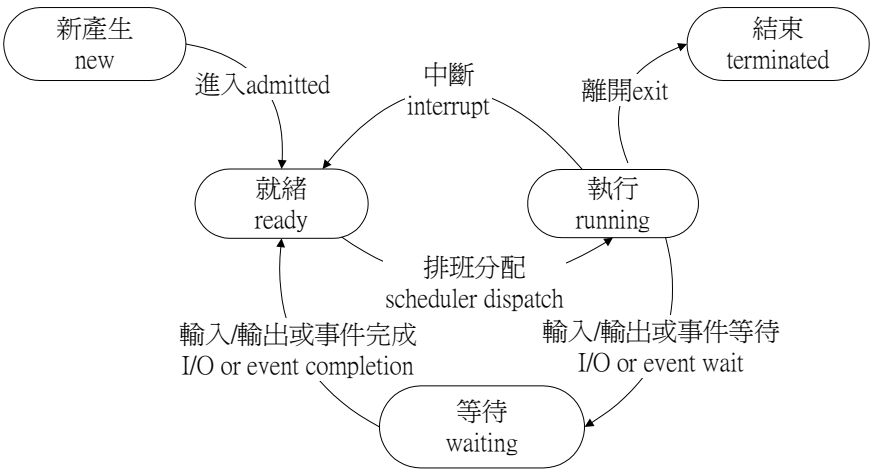
第 18 章
行　程

第 18 章 行　程

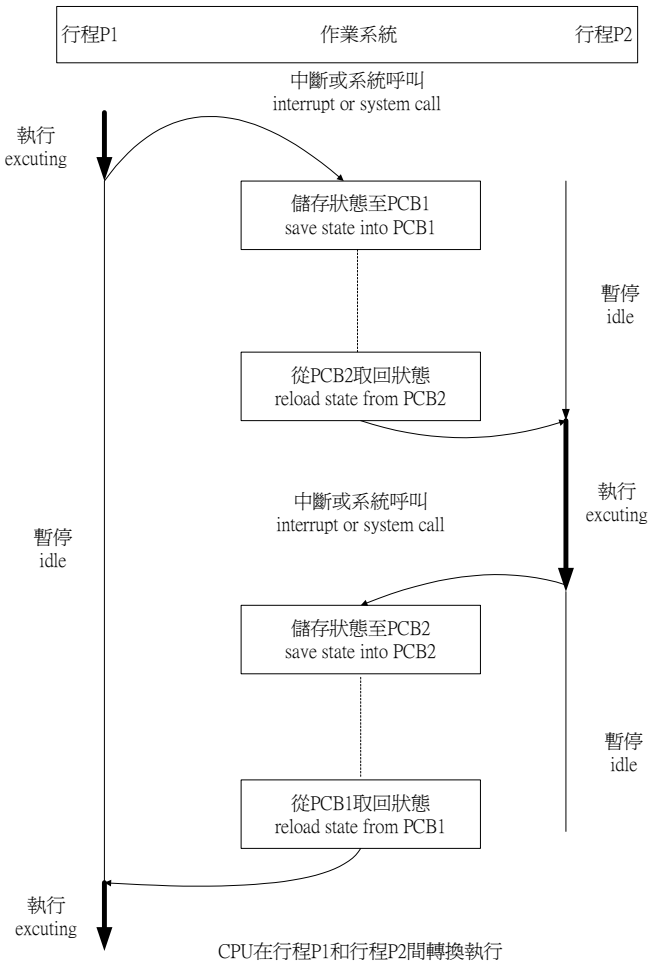
一個行程就是一個程式正在執行。當我們在終端機下達指令時，Linux 就會建立一個行程，而當我們的程式執行完時，這個行程就被終止了。在一個分時系統的 Linux 作業系統，允許多個使用者使用電腦系統，而許多行程也同步的被執行。像我們的個人電腦，一般只有一顆 CPU 中央處理器，但卻同時的處理多個行程，而這就分是分時系統。

當我們執行程式時，電腦作業系統就會使用 new 幫我們產生新的行程，而當我們的行程就緒(ready)時，而核心又排班分配 CPU 中央處理器給他時，他就會開始執行 running，直到執行結束 terminated(或 Zombie)。當然在過程中也有可能發生像系統呼叫的 System call 而發生中斷，或者改成其它行程執行(被 preempt)，這時我們的行程就會回到就緒 ready 的狀態。當然在過程中也可能發生像輸入/輸出 I/O 或事件等待(sleep)，此時，CPU 是在閒置 waiting 的狀態，而當我們的輸入/輸出或事件完成時(被 wake up)，才會到就緒 ready，等待下一次排班分配 CPU 中央處理器，直到結束(zombie 或 terminated)。



行程狀態圖

這是分時系統當行程 P1 和行程 P2 作執行上的內容轉換 Content switch。當行程 P1 在執行時，因為作業系統的中斷或系統呼叫，這時行程 P1 就要先將它的 Proccess Control Block(行程控制表 PCB1)給除儲起來。而這時後 CPU 中央處理器會載入所呼叫的行程 P2，這時會載入行程 P2 的 Proccess Control Block(行程控制表 PCB2)，這時 CPU 中央處理器就會執行行程 P2，而當執行行程 P2 一定的時間，又會將行程 P2 的 Proccess Control Block(行程控制表 PCB2)給儲存起來。而此時 CPU 中央處理器就會載入行程 P1，而執行行程 P1。



CPU在行程P1和行程P2間轉換執行

這就是行程狀態控制表。PCB(Proccess Control Block)含有行程的許多下列的內部資訊。

行程狀態：可以是 new、ready、running、waiting 或 halted。

程式計數器：指明該行程下次要執行的指令位置。

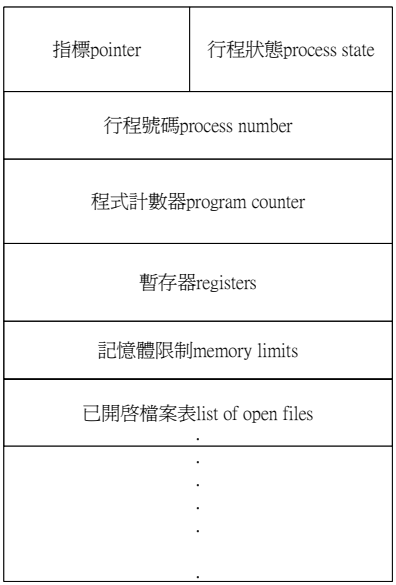
CPU 暫存器：其數量和類別依不同形式 CPU 而有不同。有累加器(accumulators)、索引暫存器(index register)、堆疊指標(stack pointer)、位址暫存器(address register)….。

記憶體管理資訊：這包括存取記憶體的基底暫存器(base register)和限制暫存器(limit register)、分頁表(page table)或記憶體系統的分段表(segment table)的資訊。

CPU 排班法則和相關資訊：包括行程的優先順序、排班法則。

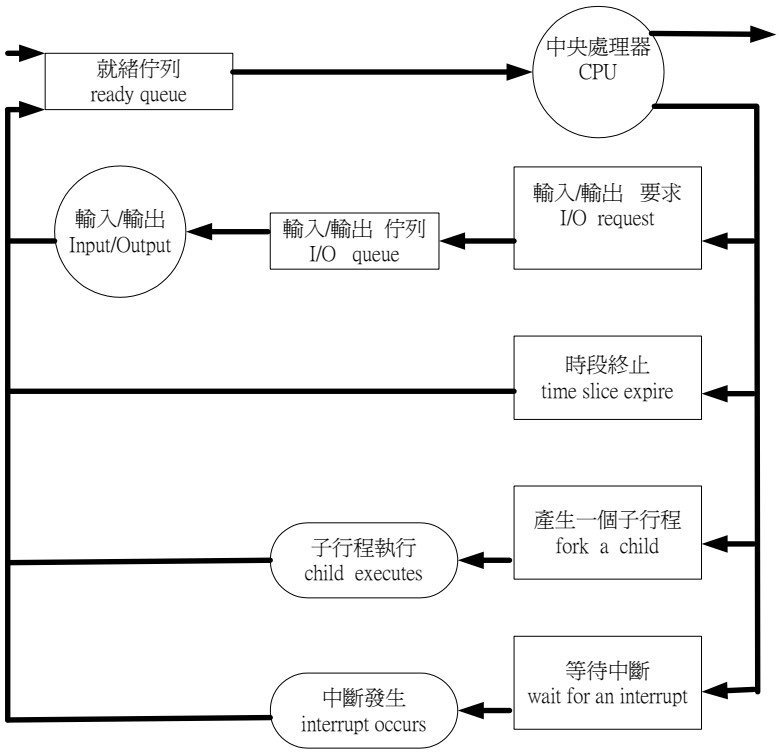
CPU 會計資訊：CPU 的使用時間、帳號、工作和行程的編號。

輸入/輸出狀態資訊：這包括了行程的輸入/輸出裝置、開啟檔案的串列。



行程控制表PCB

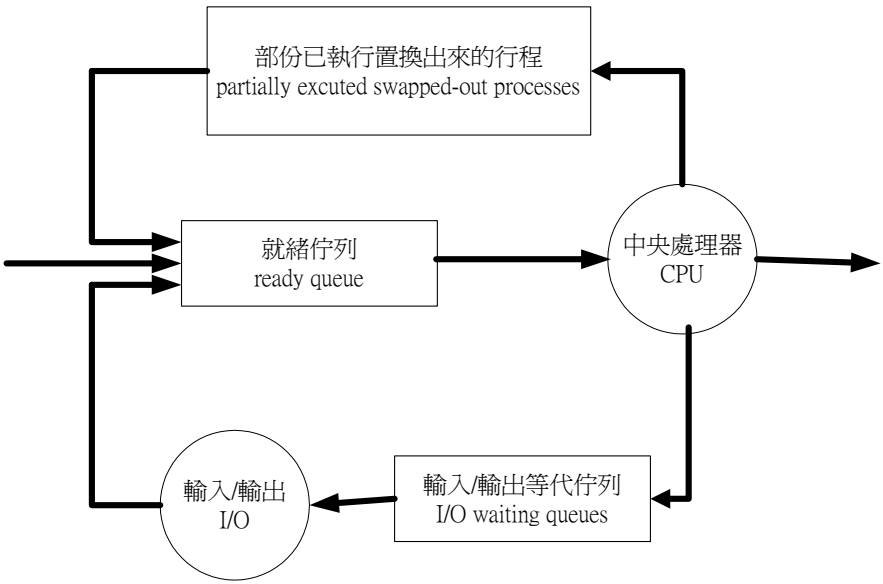
分時系統的目地是將 CPU 中央處理器在不同的行程之間不斷的轉換，以便讓使用者可以在自己的行程執行時與它交談。我們個人電腦的單一 CPU 處理器，不可能多個行程同時執行，而它們必需在一旁的佇列中等待，一直到 CPU 有空時，才可能排到它們執行。當我們新增 new 一個新的行程時，它最初是置於就緒佇列中。每一個長方形就是一個佇列，圓圈代表執行行程的資源，箭頭代表執行的方向。當我們所執行的程式或行程在進行時，它會先配置 CPU，然後有可能發出輸入/輸出的要求，然後置於一個 I/O 佇列中。程式也可能 fork 產生新的行程。行程也可強行的離開 CPU 然後就回到就緒佇列中。



行程排班佇列圖

在分時系統中一個行程在它整個生命期，將在各種不同的排班佇列中，這當然要看我們的需要設計，以及作業系統的設計。因此作業系統需按排班方式從佇列中選出行程，讓所選出的行程讓中央處理器 CPU 來執行。而我們所執行的程式也可能只執行一下子，然後就等待輸入/輸出 I/O 的要求，但 I/O 的時間，比 CPU 所需執行的時間大上很多。在整批作業系統中，大部份的行程都會以 spooling 儲存池的方式存到硬碟或大型儲存裝置，等待一個一個的執行。

分時系統的中程排班程式是將行程從記憶體中移開，因此而降低多元程式規化的程度，然後再將行程放回記憶體中，並且放在它移開前的位置繼續執行，這種方法稱為 swapping，如下圖所示。



加入中程排班的佇列圖

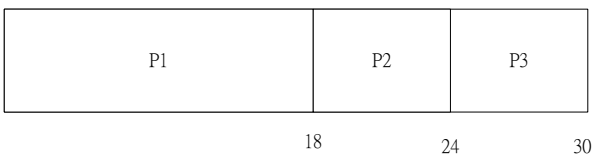
CPU 排班是多元程式規畫作業系統的基礎，藉由不同的行程在 CPU 執行中作轉換，讓我們電腦的輸出產量提高。我們在分時系統的作業系統 Linux 中，CPU 一般都是可搶先排班(Preemptive Scheduling)，也就是當某個程式 P1 在執行時(某個行程在使用 CPU 資源)，此時，另外一個程式 P2 可以強奪 P1 程式 CPU 的資源，而此時則是 P2 程式在執行。我們有多種 CPU 排班法可以來決定是否使用這些排班，而它們評估的標準有 CPU 的使用率、CPU 的產量、每一個行程回復時間、行程提出要求到第一個反應出現所需的時間。

CPU 排班是決定將 CPU 分配給就緒佇列中的那一個行程，我們將介紹先來先做的排班方法、最短的工作先作的排班方法、優先順序的排班方法和依序循環的排班方法。

這是先來先作的排班法則，也就是 FCFS 的演算法，行程 P1 第一個到，因此先將 CPU 分配給它，等到 P1 行程執行完再執行行程 P2。P3 則是最後執行。當 P1 行成執行完 18 個單位時間，P2 才開始執行，此時 P2 已經等待了 18 個單位時間。同理，P3 等待了 24 個單位時間。因此平均等待時間為(18+24)/3=14 秒。這個圖又稱為甘特圖。

先來先作排班法First Come , First Served Scheduling

行程	分割時間
P1	18
P2	6
P3	6

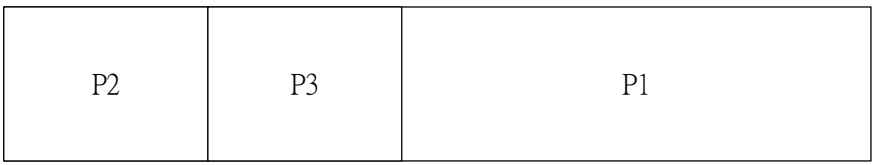


行程P1 的等待時間是0 ， 行程P2的等待時間是18， 行程P3的等待時間是24 . 因此平均等待時間是 (0+18+24) / 3 = 14 .

這是我們最短的工作(行程 Process)先作的排班法。因為行程 P2 和行程 P3 所需分割時間同為 6，因此可以作為先分配 CPU。而行程 P1 所需分割的時間最長，所以最後才分配 CPU 給行程 P1，此時 P1 已經等待了 12 個單位的時間。因此平均行程所等待的時間 P1 為 12、P2 為 0、P3 為 6，所以平均等待的時間為(12+0+6)/3=6。

最短的工作先作排班法Shortest Job First Scheduling

行程	分割時間
P1	18
P2	6
P3	6

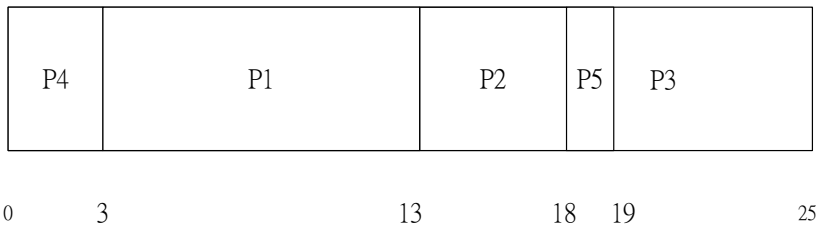


這是我們最短的工作(行程Process)先作的排班法。因爲行程P2和行程P3所需分割時間同爲6，因此可以作爲先分配CPU。而行程P1所需分割的時間最長，所以最後才分配CPU給行程P1，此時P1已經等待了12個單位的時間。因此平均行程所等待的時間P1爲12、P2爲0、P3爲6，所以平均等待的時間爲(12+0+6)/3=6。

在優先權的排班法則中行程 P1 的優先權第 2，而它的等待時間是 3。行程 P2 的優先權是第 3，因此在甘特圖中它的等待時間是 13。行程 P3 的優先權是第 5，因此在甘特圖中，它的等待時間是 19。行程 P4 的優先權是第 1，因此等待時間是 0。行程 P5 的優先權是第 4，等待時間是 18。 因此平均等待時間是(3+13+19+0+18) / 5 = 10.6 .

優先權的排班法Priority Scheduling

行程	優先順序	分割時間
P1	2	10
P2	3	5
P3	5	6
P4	1	3
P5	4	1

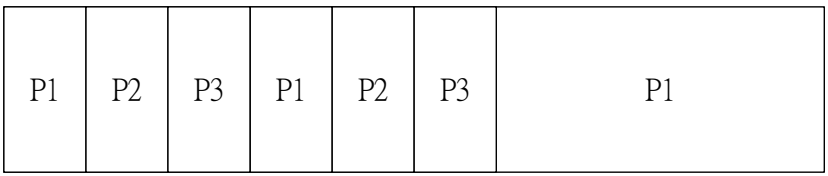


行程P1 的等待時間是3 ，行程P2的等待時間是13， 行程P3的等待時間是19 . 行程P4的等待時間是0， 行程P5的等待時間是18 因此平均等待時間是 (3+13+19+0+18) / 5 = 10.6 .

這是我們 Round-Robin 依序循環的工作排班法，我們假設每次分配 CPU 給行程 3 個單位的時間。P2 在作完行程前總共等待 12 秒，P3 在作完行程前總共等待 15 秒，而 P1 在行程作完前總共等待 18 秒。因此平均等待時間是(12+15+18)/3=15 秒。

依序循環排班法Round-Robin Scheduling

行程	分割時間
P1	18
P2	6
P3	6

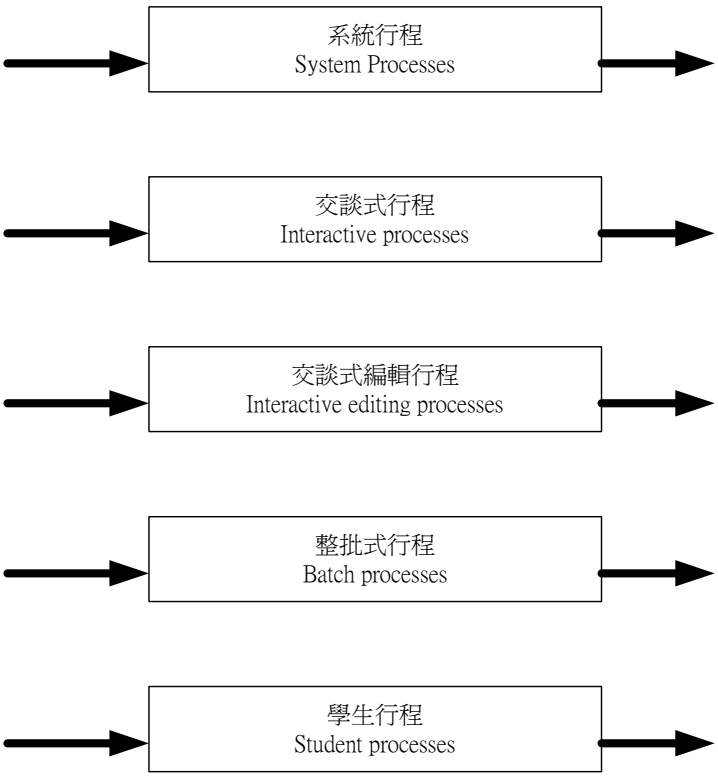


0 3 6 9 12 15 18

這是我們RR的工作排班法，我們假設每次分配CPU給行程3個單位的時間。P2在作完行程前總共等待12秒，P3在作完行程前總共等待15秒，而P1在行程作完前總共等待18秒。因此平均等待時間是(12+15+18)/3=15秒。

多層佇列之排班法則 Multilevel Queue Scheduling，是根據行程的性質將它們分成幾個不同的小組，最典型的分類方式是區分為foreground"前景執行"或background"背景執行"。前景 foreground 代表交談式的行程，而背景 Background 代表的是整批作業的行程。一般來說前景行程的優先權高於背景執行。多層佇列排班法則將就緒佇列區分為五種獨立的佇列，行程一般都是按照行程得性質決定，例如行程的優先權、記憶體的大小、作業系統設計的目地等。在我們的例子中，前景的行程使用 Round-Robin 依序循序排班的排班法則，而背景的行程使用 FCFS 先來先作排班法則。因此，前景行程的優先權大於背景行程。Linux 作業系統使用多層佇列之排班法則。

多層佇列之排班法則
Multilevel Queue Scheduling



18-1 Shell 指令的執行過程

當我們在終端機上下達指令，這時預設的 bash(Bourne shell 為父行程)，就會使用 fork 產生子行程，然後子行程會使用 Exec 去執行我們所下達的指令，當指令完成時，而 Bash 父行程又回到 exec 等待我們下達新的指令。

我們在終端機上，下達 ps。

```
[root@flash chaiyen]# ps
```

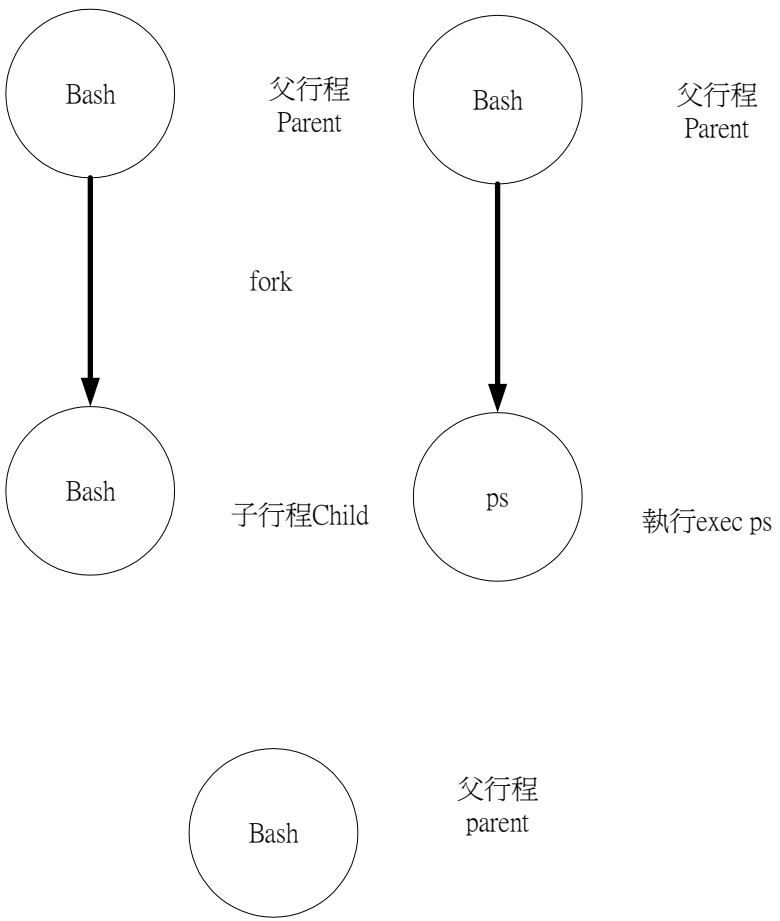
這時 bash 行程 1545 就會使用 fork 產生 1637 編號的子行程，然後執行 ps。

```
[root@flash chaiyen]# ps
  PID TTY          TIME CMD
 1544 pts/0    00:00:00 su
 1545 pts/0    00:00:00 bash
 1637 pts/0    00:00:00 ps
```

執行完 ps 後又回到了指令直譯器。

```
[root@flash chaiyen]#
```

當我們在終端機上下達ps指令，這時預設的Bash(Bourne shell爲父行程)，就會使用fork產生子行程，然後子行程會使用Exec去執行我們所下達的指令ps，當指令完成時，而Bash父行程又回到exec等待我們下達新的指令。



18-2 行程的屬性

我們可以使用 ps 來觀看目前行程的狀態。我們加入一些參數，就可以看到我們所要指定的情況。

指令：ps

參數:

-a：顯示所有在終端機執行的行程資訊，包括其它使用者的行程。

-e 或 -A：顯示所有在系統執行的行程資訊。

-j：顯示工作的資訊(包含 Parent PID、group ID,session ID)。

-l：顯示行程的狀態。

-r：顯示狀態正在執行的行程。

-u：顯示使用者行程的資訊。

-x：顯示行程的所有資訊，而沒有包含 TTYs。

-f：顯示行程間的階層與關係。

我們使用 ps a 顯示所有在終端機執行的行程資訊，包括其它使用者的行程。

```
[root@flash chaiyen]# ps a
  PID TTY      STAT   TIME COMMAND
 1502 pts/0    S      0:00 -bash
 1544 pts/0    S      0:00 su root
 1545 pts/0    S      0:00 bash
 2239 pts/0    R      0:00 ps a
```

行程狀態 state	說明
D	非中斷式 sleep(通常為 I/O 輸入/輸出)
N	低優先權行程(被 Nice 過的行程)
R	被排在執行佇列中，隨時會被執行的行程
S	Sleeping ，正在睡眠中
T	Traced 或 stopped 追蹤或停止
Z	Zombie，已經被停止的行程
W	被 swapped 到硬碟的行程

行程符號	說明
PID	為行程的編號，每一個行程都有它自己唯一的行程編號。
TTY	行程執行時的終端機。
STAT	行程的狀態
TIME	行程已經執行的時間。
CMD	行程被執行的指令名稱

我們使用 ps -j 顯示工作的資訊(包含 Parent PID、group ID,session ID)。

```
[root@flash chaiyen]# ps -j
  PID  PGID   SID TTY          TIME CMD
 1744  1744  1701 pts/0    00:00:00 su
 1746  1746  1701 pts/0    00:00:00 bash
 1852  1852  1701 pts/0    00:00:00 ps
```

我們使用 ps -r 顯示狀態正在執行的行程。

```
[root@flash chaiyen]# ps -r
  PID TTY      STAT   TIME COMMAND
 2242 pts/0    R      0:00 ps -r
```

我們使用 ps -l 顯示行程的狀態。

```
[root@flash chaiyen]# ps -l
 F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME CMD
000 S     0  1744  1701  0  75   0    -   731 wait4  pts/0    00:00:00 su
100 S     0  1746  1744  0  75   0    -   777 wait4  pts/0    00:00:00 bash
100 R     0  1855  1746  0  76   0    -   769 -      pts/0    00:00:00 ps
```

我們使用 ps -u 顯示使用者行程的資訊。

```
[root@flash chaiyen]# ps -u
USER       PID %CPU %MEM    VSZ  RSS TTY      STAT START   TIME COMMAND
root      1744  0.0  0.1   2924 1004 pts/0    S    15:56   0:00 su root
root      1746  0.0  0.2   3108 1340 pts/0    S    15:56   0:00 bash
root      1861  0.0  0.1   2608  692 pts/0    R    16:13   0:00 ps -u
```

我們使用 ps -e 顯示所有在系統執行的行程資訊。

```
[root@flash chaiyen]# ps -elmore
  PID TTY          TIME CMD
    1 ?        00:00:04 init
    2 ?        00:00:00 keventd
    3 ?        00:00:00 kapmd
    4 ?        00:00:00 ksoftirqd_CPU0
    5 ?        00:00:00 kswapd
    6 ?        00:00:00 bdflush
    7 ?        00:00:01 kupdated
    8 ?        00:00:00 mdrecoveryd
   12 ?        00:00:02 kjournald
   91 ?        00:00:00 khubd
  623 ?        00:00:00 eth0
  707 ?        00:00:00 syslogd
  712 ?        00:00:00 klogd
  732 ?        00:00:00 portmap
  760 ?        00:00:00 rpc.statd
  851 ?        00:00:00 apmd
  918 ?        00:00:00 snmpd
  937 ?        00:00:00 named
  939 ?        00:00:00 named
  940 ?        00:00:00 named
```

我們使用 ps -x 顯示行程的所有資訊，而沒有包含 TTYs。More 是分段的意義。

```
[root@flash chaiyen]# ps -x|more
  PID TTY      STAT   TIME COMMAND
    1 ?        S      0:04 init
    2 ?        SW     0:00 [keventd]
    3 ?        SW     0:00 [kapmd]
    4 ?        SWN    0:00 [ksoftirqd_CPU0]
    5 ?        SW     0:00 [kswapd]
    6 ?        SW     0:00 [bdflush]
    7 ?        SW     0:01 [kupdated]
    8 ?        SW     0:00 [mdrecoveryd]
   12 ?        SW     0:03 [kjournald]
   91 ?        SW     0:00 [khubd]
  623 ?        SW     0:00 [eth0]
  707 ?        S      0:00 syslogd -m 0
  712 ?        S      0:00 klogd -x
  851 ?        S      0:00 /usr/sbin/apmd -p 10 -w 5 -W -P /etc/sysconfig/apm-s
  918 ?        S      0:00 /usr/sbin/snmpd -s -l /dev/null -P /var/run/snmpd -a
  962 ?        S      0:00 /usr/sbin/sshd
  995 ?        S      0:00 xinetd -stayalive -reuse -pidfile /var/run/xinetd.pi
 1049 ?        S      0:00 rpc.rquotad
 1054 ?        S      0:00 rpc.mountd
 1060 ?        SW     0:00 [nfsd]
 1061 ?        SW     0:00 [nfsd]
```

我們使用 ps -aux 顯示行程的所有資訊。

```
[root@flash chaiyen]# ps -aux
USER       PID %CPU %MEM    VSZ  RSS TTY      STAT START   TIME COMMAND
root         1  0.1  0.0   1368  476 ?        S    18:57   0:04 init
root         2  0.0  0.0      0    0 ?        SW   18:57   0:00 [keventd]
root         3  0.0  0.0      0    0 ?        SW   18:57   0:00 [kapmd]
root         4  0.0  0.0      0    0 ?        SWN  18:57   0:00 [ksoftirqd_CPU0]
root         5  0.0  0.0      0    0 ?        SW   18:57   0:00 [kswapd]
root         6  0.0  0.0      0    0 ?        SW   18:57   0:00 [bdflush]
root         7  0.0  0.0      0    0 ?        SW   18:57   0:00 [kupdated]
root         8  0.0  0.0      0    0 ?        SW   18:57   0:00 [mdrecoveryd]
root        12  0.0  0.0      0    0 ?        SW   18:57   0:02 [kjournald]
root        91  0.0  0.0      0    0 ?        SW   18:57   0:00 [khubd]
root       622  0.0  0.0      0    0 ?        SW   18:57   0:00 [eth0]
root       706  0.0  0.1   1428  560 ?        S    18:57   0:00 syslogd -m 0
root       711  0.0  0.0   1364  444 ?        S    18:57   0:00 klogd -x
rpc        731  0.0  0.1   1508  544 ?        S    18:57   0:00 portmap
rpcuser    759  0.0  0.1   1556  716 ?        S    18:57   0:00 rpc.statd
root       850  0.0  0.0   1360  480 ?        S    18:57   0:00 /usr/sbin/apmd -
root       917  0.0  0.5   5528 2764 ?        S    18:57   0:00 /usr/sbin/snmpd
named      936  0.0  0.4  10384 2352 ?        S    18:57   0:00 named -u named
named      938  0.0  0.4  10384 2352 ?        S    18:57   0:00 named -u named
named      939  0.0  0.4  10384 2352 ?        S    18:57   0:00 named -u named
named      940  0.0  0.4  10384 2352 ?        S    18:57   0:00 named -u named
```

我們使用 ps –e f 就可以看到行程的階層父子關係。

```
[root@flash chaiyen]# ps -e f|more
  PID TTY      STAT   TIME COMMAND
    1 ?        S      0:04 init
    2 ?        SW     0:00 [keventd]
    3 ?        SW     0:00 [kapmd]
    4 ?        SWN    0:00 [ksoftirqd_CPU0]
    5 ?        SW     0:00 [kswapd]
    6 ?        SW     0:00 [bdflush]
    7 ?        SW     0:03 [kupdated]
    8 ?        SW     0:00 [mdrecoveryd]
   12 ?        SW     0:09 [kjournald]
   91 ?        SW     0:00 [khubd]
  622 ?        SW     0:00 [eth0]
```

我們可以看到行程編號為 2633 的是行程編號為 2634 的父行程。

```
  994 ?        S      0:00 xinetd -stayalive -reuse -pidfile /var/run/xinetd.pid
 2482 ?        S      0:00  \_ fam
 2590 ?        S      0:00  \_ in.telnetd: 61-218-29-5.HINET-IP.hinet.net
 2591 ?        S      0:00      \_ login -- chaiyen
 2592 pts/0    S      0:00          \_ -bash
 2633 pts/0    S      0:00              \_ su root
 2634 pts/0    S      0:00                  \_ bash
 2693 pts/0    R      0:00                      \_ ps -e f
 2694 pts/0    S      0:00                      \_ more
```

欄位	說明
PID	為行程的編號，每一個行程都有它自己唯一的行程編號。
TTY	行程執行時的終端機。
STAT	行程的狀態
TIME	行程已經執行的時間。
CMD	行程被執行的指令名稱
USER	行程的執行用者
%CPU	所用 CPU 與所花費的時間的比率
%MEM	記憶體的使用率
VSZ	VIRTUAL SIZE，行程在記憶體映像中的大小。
RSS	行程在實體記憶體中所佔的大小，單位 KBYTES
START	開使執行行程的時間
F	旗標。指出行程是屬於使用者 USER 或核心 Kernel。
UID	行程執行者的使用者 ID
PRI	行程被排班的優先權
PPID	父行程的行程 ID
NI	Nice 的值，Nice 為降低優先權
WCHAN	等待頻道，當為 Null 空時，表示行程正在執行，當行程在就緒時為 Waiting for

當我們要殺掉指定的程式或行程時，我們可以使用 Kill 指令將該行程的編號殺掉及可。我們使用 kill 2643 將編號為 2643 的行程給終止。

語法：

指令：kill

```
[root@flash chaiyen]# kill 2634
```

如果我們想顯示 CPU 即時的資訊，我們可以使用 top 指令。

每格數秒就會更新最新的行程資訊。我們按下 q 就可以離開 top 了。

```
 11:26pm  up  4:29,  3 users,  load average: 0.00, 0.00, 0.00
106 processes: 103 sleeping, 3 running, 0 zombie, 0 stopped
CPU states:  0.0% user,  1.1% system,  0.0% nice, 98.8% idle
Mem:   514300K av,  176012K used,  338288K free,       0K shrd,   39812K buff
Swap:  819304K av,       0K used,  819304K free                   62060K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 2916 root      15   0  1100 1100   856 R     0.3  0.2   0:00 top
 2457 root      15   0  3100 3100  2564 S     0.1  0.6   0:01 magicdev
 2461 root      15   0  5508 5508  4080 S     0.1  1.0   0:02 panel
    1 root      15   0   476  476   420 S     0.0  0.0   0:04 init
    2 root      15   0     0    0     0 SW    0.0  0.0   0:00 keventd
    3 root      15   0     0    0     0 SW    0.0  0.0   0:00 kapmd
    4 root      34  19     0    0     0 SWN   0.0  0.0   0:00 ksoftirqd_CPU0
    5 root      15   0     0    0     0 SW    0.0  0.0   0:00 kswapd
    6 root      25   0     0    0     0 SW    0.0  0.0   0:00 bdflush
    7 root      15   0     0    0     0 SW    0.0  0.0   0:03 kupdated
    8 root      16   0     0    0     0 SW    0.0  0.0   0:00 mdrecoveryd
   12 root      15   0     0    0     0 SW    0.0  0.0   0:11 kjournald
   91 root      16   0     0    0     0 SW    0.0  0.0   0:00 khubd
  622 root      15   0     0    0     0 SW    0.0  0.0   0:00 eth0
  706 root      15   0   560  560   472 S     0.0  0.1   0:00 syslogd
  711 root      15   0   444  444   384 S     0.0  0.0   0:00 klogd
```

當我們在顯示 CPU 使用情形時，按下 h 就可以看到 top 的功能鍵功能。

```
fF       add and remove fields
oO       Change order of displayed fields
h or ?   Print this list
S        Toggle cumulative mode
i        Toggle display of idle proceses
I        Toggle between Irix and Solaris views (SMP-only)
c        Toggle display of command name/line
l        Toggle display of load average
m        Toggle display of memory information
t        Toggle display of summary information
k        Kill a task (with any signal)
r        Renice a task
N        Sort by pid (Numerically)
A        Sort by age
P        Sort by CPU usage
M        Sort by resident memory usage
T        Sort by time / cumulative time
u        Show only a specific user
n or #   Set the number of process to show
s        Set the delay in seconds between updates
W        Write configuration file ~/.toprc
q        Quit
```

當我們按下 u，再輸入使用者，則可以顯示該使用者的情況。

```
 11:34pm  up  4:37,  3 users,  load average: 0.00, 0.00, 0.00
106 processes: 105 sleeping, 1 running, 0 zombie, 0 stopped
CPU states:  0.0% user,  0.6% system,  0.0% nice, 99.3% idle
Mem:   514300K av,  176508K used,  337792K free,       0K shrd,   40272K buff
Swap:  819304K av,       0K used,  819304K free                   62064K cached
$<5>$<3>$<2>$<2>Which User (Blank for All): $<2>
  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 2944 root      16   0  1100 1100   856 R     0.4  0.2   0:00 top
 2457 root      15   0  3100 3100  2564 S     0.2  0.6   0:02 magicdev
    1 root      15   0   476  476   420 S     0.0  0.0   0:04 init
    2 root      15   0     0    0     0 SW    0.0  0.0   0:00 keventd
    3 root      15   0     0    0     0 SW    0.0  0.0   0:00 kapmd
    4 root      34  19     0    0     0 SWN   0.0  0.0   0:00 ksoftirqd_CPU0
    5 root      15   0     0    0     0 SW    0.0  0.0   0:00 kswapd
    6 root      25   0     0    0     0 SW    0.0  0.0   0:00 bdflush
    7 root      15   0     0    0     0 SW    0.0  0.0   0:03 kupdated
    8 root      16   0     0    0     0 SW    0.0  0.0   0:00 mdrecoveryd
   12 root      15   0     0    0     0 SW    0.0  0.0   0:11 kjournald
   91 root      16   0     0    0     0 SW    0.0  0.0   0:00 khubd
  622 root      15   0     0    0     0 SW    0.0  0.0   0:00 eth0
  706 root      15   0   560  560   472 S     0.0  0.1   0:00 syslogd
  711 root      15   0   444  444   384 S     0.0  0.0   0:00 klogd
  731 rpc       15   0   552  552   468 S     0.0  0.1   0:00 portmap
```

當我們按下 P 時，就可以依照 CPU 的使用率來排列。

```
 11:38pm  up  4:41,  3 users,  load average: 0.00, 0.00, 0.00
106 processes: 103 sleeping, 3 running, 0 zombie, 0 stopped
CPU states:  0.1% user,  0.1% system,  0.0% nice, 99.6% idle
Mem:   514300K av,  176812K used,  337488K free,       0K shrd,   40564K buff
Swap:  819304K av,       0K used,  819304K free                   62064K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 2960 root      16   0  1100 1100   856 R     0.3  0.2   0:00 top
    1 root      15   0   476  476   420 S     0.0  0.0   0:04 init
    2 root      15   0     0    0     0 SW    0.0  0.0   0:00 keventd
    3 root      15   0     0    0     0 SW    0.0  0.0   0:00 kapmd
    4 root      34  19     0    0     0 SWN   0.0  0.0   0:00 ksoftirqd_CPU0
    5 root      15   0     0    0     0 SW    0.0  0.0   0:00 kswapd
    6 root      25   0     0    0     0 SW    0.0  0.0   0:00 bdflush
    7 root      15   0     0    0     0 SW    0.0  0.0   0:03 kupdated
    8 root      16   0     0    0     0 SW    0.0  0.0   0:00 mdrecoveryd
   12 root      15   0     0    0     0 SW    0.0  0.0   0:11 kjournald
   91 root      16   0     0    0     0 SW    0.0  0.0   0:00 khubd
  622 root      15   0     0    0     0 SW    0.0  0.0   0:00 eth0
  706 root      15   0   560  560   472 S     0.0  0.1   0:00 syslogd
  711 root      15   0   444  444   384 S     0.0  0.0   0:00 klogd
  731 rpc       15   0   552  552   468 S     0.0  0.1   0:00 portmap
  759 rpcuser   18   0   716  716   624 S     0.0  0.1   0:00 rpc.statd
```

18-3 行程和工作控制

我們可以在 Linux 上作行程和工作的管理，這包含了新建立(new)行程、行程的終止(zombie)、和正在背景或前景執行的行程、暫停行程、或將行程將背景執行或前景執行作轉換。

18-3-1 前景和背景行程

當我們在終端機上打上指令，然後按下<Enter>，shell 就會執行我們的指令，而且回傳和顯示 shell 提示。當我們的指令在執行時，我們無法再下達新的指令給 shell，直到我們先前的工作已經作完時，而且 shell 回傳時，我們才可以再度下達。當我們的指令以這種方式執行，我們稱為在『**前景**』foreground 執行。當前景在執行時，它會保留鍵盤和顯示器的控制權。

當有些時後，我們需要在 Linux 上執行一些需要很久時間才能完成的工作，而當這些工作執行時，我們又希望作一些其它的工作。這時我們就需要將指令下到『**背景**』background 去。當我們下達在『背景』執行的命令時，只要在指令後面加上&(and 的符號)。

背景行程在高 nice 的情況下，也就是低優先權。因此當其它高優先權的行程執行完，才會輪到它使用 CPU。

前景執行

指令：

command

背景執行

指令：

command &

當我們執行 find 的指令來尋找和 mail 有關的檔案，再將檔案輸入到 foo.txt 的檔案中，這是會花費很多的時間。而我們要等待它執行完才能執行下一個動作或指令。

```
[root@flash chaiyen]# find / -name mail -print > foo.txt
```

這是我們編輯 foo.txt 的情況。

```
[root@flash chaiyen]# vi foo.txt
```

這是我們使用 find 指令找到有關 mail 檔案的情況並將它輸入到 foo.txt。

```
/var/mail
/var/spool/mail
/var/mailman/mail
/etc/mail
/root/mail
/usr/lib/gimp/1.2/plug-ins/mail
/usr/share/doc/qt-devel-3.0.3/examples/network/mail
/usr/share/doc/qt2-devel-2.3.1/examples/mail
/usr/share/emacs/21.2/lisp/mail
/usr/share/rep/0.15.1/lisp/rep/mail
/usr/share/evolution/views/mail
/usr/share/epic/help/4_Misc/set/mail
/usr/share/epic/help/5_Programming/on/mail
/bin/mail
```

當我們執行 find 的指令來尋找和 mail 有關的檔案並且讓它在背景執行，再將檔案輸入到 foo.txt 的檔案中，這是會花費很多的時間，但它是在背景執行。因此我們不需等待它執行完才能下下一個動作或指令。

我們在背景執行 find 指令，我們輸入 jobs 就可以看到正在執行的工作了。而這就是在背景執行的 find。

```
[root@flash chaiyen]# find / -name mail > -print > foo.txt &
[1] 1692
[root@flash chaiyen]# jobs
[1]+  Running                 find / -name mail >-print >foo.txt &
```

有一些工作都需要花費很多的時間來執行，因此放到背景是很好的。例如：sort 排列指令、編譯 cc 或 make 指令、find 尋找檔案指令。我們使用 fg 指令就可以將背景的行程帶到前景來執行。

指令：

fg [%jobid]

參數：

%或%+：目前的工作。

%-：前一個工作。

%N：工作編號 N。

%Name：工作開始的名稱。

%?Name：指令包含名稱。

我們使用 vi 來編輯 foo.txt 檔。

```
[root@flash chaiyen]# vi foo.txt
```

我們使用<Ctrl-z>來暫停使用 vi 編輯 foo.txt 檔。

我們可以使用 jobs 來觀看目前的工作情況，可以看到我們的 vim foo.txt 的狀態是暫停 Stopped。

```
[root@flash chaiyen]# jobs
[1]+  Stopped                 vim foo.txt
```

我們現在要再次的編輯它，只要使用 fg %1，就可以將第一個工作給移到前景，這樣我們就可以開始編輯了。

```
[root@flash chaiyen]# fg %1
```

我們也可以將被暫停的工作丟到背景去，我們可以使用下列的指令 bg。當在編譯 compile 檔案時，我們就可以使用 bg，將它丟到背景去執行。

指令：

bg [%jobid]

參數：

%或%+：目前的工作。

%-：前一個工作。

%N：工作編號 N。

%Name：工作開始的名稱。

%?Name：指令包含名稱。

當我們要了解有哪一些工作時，可以使用指令 jobs。

指令：

jobs

參數：

-l：顯示該工作的行程編號。

我們使用 jobs –l 就可以顯示所有的工作，和該工作的行程編號。

```
[root@flash chaiyen]# jobs -l
[1]+  2068 停止 (tty 輸出)          vim foo.txt
```

vim 的行程編號 PID 為 2068。

```
[root@flash chaiyen]# ps
  PID TTY          TIME CMD
 1658 pts/0    00:00:00 su
 1659 pts/0    00:00:00 bash
 2068 pts/0    00:00:00 vim
 2158 pts/0    00:00:00 ps
```

18-3-2 Linux 背景行程 Daemons

任何在背景執行的程式都可以叫作 Daemon。Daemons 背景行程提供各式各樣的服務給我們的使用者，和系統管理的工作，例如：列印、e-mail 都是經過背景行程的服務。列印服務是由列印背景行程 Daemon 所提供，而 e-mail 服務是由 smtpd daemon 所提供，而網路瀏覽服務是由 httpd 背景行程所提供。

18-3-3 指令平行和循序的執行

我們可以在一行命令區輸入多個指令當作是平行或是循序的執行。

這是我們在一行命令區輸入多個指令當作是循序的執行。

語法：

指令 1；指令 2；指令 3；….；指令 N

我們使用分號當作是循序指令的分別。我們第一個是輸入 date 指令，第二個是輸入 echo 指令，我們第三個是輸入 who 指令，它們會循序的先從 date 執行，執行完再執行 echo 指令，最後才會執行 who 指令。

```
[root@flash etc]# date;echo "大家歡喜快樂喔";who
週二  8月  6 17:40:09 CST 2002
大家歡喜快樂喔
chaiyen  pts/0    Aug  6 15:10 (61-218-29-5.HINET-IP.hinet.net)
```

這是我們在一行命令區輸入多個指令當作是平行的執行(parallel execution)。

語法：

指令 1&指令 2&指令 3&….&指令 N&

我們使用&當作是平行指令的執行。我們第一個是輸入 date 指令，第二個是輸入 echo 指令，我們第三個是輸入 who 指令。它們會平行的執行。我們 date 的行程編號是 2303，然後我們 echo 的行程編號是 2304。

```
[root@flash etc]# date&echo "大家歡喜快樂喔"&who
[2] 2303
[3] 2304
週二  8月  6 17:50:18 CST 2002
大家歡喜快樂喔
chaiyen  pts/0    Aug  6 15:10 (61-218-29-5.HINET-IP.hinet.net)
[2]   Done                    date
[3]-  Done                    echo "大家歡喜快樂喔"
```

我們在最後也加入&符號，表示 date、echo、who 是平行的執行，因此 date 的行程編號為 2312、ehco 的行程編號為 2313、who 的行程編號為 2314。

```
[root@flash etc]# date&echo "大家歡喜快樂喔"&who&
[2] 2312
[3] 2313
[4] 2314
[root@flash etc]# 週二  8月  6 17:53:48 CST 2002
大家歡喜快樂喔
chaiyen  pts/0    Aug  6 15:10 (61-218-29-5.HINET-IP.hinet.net)

[2]   Done                    date
[3]   Done                    echo "大家歡喜快樂喔"
[4]-  Done                    who
```

Linux 提供我們在 shell 中執行它的子行程 child process，而我們將一群命令使用括號將它們包起來，這就稱為群組指令 command grouping，因為這些指令都是由 shell 的子行程逐一循序執行。

我們使用括號將指令給括號起來，當群組指令。

```
[root@flash chaiyen]# (date;echo "大家好")
週二  8月  6 20:12:05 CST 2002
大家好
```

我們可以輸入(date;echo “大家好”)&在背景執行。

```
[root@flash chaiyen]# (date;echo "大家好")&
[1] 1796
[root@flash chaiyen]# 週二  8月  6 20:12:34 CST 2002
大家好
```

<Enter>

```
[1]+  Done                    ( date; echo "大家好" )
```

18-3-4 指令和行程不正常的中斷

當我們執行指令時，它會執行到結束，最後才中斷。當我們想中斷正在執行的指令，我們可以使用<Ctrl+c>來將前景目前的行程來中斷。我們常用 kill 指令作軟體中斷，並將中斷的信號 siganl 送給行程。Signal 信號可分為內部信號 internal signal 和外部信號 external signal。內部信號 internal siganl 是由行程內部自己發出，而外部信號 external signal 則是像<Ctrl+c>是由外部所產生。

18-4 在 Linux 上的行程階層(Process Hierarchy in Linux)

當我們開啟 Linux 作業系統，LILO(Linux Loader)就會從硬碟載入 Linux 的核心到記憶體。它初始化我們的硬體元件，像是硬碟控制器(Disk Controller)，然後 Linux 進入保護模式(protected mode)載入作業系統，然後初始化各種核心資料結構，像 I-node 和檔案表(file tables)。這個行程的 PID 為 0。它開啟這 init 行程(此行程 PID 為 1)，而 init 行程則執行其它行程的啟動 starts。這個 init 行程啟動 daemons 背景行程 kflushd、kupdated、kpiod、kswapd….(這是在 Mandrake 上的行程，在 Redhat 上則可由底下的階層可知，但皆大同小異，只是名稱的不同，內容其實還是相同的)。Init 行程然後會初始化檔案系統，然後掛載到根 root 目錄上，然後它會執行/sbin/init 的程式，然後在每個終端機上執行 getty 行程。Getty 行程會設定終端機的屬性，然後它會顯示 login 的畫面，讓我們登錄系統。當我們登錄系統時，getty 行程就會 fork 子行程，然後它就會執行 login 的行程這個行程會找尋在/etc/passwd 檔案上的名稱和密碼是否相符。

我們使用 pstree 看看行程間的階層關係。

```
[root@flash chaiyen]# pstree
init-+-apmd
     |-atd
     |-bdflush
     |-crond
     |-eth0
     |-gdm---gdm-+-X
     |           `-gdmlogin---xsri
     |-gpm
     |-httpd---8*[httpd]
     |-kapmd
     |-keventd
     |-khubd
     |-kjournald
     |-klogd
     |-ksoftirqd_CPU0
     |-kswapd
     |-kupdated
     |-lockd---rpciod
     |-lpd
     |-mdrecoveryd
     |-6*[mingetty]
     |-named---named---3*[named]
     |-netdump-server
```

```
 |-8*[nfsd]
 |-nmbd
 |-portmap
 |-rpc.mountd
 |-rpc.rquotad
 |-rpc.statd
 |-safe_mysqld---mysqld---mysqld---2*[mysqld]
 |-sendmail
 |-smbd
 |-snmpd
 |-squid---squid---unlinkd
 |-sshd
 |-syslogd
 |-tserver
 |-xfs
 `-xinetd---in.telnetd---login---bash---su---bash---pstree
```

我們使用 ps –aux|more 顯示行程的 PID 順序。

```
  PID %CPU %MEM    VSZ  RSS TTY      STAT START   TIME COMMAND
    1  0.0  0.0   1368  476 ?        S    08:04   0:04 init
    2  0.0  0.0      0    0 ?        SW   08:04   0:00 [keventd]
    3  0.0  0.0      0    0 ?        SW   08:04   0:00 [kapmd]
    4  0.0  0.0      0    0 ?        SWN  08:04   0:00 [ksoftirqd_CPU0]
    5  0.0  0.0      0    0 ?        SW   08:04   0:00 [kswapd]
    6  0.0  0.0      0    0 ?        SW   08:04   0:00 [bdflush]
    7  0.0  0.0      0    0 ?        SW   08:04   0:04 [kupdated]
    8  0.0  0.0      0    0 ?        SW   08:04   0:00 [mdrecoveryd]
   12  0.2  0.0      0    0 ?        SW   08:04   0:24 [kjournald]
   91  0.0  0.0      0    0 ?        SW   08:04   0:00 [khubd]
  622  0.0  0.0      0    0 ?        SW   08:04   0:00 [eth0]
  706  0.0  0.1   1428  560 ?        S    08:04   0:00 syslogd -m 0
  711  0.0  0.0   1364  444 ?        S    08:04   0:00 klogd -x
  731  0.0  0.1   1508  544 ?        S    08:04   0:00 portmap
  759  0.0  0.1   1556  716 ?        S    08:04   0:00 rpc.statd
  850  0.0  0.0   1360  480 ?        S    08:04   0:00 /usr/sbin/apmd -
  917  0.0  0.5   5528 2764 ?        S    08:04   0:00 /usr/sbin/snmpd
  936  0.0  0.4  10384 2360 ?        S    08:04   0:00 named -u named
  938  0.0  0.4  10384 2360 ?        S    08:04   0:00 named -u named
  939  0.0  0.4  10384 2360 ?        S    08:04   0:00 named -u named
  940  0.0  0.4  10384 2360 ?        S    08:04   0:00 named -u named
```

我們使用 ps –e f|more 指令，觀看整個 Linux 行程的階層。行程編號 994 為啟動 xinetd 網路服務。

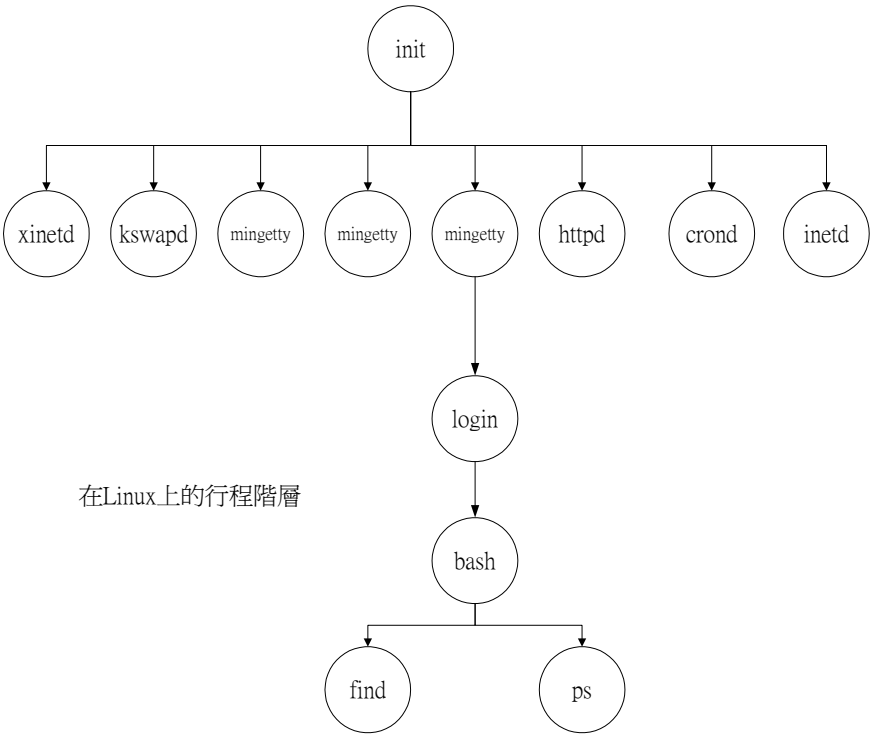
```
[root@flash sbin]# ps -e f|more
  PID TTY      STAT   TIME COMMAND
    1 ?        S      0:04 init
    2 ?        SW     0:00 [keventd]
    3 ?        SW     0:00 [kapmd]
    4 ?        SWN    0:00 [ksoftirqd_CPU0]
    5 ?        SW     0:00 [kswapd]
    6 ?        SW     0:00 [bdflush]
    7 ?        SW     0:04 [kupdated]
    8 ?        SW     0:00 [mdrecoveryd]
   12 ?        SW     0:25 [kjournald]
   91 ?        SW     0:00 [khubd]
  622 ?        SW     0:00 [eth0]
  706 ?        S      0:00 syslogd -m 0
  711 ?        S      0:00 klogd -x
  731 ?        S      0:00 portmap
  759 ?        S      0:00 rpc.statd
  850 ?        S      0:00 /usr/sbin/apmd -p 10 -w 5 -W -P /etc/sysconfig/apm-sc
  917 ?        S      0:00 /usr/sbin/snmpd -s -l /dev/null -P /var/run/snmpd -a
  936 ?        S      0:00 named -u named
  938 ?        S      0:00  \_ named -u named
  939 ?        S      0:00      \_ named -u named
  940 ?        S      0:00      \_ named -u named
  941 ?        S      0:00      \_ named -u named
```

```
 941 ?        S      0:00      \_ named -u named
 961 ?        S      0:00 /usr/sbin/sshd
 994 ?        S      0:00 xinetd -stayalive -reuse -pidfile /var/run/xinetd.pid
2366 ?        S      0:00  \_ in.telnetd: 61-218-29-5.HINET-IP.hinet.net
2367 ?        S      0:00      \_ login -- chaiyen
2368 pts/0    S      0:00          \_ -bash
2411 pts/0    S      0:00              \_ su root
2412 pts/0    S      0:00                  \_ bash
2710 pts/0    R      0:00                      \_ ps -e f
2711 pts/0    R      0:00                      \_ bash
1018 ?        S      0:00 lpd Waiting
1048 ?        S      0:00 rpc.rquotad
1053 ?        S      0:00 rpc.mountd
1059 ?        SW     0:00 [nfsd]
1060 ?        SW     0:00 [nfsd]
1061 ?        SW     0:00 [nfsd]
1062 ?        SW     0:00 [nfsd]
1063 ?        SW     0:00 [nfsd]
1064 ?        SW     0:00 [nfsd]
1065 ?        SW     0:00 [nfsd]
1066 ?        SW     0:00 [nfsd]
1069 ?        SW     0:00 [lockd]
1070 ?        SW     0:00  \_ [rpciod]
```

```
1088 ?        S      0:00 /bin/sh /usr/bin/safe_mysqld --defaults-file=/etc/my.
1130 ?        S      0:00  \_ /usr/libexec/mysqld --defaults-file=/etc/my.cnf -
1162 ?        S      0:00      \_ /usr/libexec/mysqld --defaults-file=/etc/my.c
1163 ?        S      0:00          \_ /usr/libexec/mysqld --defaults-file=/etc/
1180 ?        S      0:00          \_ /usr/libexec/mysqld --defaults-file=/etc/
1164 ?        S      0:00 /usr/sbin/netdump-server --daemon
1193 ?        S      0:00 sendmail: accepting connections
1212 ?        S      0:00 gpm -t ps/2 -m /dev/mouse
1235 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
1254 ?        S      0:00  \_ /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE
1255 ?        S      0:00  \_ /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE
1256 ?        S      0:00  \_ /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE
1257 ?        S      0:00  \_ /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE
1258 ?        S      0:00  \_ /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE
1259 ?        S      0:00  \_ /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE
1260 ?        S      0:00  \_ /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE
1261 ?        S      0:00  \_ /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE
1253 ?        S      0:00 crond
1301 ?        S      0:00 /usr/bin/tserver
1302 ?        S      0:00 squid -D
1304 ?        S      0:03  \_ (squid) -D
1305 ?        S      0:00      \_ (unlinkd)
1381 ?        S      0:00 xfs -droppriv -daemon
```

```
1381 ?        S      0:00 xfs -droppriv -daemon
1399 ?        S      0:00 smbd -D
1404 ?        S      0:00 nmbd -D
1440 ?        S      0:00 /usr/sbin/atd
1466 tty1     S      0:00 /sbin/mingetty tty1
1467 tty2     S      0:00 /sbin/mingetty tty2
1468 tty3     S      0:00 /sbin/mingetty tty3
1469 tty4     S      0:00 /sbin/mingetty tty4
1470 tty5     S      0:00 /sbin/mingetty tty5
1471 tty6     S      0:00 /sbin/mingetty tty6
1472 ?        S      0:00 /usr/bin/gdm -nodaemon
1480 ?        S      0:00  \_ /usr/bin/gdm -nodaemon
1481 ?        S      0:01      \_ /usr/bin/X11/X :0 -auth /var/gdm/:0.Xauth
1487 ?        S      0:00      \_ /usr/bin/gdmlogin --disable-sound --disable-c
1488 ?        S      0:00          \_ /usr/bin/xsri --redhat-login --run
```

在 Linux 上的行程階層，我們可以看到最上層的為 init 行程，而這個行程顯示我們有個使用者正在使用 Bash shell 當作是 login shell 來執行 find 指令和 ps 指令。因此當我們登錄 Linux 時，系統會為我們建立 login 行程，然後建立 login shell，然後這個 shell 執行或直譯我們的指令。



在Linux上的行程階層

18-5 開機程序

開機時第一個動作就是由 LILO 將作業系統的核心載入。

我們可以使用 locate lilo.conf 來找尋 lilo 的設定檔，它一般是放在/etc 的目錄下。

```
[root@flash home]# locate lilo.conf
/etc/lilo.conf.anaconda
/usr/share/man/man5/lilo.conf.5.gz
```

我們使用 vi 來編輯 lilo 的設定檔。

```
[root@flash home]# vi /etc/lilo.conf.anaconda
```

第一行的 prompt 為設定對話模式。

第二行的 timeout 是設定等候使用者的時間，單位是 0.1 秒，因此 50 代表等候 5 秒。第四行的 boot 為指定 lilo 的啟動磁區在哪一個分割區。第十行的 image 為放置 Linux 核心的地方，我們預設是放在/boot 目錄下。

```
 1 prompt
 2 timeout=50
 3 default=linux
 4 boot=/dev/hda
 5 map=/boot/map
 6 install=/boot/boot.b
 7 message=/boot/message
 8 lba32
 9
10 image=/boot/vmlinuz-2.4.18-3
11         label=linux
12         initrd=/boot/initrd-2.4.18-3.img
13         read-only
14         root=/dev/hda1
```

當載入核心後，系統會進入執行/sbin/init 執行作業系統上的各個行程，然後再設定作業系統的環境變數(系統設定檔/etc/rc.d/rc.sysinit)，最後執行開機模式(開機模式在/etc/inittab)。

這是我們在/etc/inittab 所設定開機的模式。

Run level	說明
0	關機
1	單人使用模式
2	多人使用模式(無支援 NFS)
3	完整多人使用模式
4	未使用模式
5	X 視窗模式
6	重新開機模式

我們使用 vi /etc/inittab 編輯設定開機模式檔，我們第 18 行 id:5 設定為 X11 開機(開機時顯示 X 視窗)

```
[root@flash chaiyen]# vi /etc/inittab
```

```
 1 #
 2 # inittab       This file describes how the INIT process should set up
 3 #               the system in a certain run-level.
 4 #
 5 # Author:       Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
 6 #               Modified for RHS Linux by Marc Ewing and Donnie Barnes
 7 #
 8
 9 # Default runlevel. The runlevels used by RHS are:
10 #   0 - halt (Do NOT set initdefault to this)
11 #   1 - Single user mode
12 #   2 - Multiuser, without NFS (The same as 3, if you do not have networ
   king)
13 #   3 - Full multiuser mode
14 #   4 - unused
15 #   5 - X11
16 #   6 - reboot (Do NOT set initdefault to this)
17 #
18 id:5:initdefault:
19
20 # System initialization.
21 si::sysinit:/etc/rc.d/rc.sysinit
```

我們使用 shutdown 或 halt 指令就可以關機。這是指定在早上 8:00 關機。

```
[root@flash rc.d]# shutdown -h 8:00
```

這是指定電腦在三分鐘後關機。

```
[root@flash rc.d]# shutdown -h +3
```

這是指定電腦立刻關機。

```
[root@flash rc.d]# halt
```

這是指定電腦重新開機。

```
[root@flash rc.d]# reboot
```

將電腦關機。語法：

指令：shutdown

將電腦關機。語法：

指令：halt

將電腦重新開機。語法：

指令：reboot