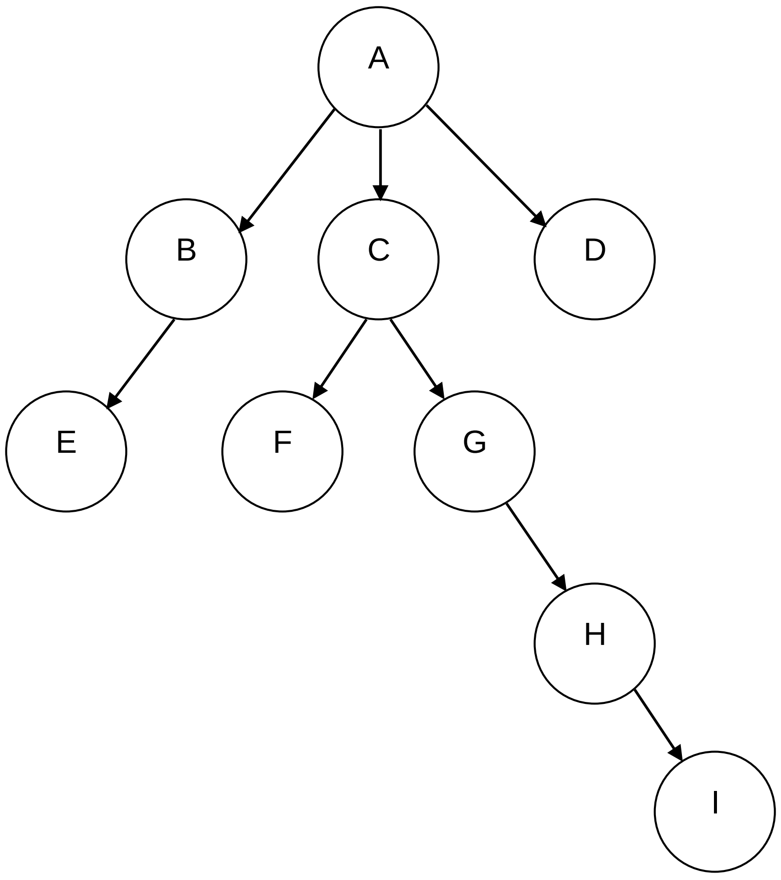


Fig. 層次性程序結構

* 暫停（suspend）與取回（resume）

所謂暫停一個程序是指暫時使此程序不再執行(並非此程序)。一個被暫停的程序一直都不能被執行直到另一個程序取回(resume)它方可再度執行。提供此兩功能的理由有：

1. 系統在功能上出了問題，可暫停程序的執行直到恢復功能方取回執行。

2. 使用者懷疑程式有誤，因此暫停執行，直到確認正確後，再取回執行。

3. 為了平衡系統負載，當系統負載過多時（欲被執行的程序過多）則可暫停一些程序，直到負載減輕時，方取回執行。

增加此兩功能後，程序狀態變化擴展城下圖所示。其中所增加的狀態轉移有：
suspend (processname) : ready→suspended ready
resume (processname) : suspended ready→ready
suspend (processname) : blocked→suspended blocked
resume (processname) : suspended blocked→blocked
completion (processname) : suspended blocked→
suspended ready

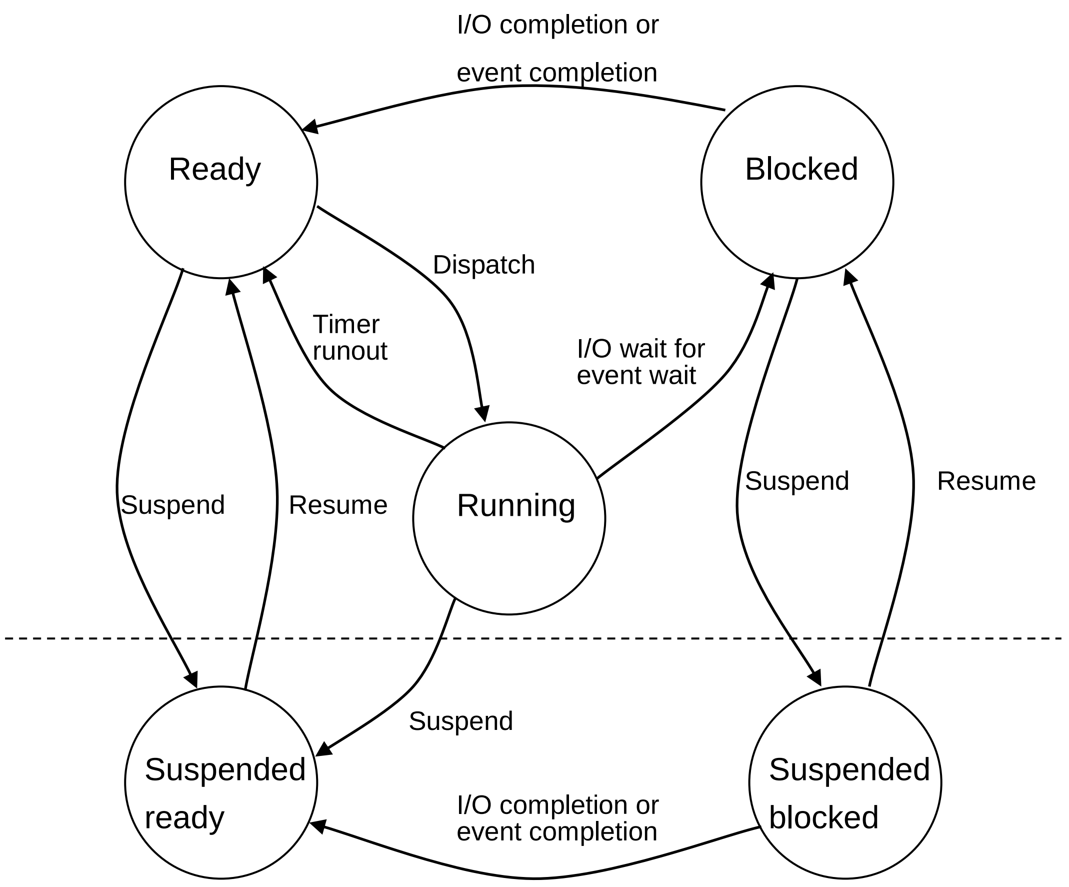
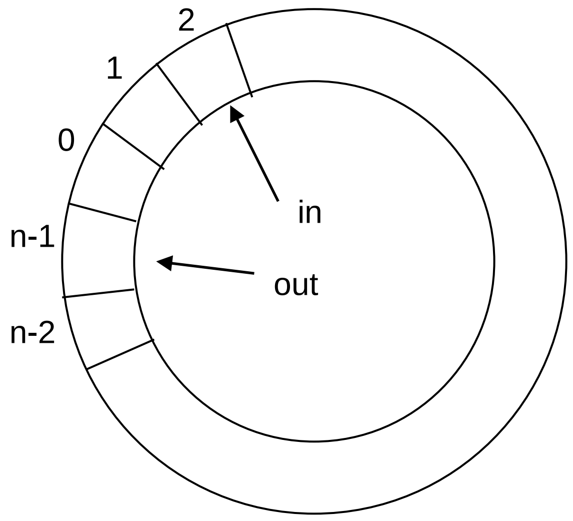


Fig. 具有 suspend 與 resume 的程序狀態圖

* Cooperating processes

cooperating processes can affect or be affected by the other processes. (ex. Share data)



Buffer [0..n-1]

Initially, in=out=0;

（表示全空）

(* 不能放太快，也不能 read 太快→need synchronization *)
Producer:

```
        repeat
           …..
           produce an item nextp;
           …..
           while ((in+1) mod n) = out do no-op;(*表示(n-1)滿*)
           buffer [in] := nextp;
           in := (in+1) mod n;
        (*Load in; Add 1 in, mod n, store in *)
    until false;
```

Considering: n=7
In: slot 0,1,2,3,4,5 have been filled up ➔ in then = 6,
next time, stop
Or
Out: 0, stop

Consumer: (*只放 n-1 items *)

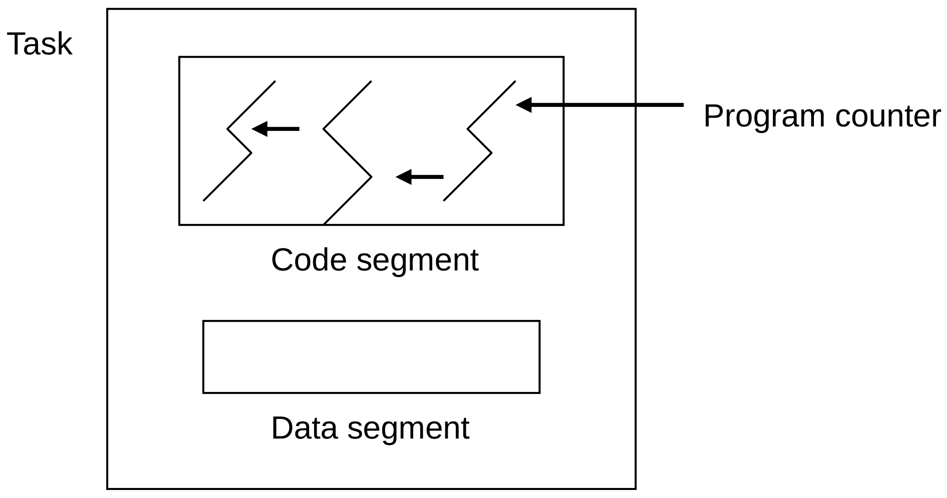
```
    repeat                                    (*為了避免判斷不出
        while in = out do no-op;              in= out 表全滿 or 全空*)
        nextc := buffer [out];
        out := (out+1) mod n;
        .....
        consume the item in nextc;
    until false;
```

* **4.5 Threads**

Reduce context switch overhead

– Light weight process (LWP): a basic unit of CPU utilization.

– Threads consist of
  program counter
  a register set
  a stack space
  但 share code, data



– A (traditional) process, i.e.., a heavy weight process, is a take with one thread!

– Threads creation and context switching is efficient at the cost of no mutual protection.
ex, producer + consumer, 打橋牌

Different threads in a process are not quite as independent as different processes, however. All threads have exactly the same address space, which means that they also share the same global variables. Since every thread can access every virtual address, one thread can read, write, or even completely wipe out another thread’s stack. There is no protection between threads because (1) it is impossible, and (2) it should not be necessary. Unlike different processes, which may be form different users and which may be hostile to one another, a process is always owned by a single user, who had presumably created multiple threads so that they can cooperate, not fight. In addition to sharing an address space, all the threads share the same set of open files, child processes, timers, and signals, etc. as shown in Fig. Thus the organization of Fig would be used when the three processes are essentially unrelated, whereas Fig would be appropriate when the three threads are actually part of the same job and are actively and closely cooperating with each other.

Per thread items	Per process items
Program counter	Address space
Stack	Global variable
Register set	Open files
Child threads	Child processes
State	Timers
	Signals
	Semaphores
	Accounting information

Fig. Per thread and per process concepts.

Like traditional processes (i.e., process with only one thread), threads can be in any one of several states: running, blocked, ready, or terminated. A running thread currently has the CPU and is active. A blocked thread is waiting for another thread to unblock it (e.g., on a semaphore). A ready thread is scheduled to run, and will as soon as its turn comes up. Finally, a terminated thread is one that has exited, but which has not yet been collected by its parent (in UNIX terms, the parent thread has not yet done a WAIT).

* **Interprocess**

– shared memory

– message passing ~ gernel properties such as msg size,comm.,links, …

– Naming
Process must have a way to refer to each other!

(A) Direct Communication
Process must explicit name the recipient or sender of a communication
send (P, msg)
receive (Q, msg)
properties:

a. Communication links are established automatically.

b. Two process / a link

c. One link / a pair of processes

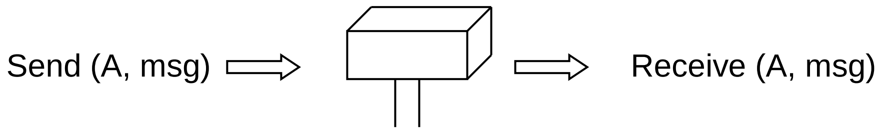
d. Bidirectional or unidirectional

Issues:
Symmetric addressing vs. asymmetric addressing
receive (id, msg)

Difficulty:
Process naming vs. modulability
（雙方不得任意改名→否則下次找不到）

(B) Indirect Communication

Two processes can communicate only if the process share a mailbox (or ports)



Properties:

a. A link is established between a pair of processes only if they share a mailbox.

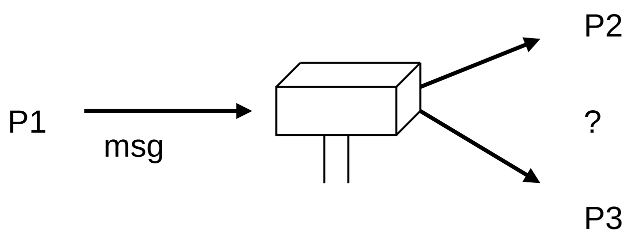
b. N processes / link & n>=2

c. N links / a pair of processes & n>=1
one link / a mailbox

d. Bidirectional or unidirectional

Issues:

a. Who is the recipient of a message (msg)? (不知)



b. Owners vs. users

- process→ owner as the sole recipient?
  privileges can be passed?
- O.S.→creator as owner?
  privileges can be passed?
  garbage collection?

* Interprocess communication 的定義

允許程序間做資料訊息的傳送稱之。資料傳送的媒介有二種方式。

1. shared memory: 建立一個 buffer 做兩個 processes 間的資料存取，如 producer / consumer。

2. message system: 由 OS 在邏輯階段建立兩個 processes 間 communication link 的。經由此就可做資料傳送。

* 通信路徑 (communication link) 的建立方式

1. 直接通信：
   由作業系統直接提供系統呼叫（system call）以指示 processes 的通信對象，如

   send (P, message): Send a message to process P.
   receive (Q, message): Receive a message from process Q.

例題：producer 與 consumer 兩 processes 間的直接通信，其程式設計如下：

```
type item = …;
   var nextp, nextc: item;
   parbegin
        producer: repeat
                                 …..
                              produce an item in nextp;
                              send (consumer, nextp);
                         until false;
        consumer: repeat
                              receive (producer, nextc);
                                 …..
                              consume the item in nextc
                         until false;
   parend;
```

(* buffer 在哪？系統負責*)

2. 間接通信：
   不直接指出通信 processes 的名稱，而是經由一個稱為 mailbox 的 buffer 做通信埠（port），即通信的 processes 傳送資料皆以此 mailbox 為主，例如：

   send (A, message): Send a message to mailbox A.
   receive (A, message): Receive a message from mailbox A.

* **Buffering**

A link capacity → # of msgs held in a link

A. Zero capacity (no buffering)
   Msg transfer must be synchronized
   ~ rendezvous! （一送 – 馬上收）

B. Bounded capacity
   Sender can continue execution without waiting till the link is full!

C. Unbounded capacity
   Sender is never delayed!

* Items B & C are for asynchronous communication and may need acknowledgement.

– Special cases

a. Msgs may be lost if the receiver can not catch up with msg sending → synchronization

b. Sender are blocked until the receiver have received msgs and replied by reply msgs
   →Remote Procedure Call (RPC) framework

* **Exception Conditions**

– Process termination （其中有一方已不存在）

a. Sender termination → notify or terminate receiver!

b. Receiver termination
no capacity → server is blocked
buffering → msgs are accumulated!

– Lost msg (due to hardware or network failure)
Ways to recover errors:

a. O.S. detects & resends msgs

b. Sender detects & resends msgs

c. O.S. detects & notify sender to handle it.

Issue:

a. Detecting methods such as <u>timeout</u>!

b. Distinguish <u>multiple</u> copies if retransmitting is possible.

– Scrambled messages
Usually, O.S. adds checksums such as CRC inside messages & resend them as necessary!