
Linux Network Kernel Inter-process Communication Module

國立臺北科技大學資訊工程系

郭忠義教授

jykuo@ntut.edu.tw

Outline

- ❑ Linux Concept
- ❑ Linux sudoer
- ❑ Linux File System
- ❑ Linux Inter-Process

Linux Concept

計算機系統

□ 硬體

- 提供基本的計算資源CPU，記憶體，I/O設備

□ 作業系統

- 控制和協調各種應用程式和使用者間對硬體的使用

□ 應用程式

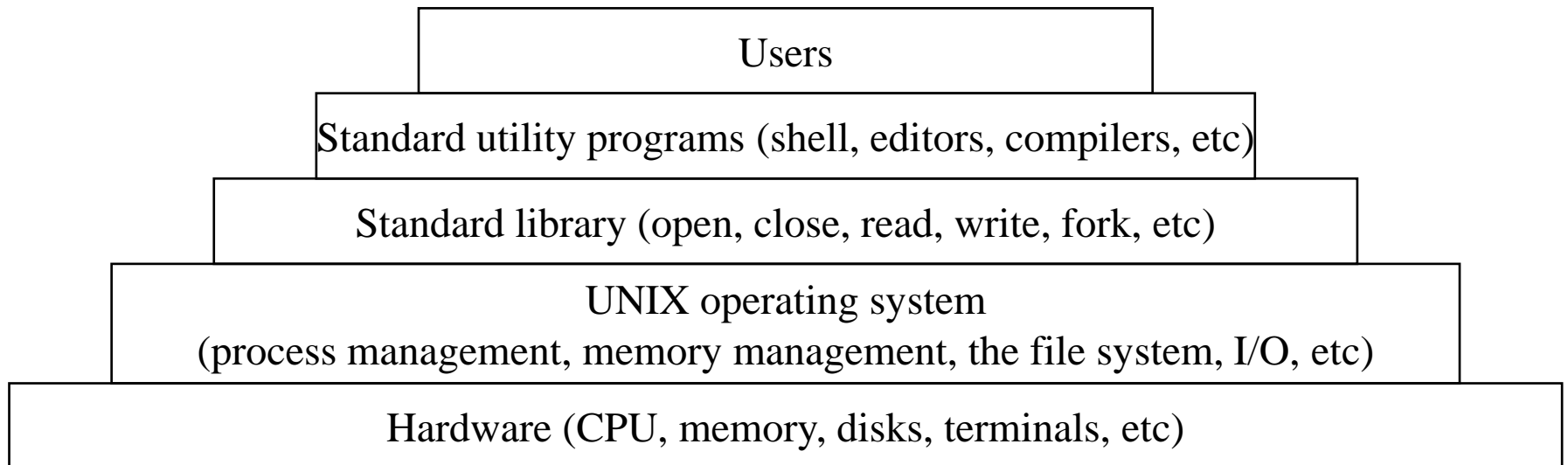
- 定義系統資源用於解決使用者計算問題的方式
- 文字處理器，編譯器，Web瀏覽器，資料庫系統，遊戲

□ 使用者

- 人，機器，其他計算機

計算機系統階層架構

Banking system	Airline reservation	Adventure games	}	Application programs
Compilers	Command interpreter	Editors		
Operating system			}	System programs
Machine language				
Microprogramming			}	Hardware
Physical devices/Computer hardware				



作業系統目的

- 使用者想要方便，易用和良好的性能
 - 不會管理到計算資源的利用
- 大型電腦
 - 需共享計算資源，讓所有使用者滿意
- 行動裝置
 - 計算機資源較少，針對可用性和電池壽命進行優化
- 嵌入式系統使用者介面簡單
 - 例如設備和汽車的行車電腦

作業系統定義

❑ 資源分配器

- 管理所有資源
- 在有效與公平使用資源之衝突請求間做出決定

❑ 控制程式

- 控制執行的程式，防止錯誤和不適當使用計算機

❑ Kernel，一直長駐計算機上執行的程式。

- 其他都是系統程式（作業系統附帶），或應用程式。

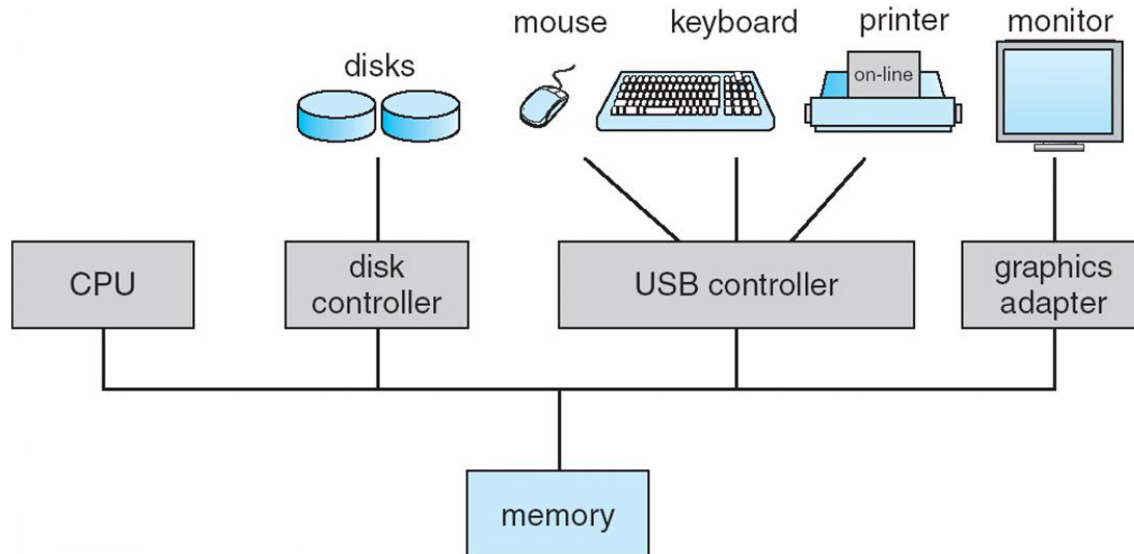
❑ Bootstrap

- 電腦通電或重新啟動時載入
- 通常儲存在ROM或EPROM中，稱為韌體
- 初始化系統
- 載入作業系統kernel並開始執行

作業系統組織

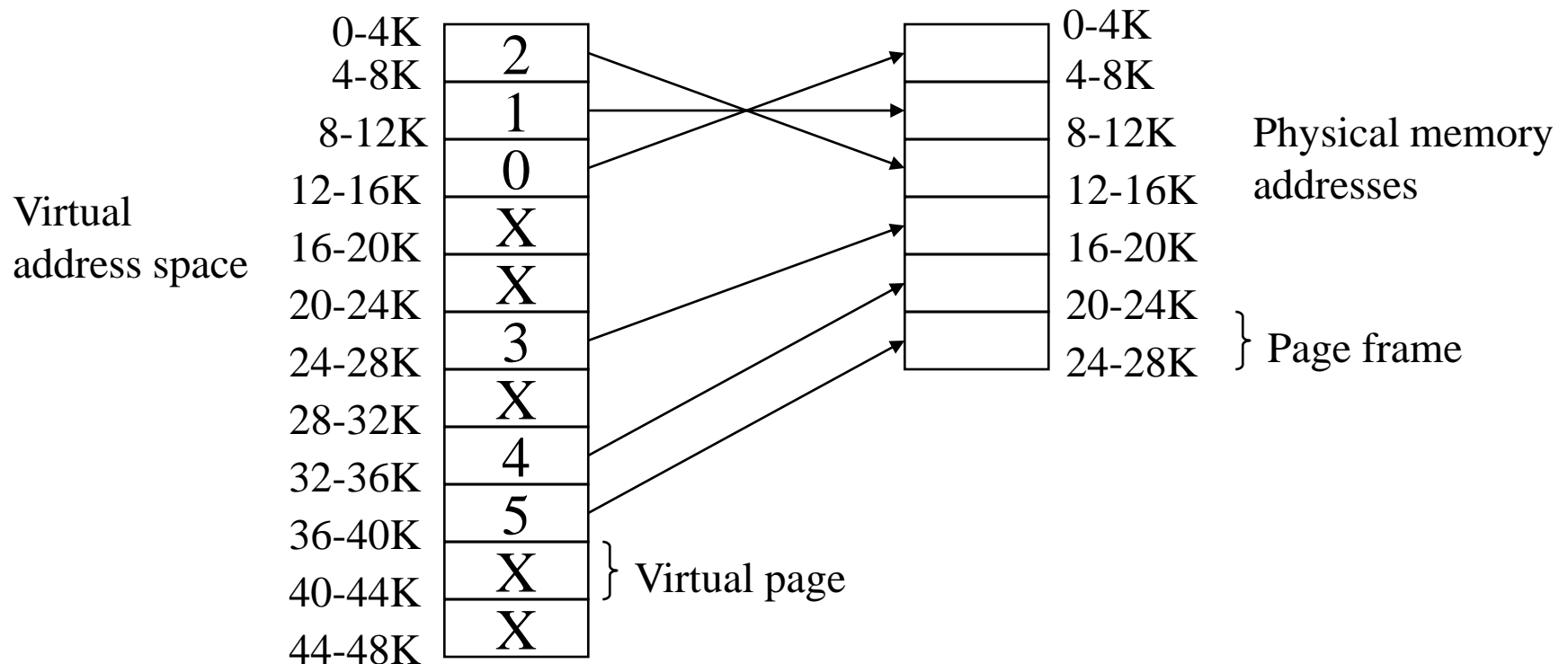
□ 作業系統

- 一或多個CPU，裝置控制器透過Bus連結存取記憶體
- CPU和I/O裝置同時執行，互相競爭存取記憶體
 - 每個控制器負責特別的裝置，各有其本地緩衝區(Buffer)
 - I/O是從裝置到控制器的本地緩衝區
 - 透過Interrupt，裝置控制器通知CPU完成其操作



作業系統概念

❑ 虛擬記憶體與Page table

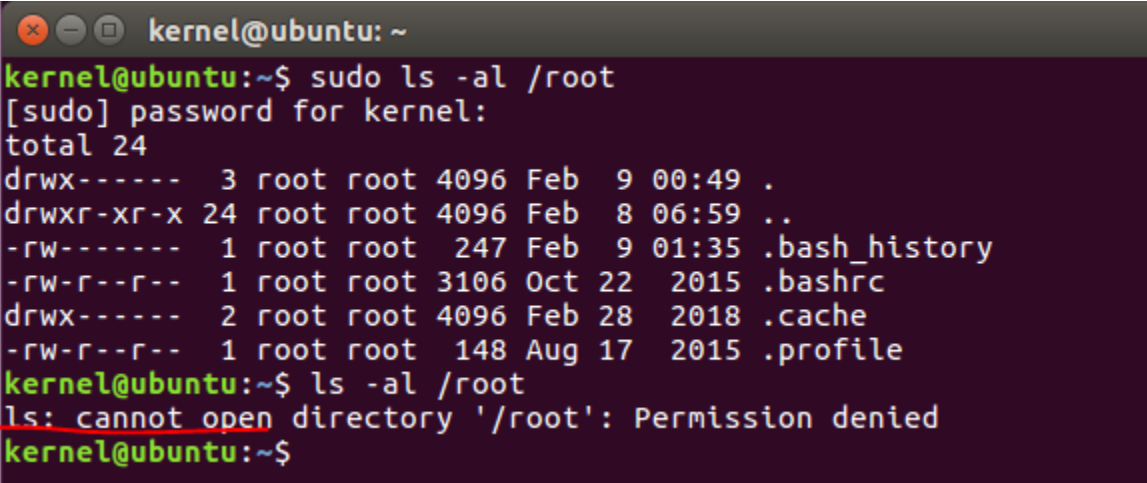


Linux 超級使用者

sudoer

超級使用者-sudo

- ❑ Ubuntu Linux 系統預設安裝時基於安全考量，不啟用 root 帳號，無法用 root 直接登入系統，所有系統管理都透過 sudo 取得 root 權限。
- ❑ sudo 指令
 - sudo 執行完指定指令後會自動離開 root 權限
 - 以 root 權限執行 ls -al，查看一般使用者無法讀取的目錄內容。



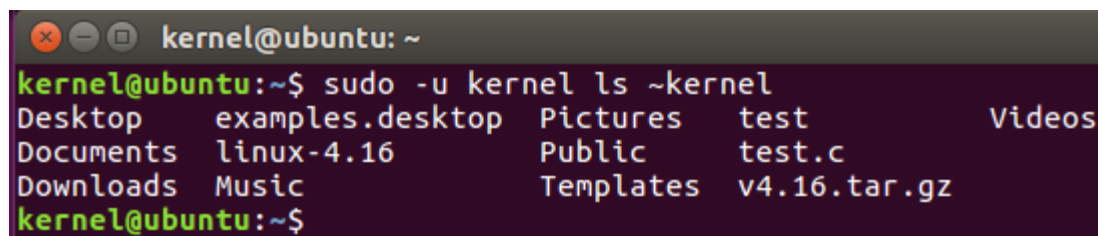
```
kernel@ubuntu: ~  
kernel@ubuntu:~$ sudo ls -al /root  
[sudo] password for kernel:  
total 24  
drwx----- 3 root root 4096 Feb  9 00:49 .  
drwxr-xr-x 24 root root 4096 Feb  8 06:59 ..  
-rw----- 1 root root  247 Feb  9 01:35 .bash_history  
-rw-r--r-- 1 root root 3106 Oct 22  2015 .bashrc  
drwx----- 2 root root 4096 Feb 28  2018 .cache  
-rw-r--r-- 1 root root  148 Aug 17  2015 .profile  
kernel@ubuntu:~$ ls -al /root  
ls: cannot open directory '/root': Permission denied  
kernel@ubuntu:~$
```

- /root 目錄只有 root 權限可以查看

超級使用者-sudo

❑ `sudo -u kernel ls ~kernel`

○ 切換到 kernel 帳號，查看 kernel 家目錄



A terminal window titled 'kernel@ubuntu: ~' showing the command 'sudo -u kernel ls ~kernel' being executed. The output lists the contents of the kernel user's home directory: Desktop, examples.desktop, Pictures, test, Videos, Documents, linux-4.16, Public, test.c, Downloads, Music, Templates, and v4.16.tar.gz.

```
kernel@ubuntu:~$ sudo -u kernel ls ~kernel
Desktop    examples.desktop  Pictures    test        Videos
Documents  linux-4.16        Public      test.c
Downloads  Music             Templates  v4.16.tar.gz
kernel@ubuntu:~$
```

○ /root 目錄只有 root 權限可以查看

超級使用者-sudo

❑ `sudo -g adm more /var/log/syslog`

○ 切換到 adm 群組，查看 (more) 系統紀錄檔案

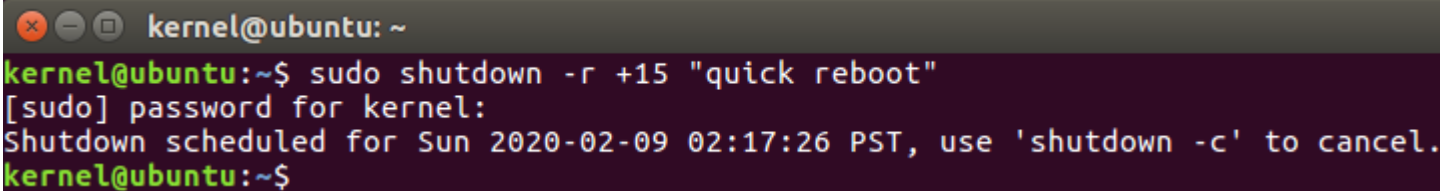
```
kernel@ubuntu: ~  
kernel@ubuntu:~$ sudo -g adm more /var/log/syslog
```

```
kernel@ubuntu: ~  
Feb  9 00:49:07 ubuntu rsyslogd: [origin software="rsyslogd" swVersion="8.16.0"  
x-pid="723" x-info="http://www.rsyslog.com"] rsyslogd was HUPed  
Feb  9 00:49:13 ubuntu anacron[707]: Job `cron.daily' terminated  
Feb  9 00:49:13 ubuntu anacron[707]: Normal exit (1 job run)  
Feb  9 00:55:44 ubuntu dhclient[947]: DHCPREQUEST of 192.168.6.143 on ens33 to 1  
92.168.6.254 port 67 (xid=0x53d47ea4)  
Feb  9 00:55:44 ubuntu dhclient[947]: DHCPACK of 192.168.6.143 from 192.168.6.25  
4  
Feb  9 00:55:44 ubuntu NetworkManager[731]: <info> [1581238544.1229] address  
192.168.6.143  
Feb  9 00:55:44 ubuntu NetworkManager[731]: <info> [1581238544.1230] plen 24  
(255.255.255.0)  
Feb  9 00:55:44 ubuntu NetworkManager[731]: <info> [1581238544.1231] gateway  
192.168.6.2  
Feb  9 00:55:44 ubuntu NetworkManager[731]: <info> [1581238544.1231] server i  
dentifier 192.168.6.254  
Feb  9 00:55:44 ubuntu NetworkManager[731]: <info> [1581238544.1233] lease ti  
me 1800  
Feb  9 00:55:44 ubuntu NetworkManager[731]: <info> [1581238544.1233] nameserv  
er '192.168.6.2'  
Feb  9 00:55:44 ubuntu NetworkManager[731]: <info> [1581238544.1234] domain n  
ame 'localdomain'  
Feb  9 00:55:44 ubuntu NetworkManager[731]: <info> [1581238544.1234] wins '19  
--More-- (10%)
```

超級使用者-sudo

❑ `sudo shutdown -r +15 "quick reboot"`

○ 等待十五分鐘之後重新開機



```
kernel@ubuntu: ~  
kernel@ubuntu:~$ sudo shutdown -r +15 "quick reboot"  
[sudo] password for kernel:  
Shutdown scheduled for Sun 2020-02-09 02:17:26 PST, use 'shutdown -c' to cancel.  
kernel@ubuntu:~$
```

超級使用者-sudoers

❑ /etc/sudoers

- 在此檔案設定的使用者或群組才能使用 sudo 指令。

❑ sudo visudo

- 使用 vi 編輯 /etc/sudoers 檔案，且可以偵錯

❑ sudo more /etc/sudoers

```
# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo  ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "#include" directives:

#include /etc/sudoers.d
kernel@ubuntu:~$
```

超級使用者-sudoers

□ /etc/sudoers

```
root ALL=(ALL:ALL) ALL
%admin ALL=(ALL) ALL
%sudo ALL=(ALL:ALL) ALL
```

被授權使用者/組 主機=[(切換到哪些用戶:組)] [是否需密碼驗證] 命令1,命令2,...

- 主機，表示允許登入的主機，ALL表示所有主機
- []是可以省略
- 命令，是允許執行的命令，ALL表示可執行任何命令。
- %，群組，沒有%是使用者
- (切換到哪些用戶:組)，省略，表示 (root:root)；(ALL:ALL)表示可切換到所有使用者
- 是否需密碼驗證，NOPASSWD:，表示不須密碼驗證，有冒號

超級使用者-sudoers

❑ /etc/sudoers

```
jack mycomputer=/usr/sbin/reboot,/usr/sbin/shutdown
```

- 一般使用者jack在主機mycomputer上,可透過sudo執行reboot和shutdown兩個命令。
- ()省略,相當於(root:root),表示可透過sudo提權到root。

```
lucy ALL=(ALL) NOPASSWD: /bin/useradd
```

- 表示一般使用者lucy可在任何主機,透過sudo執行/bin/useradd命令,且不需輸入密碼。

```
peter ALL=(ALL) NOPASSWD: ALL
```

- 表示一般使用者lucy可在任何主機,透過sudo執行/bin/useradd命令,且不需輸入密碼。

超級使用者-sudoers

□ /etc/sudoers

○lucy ALL=(ALL) chown,chmod,useradd

- 使用者可能創建一個自己的程式，命名為useradd，放在家目錄路徑中，如此能使用root執行"名為useradd的程式"。
- 這是相當危險。要使用絕對路徑。
- 命令的絕對路徑可使用which指令查
- which useradd可查到useradd的絕對路徑: /usr/sbin/useradd

○papi ALL=(root) NOPASSWD: /bin/chown,/usr/sbin/useradd

- 使用者papi能在所有主機提權到root下執行/bin/chown，不必輸入密碼；但執行/usr/sbin/useradd 命令需密碼。因NOPASSWD:只影響其後第一個命令。

被授權使用者/組 主機=[(切換到哪些用戶:組)] [是否需密碼驗證] 命令1,
[(切換到哪些用戶:組)] [是否需密碼驗證]命令2,...

超級使用者-sudoers

□ /etc/sudoers

- papi ALL=/usr/sbin/*,/sbin/*,!/usr/sbin/fdisk

- 通用符號*，命令前加!表示取消該命令。

- papi在所有主機，能執行目錄/usr/sbin和/sbin下所有程式，但fdisk除外。

- 推薦做法，不是直接修改/etc/sudoers，而是將修改寫在/etc/sudoers.d/目錄下的檔中。

- 使用這種方式修改sudoers，需在/etc/sudoers檔的最後行，加上#includedir /etc/sudoers.d一行(預設已有)。

- 任何在/etc/sudoers.d/目錄下，不以~號結尾和不包含.號的檔案，都會被解析成/etc/sudoers的內容。

超級使用者-sudoers

□ /etc/sudoers 變更時間

- sudo 後輸入密碼，不會顯示任何東西，包括星號。若要顯示。
 - 開啟/etc/sudoers，找到Defaults env_reset，修改成：
➤ Defaults env_reset, pwfeedback
- 修改sudo會話時間
 - 成功輸入一次密碼後，可不用再輸入密碼執行幾次sudo命令。
 - 但預設15分鐘後，sudo 命令會再次要求輸入密碼。
 - timestamp_timeout=分鐘數，可修改預設值。
 - -1表示永不過期，強烈不推薦。
- 將時間延長到1小時，Defaults env_reset，改成：
 - Defaults env_reset, pwfeedback, timestamp_timeout=60

超級使用者-sudoers

□ /etc/sudoers使用別名

- 有時 /etc/sudoers 設定較複雜，例如很多使用者及指令組合，可使用別名（alias）管理設定：

```
User_Alias MYACC = accmgr, gtwang, seal  
Cmnd_Alias MYEXE = !/usr/bin/passwd, /usr/bin/passwd [A-Za-z]*, !/usr/bin/passwd root  
Runas_Alias PT = #505, austin, jam, !WHEEL_GRP  
MYACC ALL=(PT) MYEXE
```

- 使用 User_Alias 建立一個帳號別名 MYACC，內容是等號後方那些帳號名稱，
- Cmnd_Alias 是建立指令的別名
- 建立來源主機的別名用 Host_Alias，
- 所有別名都要大寫英文字母命名
- Runas_Alias，欲分權的帳號，也可使用#加上uid。
- 可將冗長設定簡化，且重複使用，日後修改較方便。

超級使用者-sudo

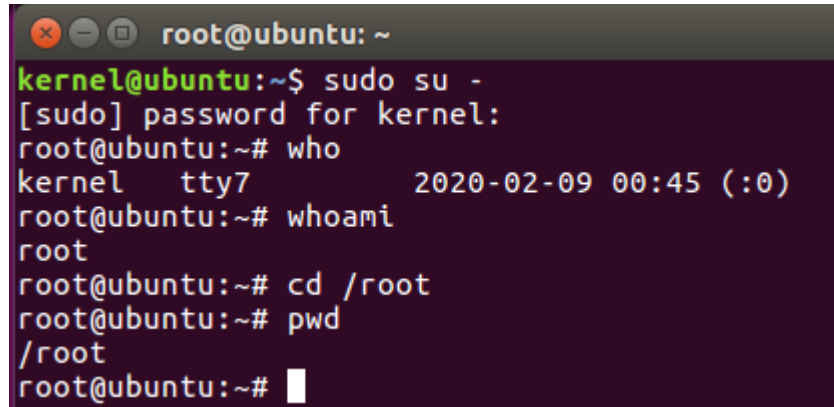
□ sudo指令

	選項	功能
指令名稱/功能/命令使用者 sudo/ (superuser do)/ Any	-b	在背景執行指令
	-E	保留目前使用者的環境變數
	-H	將 HOME 環境變數設為新身份的 HOME 環境變數
	-k	再執行 sudo 時需要輸入自己的密碼
	-l	(小寫的 L)列出被分權的指令
	-p [%u][%h][%H]	改變詢問密碼的提示符號,可接的選項有: "%u" :以使用者為提示符號 "%h" :以主機名稱為提示符號 "%H" :以主機名稱 + domain 名稱為提示符號
	-u <帳號>	以指定的帳號執行指令(無此選項時預設是 root)
	-v	再延長密碼有效期限 5 分鐘
	-V	顯示版本訊息
	--help	指令自帶說明

超級使用者

❑ 使用 root 帳號

- sudo su -



```
root@ubuntu: ~  
kernel@ubuntu:~$ sudo su -  
[sudo] password for kernel:  
root@ubuntu:~# who  
kernel    tty7          2020-02-09 00:45 (:0)  
root@ubuntu:~# whoami  
root  
root@ubuntu:~# cd /root  
root@ubuntu:~# pwd  
/root  
root@ubuntu:~#
```

- who

 - 目前有那些使用者

- whoami

 - 目前下指令的人是誰

- cd

 - 切換目錄

- pwd

 - 顯示目前所在目錄

超級使用者

□ who

	選項	功能
指令名稱/功能/命令 使用者 who/(who is logged on)顯示登入資訊 / Any	-a	all
	-b	最後開機時間
	-d	顯示死進程
	-H	顯示各欄位元的標題資訊列
	-r	顯示 runlevel
	-q	顯示登入用戶和人數
	-w 或 -T	顯示使用者的資訊狀態列
	i am	顯示自己的登入資訊

超級使用者

❑ lastlog

	選項	功能
指令名稱/功能/命令使用者 lastlog/(last login)帳號 登錄查詢 / Any	-b DAYS	顯示從目前算起早於 DAYS 之前的登入者
	-t DAYS	顯示從目前算起 DAYS 天內的登入者
	--u USER	只顯示指定的帳號
	--help	指令自帶說明

超級使用者

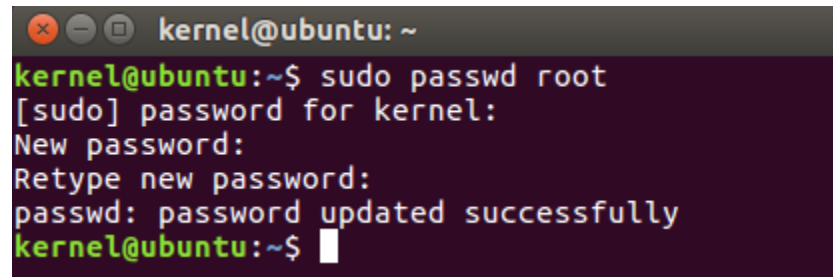
❑ id

	選項	功能
指令名稱/功能/ 命令使用者 id/(print user identity)顯示帳 號 ID / Any	-a	顯示帳號所有資訊(此為預設值)
	-Z	顯示帳號和安全有關的資訊
	-g	顯示有效群組 ID
	-G	顯示所有的群組成員 ID
	-n	顯示帳號或群組名稱(需配合"-g"、"-G"或"-u"使用)
	-r	顯示帳號的 UID/GID 號碼(需配合"-g"、"-G"或"-u"使用)
	-u	顯示帳號有效的 UID 號碼
	--help	指令自帶說明

超級使用者

❑ 變更root密碼

- 安裝完Linux，第一次使用root帳號，要變更密碼
- `sudo passwd root`



```
kernel@ubuntu: ~  
kernel@ubuntu:~$ sudo passwd root  
[sudo] password for kernel:  
New password:  
Retype new password:  
passwd: password updated successfully  
kernel@ubuntu:~$
```

- `sudo`要先輸入自己的密碼
- 輸入兩次root設定的密碼
- 操作完，關掉Terminal，或`exit`

超級使用者

❑ su

- 一般 Linux 使用者取得 root 權限，可對系統進行各種變更動作。
- 帳號 user id 變成 0 (root 的 user id)，但環境變數沒有改變。

```
kernel@ubuntu:~$ su
Password:
root@ubuntu:/home/kernel# id
uid=0(root) gid=0(root) groups=0(root)
root@ubuntu:/home/kernel# env | grep kernel
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/kernel
GPG_AGENT_INFO=/home/kernel/.gnupg/S.gpg-agent:0:1
PWD=/home/kernel
XAUTHORITY=/home/kernel/.Xauthority
root@ubuntu:/home/kernel#
```

- su 後要輸入 root 密碼
- id 顯示目前使用者密碼
- env顯示環境變數
- grep 過濾字串
- 操作完，關掉Terminal，或exit

超級使用者

❑ su -

- 要進行較複雜系統管理，牽涉 root 帳號的環境變數(例如 PATH 或 MAIL 等)，用下面方式。
- 仿照 root 登入，進入一個完整 shell 環境，跟使用 root 重新登入一樣。

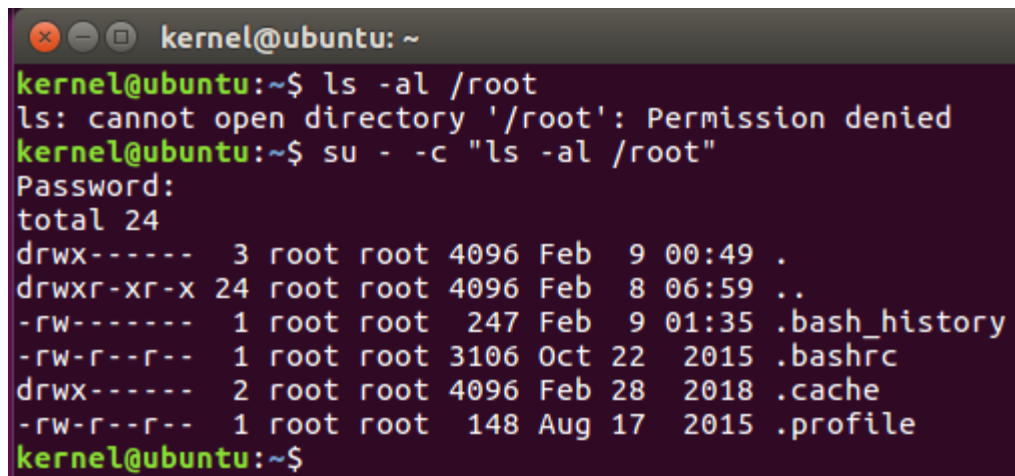
```
root@ubuntu: ~  
kernel@ubuntu:~$ su -  
Password:  
root@ubuntu:~# id  
uid=0(root) gid=0(root) groups=0(root)  
root@ubuntu:~# env | grep root  
USER=root  
MAIL=/var/mail/root  
PWD=/root  
HOME=/root  
LOGNAME=root  
root@ubuntu:~#
```

- 操作完，關掉Terminal，或exit

超級使用者

❑ su - -c "指令"

- 進入新的 shell 後，僅執行一行簡單指令，執行完後馬上跳出。



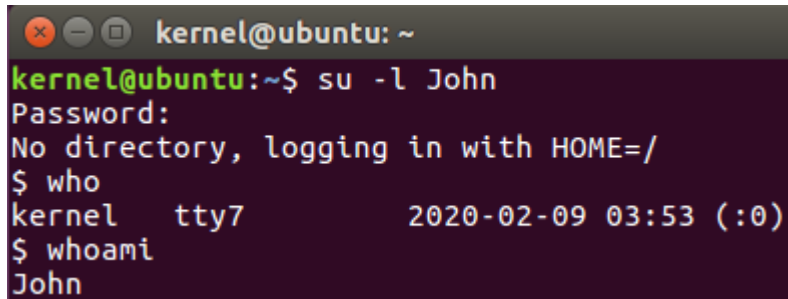
```
kernel@ubuntu: ~  
kernel@ubuntu:~$ ls -al /root  
ls: cannot open directory '/root': Permission denied  
kernel@ubuntu:~$ su - -c "ls -al /root"  
Password:  
total 24  
drwx----- 3 root root 4096 Feb  9 00:49 .  
drwxr-xr-x 24 root root 4096 Feb  8 06:59 ..  
-rw----- 1 root root  247 Feb  9 01:35 .bash_history  
-rw-r--r-- 1 root root 3106 Oct 22  2015 .bashrc  
drwx----- 2 root root 4096 Feb 28  2018 .cache  
-rw-r--r-- 1 root root  148 Aug 17  2015 .profile  
kernel@ubuntu:~$
```

- 一般使用者不能查看 /root 目錄 `ls -al /root`
- `su - -c "ls -al /root"` 改用 root 權限查看，執行完立刻跳出 root
- 操作完，不須關掉 terminal

超級使用者

❑ su -l 帳號

- 取得某帳號權限
- 加入 John帳號，sudo useradd John, sudo passwd John
- 取得 John帳號權限



```
kernel@ubuntu: ~  
kernel@ubuntu:~$ su -l John  
Password:  
No directory, logging in with HOME=/  
$ who  
kernel    tty7          2020-02-09 03:53 (:0)  
$ whoami  
John
```

- exit退出

Linux File System

檔案系統

檔案系統

▣ Linux發行版檔案系統

- 預設採用ext4，ext4是ext3的下一代，
- ext3則是ext2的下一代，ext3是日誌式檔案系統 (JournalFileSystem)，在ext2的格式下加上日誌功能。
- 日誌式檔案系統提供更好的安全性。將整個磁碟做過的更動，像日記一樣完整記錄。一旦發生非預期當機，會在下次啟動時，依日誌記錄動作再做一次，將系統恢復到當機前正常狀態。
- ext2檔案系統需執行fsck指令檢查與修復整個檔案系統。耗費時間且不能保證所有資料都不會流失。
- ext4支援大硬碟，單一檔案最大容量16TB，一個目錄可建立子目錄數量沒有限制。加快檔案讀寫速度，減少檔案不連續存放問題，避免系統使用越久，檔案越不連續，讀寫越慢。

檔案系統

□ Linux發行版檔案系統

- 要將/dev/sda3的檔案系統由ext3轉換為ext4，操作

```
[root@free ~]# umount /dev/sda3          ← 先卸載 /dev/sda3
[root@free ~]# tune2fs -O extents, uninit_bg, dir_index /dev/sda3
tune2fs 1.42 (29-Nov-2011)              將檔案系統轉換為 ext4
Please run e2fsck on the filesystem.

[root@free ~]# e2fsck -y -fD /dev/sda3 ← 檢查並修正檔案系統
e2fsck 1.42 (29-Nov-2011)
Group descriptor 0 checksum is invalid.  Fix? yes
...
```

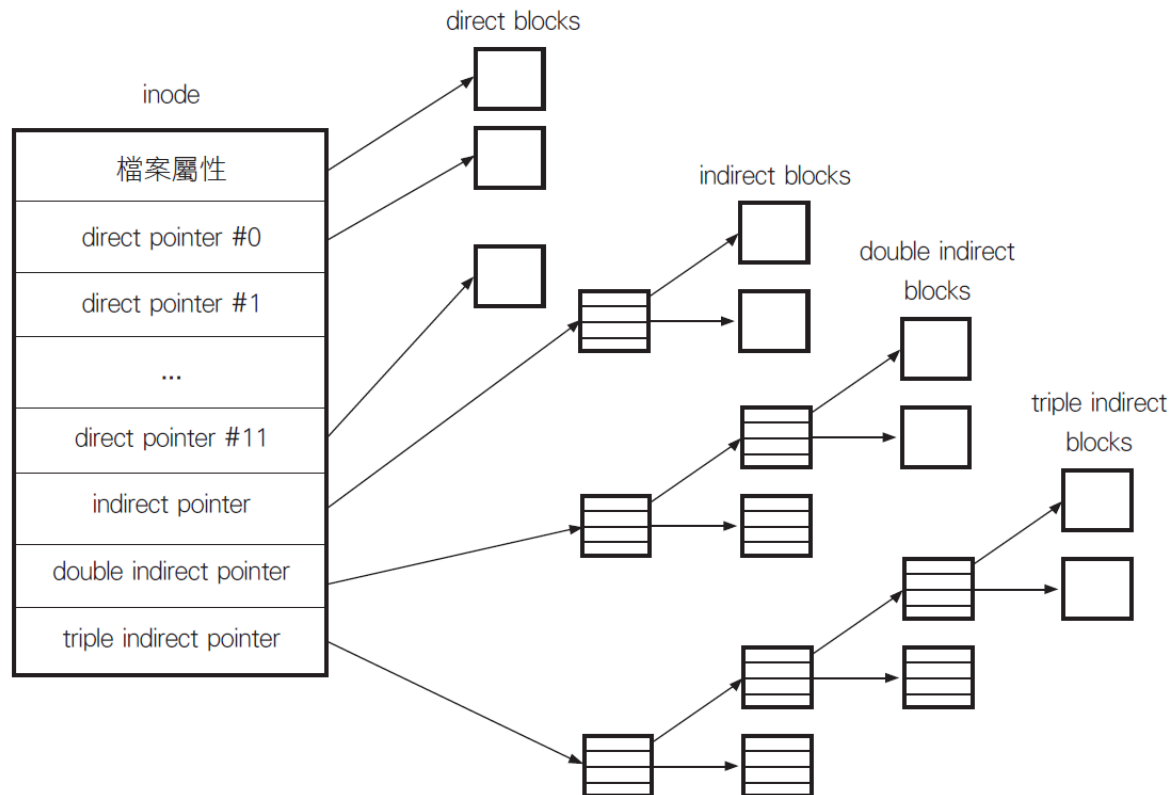
- 建立日誌之後，請修改/etc/fstab檔：重新啟動後，該分割區就開始使用ext4檔案系統

UUID=...	/boot	ext3	defaults	1	2
UUID=...	/	ext4	defaults	1	1
tmpfs	/dev/shm	tmpfs	defaults	0	0
devpts	/dev/pts	devpts	gid=5, mode=620	0	0
sysfs	/sys	sysfs	defaults	0	0
proc	/proc	proc	defaults	0	0
UID=...	swap	swap	defaults	0	0
/dev/sda3	<u>/home1</u>	<u>ext4</u>	defaults	0	0
	掛載點	檔案系統: 將檔案系統改為 ext4			

檔案系統

Linux發行版檔案系統

- ext4、ext3與ext2使用的檔案結構稱為inode(indexnode)。記錄檔案類型、大小、權限、擁有者、檔案連結的數目等屬性，及指向資料區塊(block)的指標(pointer)



檔案系統

□ inode

- 指標指到磁碟實際存放檔案資料的區塊。小的檔案僅需direct blocks空間，大檔案使用indirect blocks、double indirect blocks或triple indirect blocks。
- inode記錄檔案屬性，不實際儲存檔案資料。存放地是資料區塊。每個檔案都會用一個inode，最少佔用一個資料區塊。

□ inode內容：

- 檔案模式(mode)：描述對應的資料類型，可以是檔案、目錄、符號連結(symbolic link)或周邊設備代號(包括儲存設備的分割區編號)等，及權限設定資訊。
- 擁有者資訊：檔案或目錄擁有者的UID與GID。
- 檔案大小(size)：單位以byte計算。
- 時間戳記(time stamp)：資料最初建立時間與最後修改時間。

檔案系統

□ inode 內容：

- 資料區塊位址(address of data block)：存放檔案需使用資料區塊。若inode對應實體檔案(非如/proc目錄內虛擬檔案)，則會紀錄資料區塊位址，讓系統找到並使用。一個inode指向12個資料區塊，若檔案太大會用間接指向指標(indirect pointer)，透過另一個資料區塊指向更多資料區塊。

基本目錄檔案指令

❑ clear

- 清除螢幕

❑ cd

- 變換工作路徑。

❑ cd ..

- 上一層目錄

❑ cd ../xx

- 到相對路徑

❑ pwd

- 顯示目前所在目錄

基本目錄檔案指令

❑ ls

- 顯示檔案名稱與內容的指令彩色顯示檔案資料
- `#ls -l`詳細列出檔案系統結構
- `#ls -a`顯示隱藏檔（以"."開頭的檔案）
- `#ls -al`同時顯示隱藏檔與詳細資料
- `#ls -al | more`將檔案內容以一頁一頁顯示

❑ cat

- 將檔案內容列出
- 例如在/root下有一個檔名為.bashrc，是隱藏檔，按下cd回到/root目錄後，執行：
- `#cat .bashrc`

基本目錄檔案指令

❑ more

- 用more來一頁頁讀取檔案內容
- more可與其他程式合併使用，例如ls -al | more
- #more .bashrc
- #ls -al | more

❑ mkdir

- 建立新的目錄
- #mkdir test
- #ls -l
- 再執行ls -l後，就可看到test目錄

基本目錄檔案指令

❑ rm -irf

- 移除
- -i 指檔案被刪除前會確認。
- -rf 目錄下的東檔案一起刪除
- #rm test
- #rm -rf test

❑ rmdir

- 移除目錄的指令。
- 若欲移除的目錄有檔案或其他目錄，就無法移除，要用rm -rf

基本目錄檔案指令

❑ mv

- 移動檔案或目錄，例如要將.bashrc檔案移動至根目錄下，
 - #mv .bashrc /
- 將檔案移動至目前的工作目錄，加上"."
 - #mv /.bashrc .
- 語法：mv來源檔（或目錄）目的檔（或目錄）

❑ cp

- copy。例如要將.bashrc檔案複製到/home底下
- #cp .bashrc /home
- 語法：cp來源檔目的檔

基本目錄檔案指令

❑ find

- 找檔案
- #find / -name bin
- 在/目錄（根目錄）下尋找檔名（-name）為bin的檔案
- 語法：find路徑-name檔名

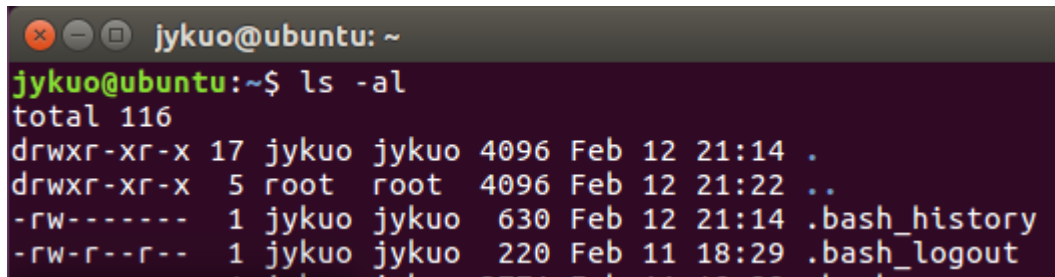
基本目錄檔案指令

❑ whereis

- whereis 利用曾找過的系統資訊內的資料找檔案，速度快
- whereis 找不到，不代表該檔案真的不存在
- #whereis bin
- bin:/usr/bin

檔案權限

❑ ls -al



```
jykuo@ubuntu: ~  
jykuo@ubuntu:~$ ls -al  
total 116  
drwxr-xr-x 17 jykuo jykuo 4096 Feb 12 21:14 .  
drwxr-xr-x  5 root  root  4096 Feb 12 21:22 ..  
-rw-----  1 jykuo jykuo   630 Feb 12 21:14 .bash_history  
-rw-r--r--  1 jykuo jykuo   220 Feb 11 18:29 .bash_logout
```

○ 10個字元中的第1個字元用於標示檔案屬性：

- d：目錄。目錄視為特殊檔案。
- -：普通檔案。
- l：符號連結的檔案，實際指向另一個檔案。
- b、c：分別代表區塊設備和其他周邊設備，是特殊型態檔案。
- s、p：這些檔案關係到系統資料結構和管線，通常很少見到。

檔案權限

□ 一般權限

- 第2～10字元，每3個一組，分別標示擁有者、群組、任何人
 - r(Read，讀取)：對檔案而言，使用者具讀取內容權限；對目錄而言，使用者擁有瀏覽目錄內容權限(不一定可讀取該目錄下的檔案，是否可讀取，取決於要讀取"檔案"的"r"讀取權)。
 - w(Write，寫入)：對檔案而言，使用者具有修改檔案權限；對目錄而言，使用者具有刪除、或移動目錄內檔案權限。
 - x(eXecute，執行)：對檔案而言，使用者具執行檔案權限；對目錄而言，使用者具進入目錄權限。
 - -：表示不具該權限。
- Linux系統執行檔，副檔名毋需為.exe，只要可執行權限。

檔案權限

□ 一般權限

- `-rwx-----`：檔案擁有者對檔案具有讀取、寫入與執行的權限。
- `-rwxr--r--`：檔案擁有者具有讀、寫與執行的權限，同群組及其他使用者具有讀取的權限。
- `-rw-rw-r--`：檔案擁有者與同群組的使用者對檔案具有讀寫的權限，其他使用者僅具有讀取權限。
- `drwx--x--x`：目錄擁有者具有讀、寫與進入目錄的權限，同群組及其他使用者僅能進入該目錄，卻無法讀取檔案列表。
- `drwx-----`：除了目錄擁有者具有完整的權限之外，同群組與其他使用者對該目錄沒有任何權限。
- 使用者都擁有家目錄，預設權限為"`drwx-----`"，表示目錄擁有者本身具備全部權限，而同群組與其他使用者沒有任何權限。

檔案權限

❑ 特殊權限SUID(SetUID)

- 使用者無特殊需求，不應開啟特殊權限檔案，避免安全漏洞。

- SUID

- 可執行檔案具此權限，能得到特權，可任意存取該檔案擁有者能使用的全部系統資源。

- 擁有此權限的檔案，可任意存取整個群組所能使用的系統資源。

- T(Sticky)：開啟暫存目錄/tmp的Sticky權限，存放該目錄的檔案，僅准許其擁有者刪除與搬移，避免其他使用者誤用。

- 須將檔案放在具Sticky權限目錄，才能讓此權限產生效用。

- 特殊權限SUID、SGID、Sticky佔用x位置，假設同時開啟執行權限和SUID、SGID與Sticky，則其權限標示型態：

```
-rwSr-Sr-T  1 root    root   Dec 2 21:47 showme
```


變更檔案權限

❑ chmod

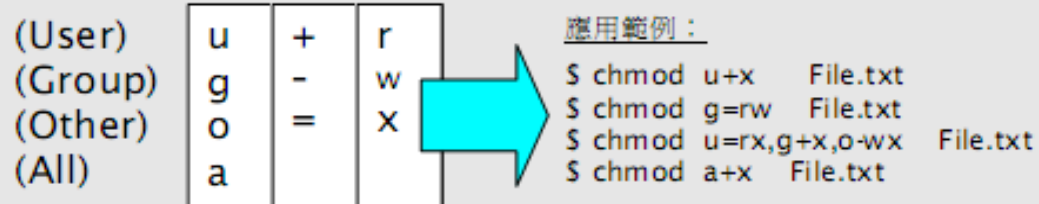
- 變更檔案屬性，檔案屬性，r為4、w為2，x為1。
- 檔案或目錄的權限，是用"rwx"3個字元重複3次形成9個字元，分別代表擁有者、同群組使用者和其他使用者的權限。
- 建立檔案所有人都可讀，-rw-r--r--，三個群組分別r+w=6，r=4，r=4
- #chmod 644 .bashrc
- #ls -al .bashrc
- -rw-r--r-- 1 root root 216 Apr 8 13:54 .bashrc

-rwx-----：等於數字標示700。
-rwxr--r--：等於數字標示744。
-rw-rw-r-x：等於數字標示665。
drwx--x--x：等於數字標示711。
drwx-----：等於數字標示700

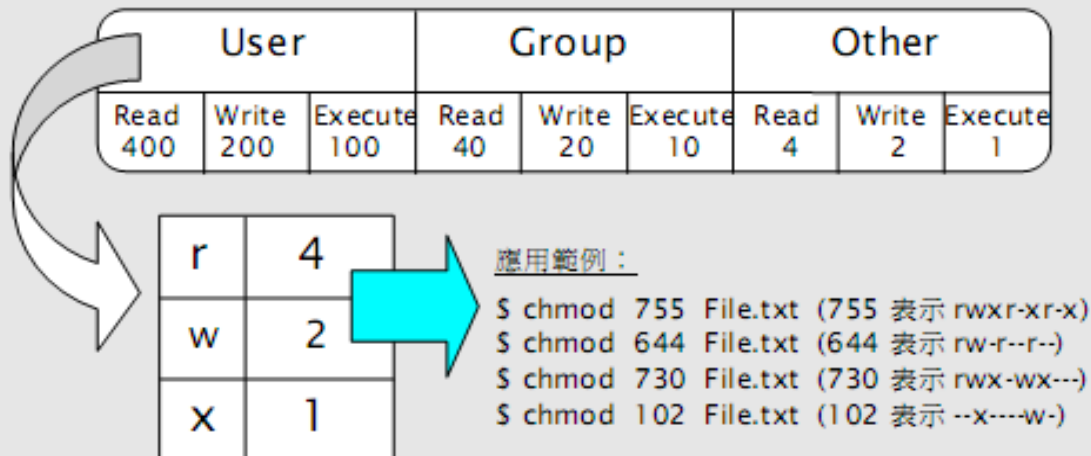
變更檔案權限

❑ chmod

⚡ 符號表示法：



⚡ 數字表示法：



變更檔案所屬群組

▣ 群組管理

- 共享目錄或檔案。
- 把某目錄設Group ID Bit，任何使用者將檔案移到此目錄內，該檔案的群組會自動變成和目錄群組相同。
- 許多帳號設同一群組或加入某附加群組，要分享的目錄或檔案群組的擁有者設共同群組名稱，可共享目錄或檔案。

▣ chgrp

- 改變檔案或目錄的『擁有群組』。

變更檔案所屬群組

□ 群組管理

chgrp/ (change group) 變更群組擁有者/ Any	-c	只顯示異動部分
	-f	不顯示錯誤
	-h	只對 符號連結 檔變更但不影響目的檔
	-R	遞回(recursive)將目錄下所有檔案及子目錄變更
	-v	顯示處理過程
	--help	指令自帶說明

- `chgrp rd_grp file`←將檔案的群組改為"rd_grp"
- `chgrp -R sub_grp ~/homework`←將家目錄內目錄"homework"下所有的檔案及子目錄一併變更群組擁有者為"sub_grp"

變更擁有者

❑ chown(chage owner)

- 若要把檔案給其他使用者修改，就需修改檔案、目錄擁有者、所屬群組。
- 變更擁有者:chown擁有者檔案或目錄
- 變更擁有者和所屬群組:chown擁有者.群組成員檔案或目錄。
- 只變更所屬群組:chown .群組成員檔案或目錄。
 - `chown aaa my_file`←將檔案"my_file"擁有者改為"aaa"
 - `chwon aaa . bbb my_file`←將檔案"my_file"擁有者改為"aaa"，群組擁有者改為"bbb"
 - `chwon . bbb my_file`←將檔案"my_file"群組擁有者改為"bbb"
 - 將整個目錄下的檔案都改變擁有者與擁有群組
 - `chown -R username: group name directory(ex. chown -R root:root/root)`
 - 例如root，copy一個檔案給vbird，需將檔案擁有人改成vbird

變更擁有者

❑ chown(chage owner)

chown/ (change owner) 變更擁有者/ Any	-c	只顯示異動部分
	-f	不顯示錯誤
	-h	只對 符號連結 檔變更，但不影響目的檔
	-R	遞回(recursive)將目錄下所有的檔案及子目錄一併變更
	-v	顯示處理過程
	--help	指令自帶說明

查看硬碟空間

▣ df查看硬碟空間

○若規劃多的硬碟，則可查看硬碟空間資訊：

○#df

```
root@islab-virtual-machine:/# df
Filesystem      1K-blocks    Used Available Use% Mounted on
udev            4031204         0   4031204   0% /dev
tmpfs           813936       1480    812456   1% /run
/dev/sda5       50771456 9286716  38873284  20% /
tmpfs           4069676         0   4069676   0% /dev/shm
tmpfs           5120          0        5120   0% /run/lock
tmpfs           4069676         0   4069676   0% /sys/fs/cgroup
/dev/loop0       63488       63488         0 100% /snap/core20/1611
```

○Filesystem是硬碟劃分表，Used指使用掉的硬碟空間（KB），Available是剩下空間，Mounted on是硬碟代表哪一個目錄。

○根目錄(/)在sda5這顆硬碟，總空間有50771456KB，剩下可用空間為38873284KB。要將資料型態以MB數顯示，可輸入df -m。

查看硬碟空間

▣ du

- 查看目錄內所有檔案使用掉的空間的情況：
- #du -m
- du預設的檔案輸出資料為KB，以參數-m使檔案顯示為MB。

轉換複製

□ dd

- 指定讀取來源，將讀取內容寫入指定目的。
- 可指定每次存取多少量，才進行寫入動作。
- 可指定讀取時先移動到某個位置才讀取，寫入時可指定偏移位置。
- 參數說明
 - if為 input file，表示指定讀取來源，預設為 stdin
 - of為 output file，表示指定寫入目的，預設為 stdout
 - bs為 block size，表示讀入與寫入的大小
 - count表示處理的次數

轉換複製

□ dd

- Linux 週邊裝置視為檔案，在 /dev 目錄內許多週邊裝置檔案清單，例如 IDE 硬碟為 /dev/hda、/dev/hdb，而 SATA 與 SCSI 使用 /dev/sda、/dev/sdb。
- 硬碟複製，dd if=/dev/hda of=/dev/hdb bs=4096k
 - 讀取來源 hda，IDE0 上 Primary Master 的 HD，複製寫入到 IDE0 上 Primary Slave 的 HD，指定每次讀入與寫入的處理量為 4M。
 - 指令運作結束，hdb 的硬碟會與 hda 硬碟內容完全相同。
 - 兩顆硬碟容量規格相同較適合使用。hdb 硬碟較大也可進行，後續沒分配到的空間需手動建立分割區才可使用。
- dd 運作為磁區對磁區，若硬碟有 320G 空間，若只放 100MB 資料，dd 仍要運作到 320G。

轉換複製

- ❑ 清空裝置的資料，`dd if=/dev/zero of=/dev/hdb bs=4096k`
 - 指定讀取來源為 `/dev/zero`，是特殊的字元裝置，讀取內容會得到 `0x00`的空資料。
 - `hdb` 硬碟內容填入空資料，硬碟清空。
- ❑ 製作空的大檔案
 - `dd if=/dev/zero of=file bs=1M count=100`
 - `dd if=/dev/zero of=fd.img bs=1024 count=102400`
 - 假設`BS=1024`，則每sector大小是1024位元組，可省略不設，預設大小512位元組。`count`表示需建檔的大小，以sector為單位，若沒設`bs`，預設每sector大小是512位元組，若 `fd.img` 大小是1M，則`count=2000`。

轉換複製

- ❑ 備份 MBR 資料，`dd if=/dev/hda of=mbr.dat bs=512 count=1`
 - 電腦 HD 開機資料位於硬碟最前面位置的 MBR(Master Boot Record)。MBR 若故障毀損系統無法正常開機。
 - MBR 位於第 0 軌，第 0 面第一磁區，佔用 512 bytes。包含 boot loader 開機程式與 Partition table 分割表。
 - 電腦開機 BIOS 讀入硬碟，啟動 boot loader，依據 partition table 決定進入那一個作業系統。
 - 讀取來源為 hda，備份儲存的檔案為 mbr.dat，指定處理量 512 bytes，讀寫一次即可。後續 MBR 毀損，使用命令回寫。
 - `dd if=/dev/mbr.dat of=/dev/hda bs=512 count=1`

轉換複製

- ❑ 備份 USB隨身碟 `dd if=/dev/sda of=usb-backup.img bs=4096k`
 - 若 USB 隨身碟裝置為 `sda`，備份隨身碟資料，提供後續複製：
 - 得到 `usb-backup.img` 檔案。後續若要複製產生另外相同內容的 USB 磁碟，使用：`dd if=usb-backup.img of=/dev/sda bs=4096k`
- ❑ 存取 Image 檔案而不需還原
 - Linux 內建存取 Image 功能，可線上掛載 Image 檔案到目錄，後續可到該目錄讀寫檔案內容。
 - 例如將系統 `hda1` 分割區資料備份成為 `hda1.img` 檔案：
 - `dd if=/dev/hda bs=hda1.img bs=4096k`
 - 要存取 `hda1.img` 檔案內容，使用 `mount` 指令。
 - `mount -o loop hda1.img /mnt`
 - `hda1` 分割區資料可在 `/mnt` 目錄內存取
 - `mount` 搭配 `-t` 表示檔案系統類型：`mount -t vfat -o loop hda1.img /mnt`

轉換複製

- ❑ 讀取 ISO 檔案：`mount -t iso9660 -o loop filename.iso.img /mnt`
 - DVD 的 ISO，使用 `-t udf`。
 - `mount -t udf -o loop filename.iso.img /mnt`
- ❑ swap 規劃配置
 - 先建立 `swapfile.dat` 共 120M：
 - `dd if=/dev/zero of=swapfile.dat bs=1024k count=120`
 - 格式化 `swapfile.dat` 成為 linux swap 結構：`mkswap swapfile.dat`
 - 啟用 `swapfile` 的使用：`swapon swapfile.dat`
 - 暫時增加系統虛擬記憶體的可利用空間。

基本指令-檔案與目錄管理

❑ quotacheck , edquota

- 限制使用者在Linux主機上的硬碟使用容量。
- `sudo apt-get install quota`

建立連結

❑ 硬連結(hardlink)

- 檔案分享的一種方法，不需複製一份檔案浪費磁碟空間。

❑ ln

- ln -s 真實目錄或檔案連結的目錄或檔案：
- 連結檔案或目錄
- 例如將/usr/bin這個目錄連接到/root底下
- #ln -s /usr/bin bin
- /root下的bin目錄中的所有檔案都是/usr/bin裡的檔案，若進入/root/bin刪除檔案，等於將/usr/bin內的檔案刪除
- 執行ls-l指令，看複製與連結的檔案有甚麼不同

建立連結

❑ 硬連結(hardlink)

- 若使用者無存取來源檔案權限，系統仍允許產生檔案連結，但該連結的檔案屬性仍與來源檔案屬性相同。無法讀取的檔案即使以連結方式到家目錄，仍沒有權限存取該檔案。
- 檔案連結數是2，表示此檔案除本身外，還有另一個分身。假使再對該檔案建立連結，連結數會增加。每刪除一個，連結數遞減，直降為1，該檔案在檔案系統不存在任何分身。
- 連結的檔案實際指向磁碟中相同資料，因每個檔案僅佔一個inode，所以inode編號應一樣。
- 執行ls-i指令查看檔案inode編號，若是複製各自擁有inode編號。

```
[lambert@free ~]$ ls -i LambertLink
10423 LambertLink
[lambert@free ~]$ ls -i /var/tmp/ForEveryone
10423 /var/tmp/ForEveryone
```

建立連結

❑ 符號連結(軟式連結)ln-s

○ 執行ls-l指令觀看

```
[lambert@free ~]$ ln -s LambertFile SymLink
```

對 LambertFile 建立符號連結，檔名為 SymLink

-rw-r--r--	1	lambert	lambert	1502892	Jun 3	9:14	LambertFile
-rw-r--r--	2	cassia	cassia	1502892	Jun 3	19:35	LambertLink
lrwxrwxrwx	1	lambert	lambert	11	Jun 3	20:20	SymLink -> LambertFile

這裡指向建立符號連結的原始檔案

- 檔案LambertFile和SymLink的連結數都沒改變，而SymLink檔案權限第1個字元"l"，表示符號連結，權限為"rwxrwxrwx"全部開放，代表真正權限以所指檔案為準，符號連結不做限制。

建立連結

❑ 符號連結(軟式連結)ln-s

- 符號連結並不保存檔案資料，真正內容是一個字串指向原來的檔案，類似"捷徑"，因此若把其指向的檔案刪除或更改檔名，則SymLink就會指向一個不存在的檔案，內容會變成空白。符號連結本身會佔用一個inode：

```
[lambert@free ~]$ ls -li SymLink  
366959 SymLink
```

- 由於連結方式不同，硬連結與符號連結差異：
 - 當原檔刪除後，符號連結會失效，硬連結仍可繼續使用。
 - 硬連結只能連結同一個分割區內檔案，符號連結因只是一個指向檔案字串，所以可跨越不同分割區。
 - 硬連結不能連結目錄，因目錄的inode中，計算連結數的欄位已有其他用途。符號連結可指向目錄，如同真的目錄一樣使用。

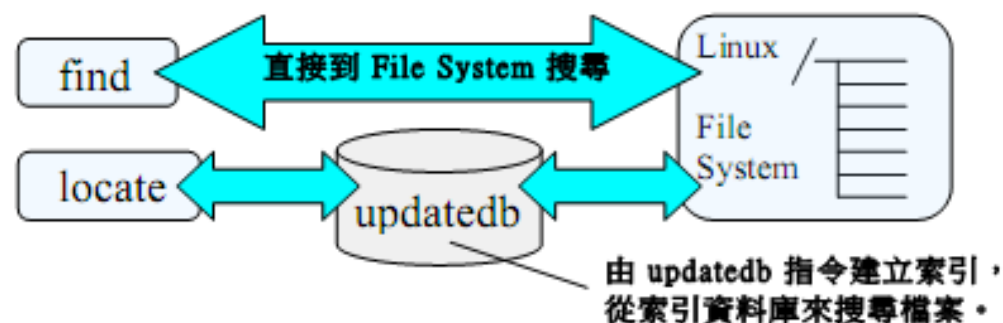
指令

查詢狀態列	
ls	查看目錄內容
pwd	查看目前所在目錄位置
目錄類	
cd	切換目錄
mkdir	建立目錄
rmdir	刪除空目錄（如果目錄裡有資料，要用 rm -R）
檔案類	
touch	建立空檔案（touch 實際的用法是改變檔案建立時間）
rm	刪除檔案（刪除目錄可打 rm -R）
cp	複製檔案（複製目錄可打 cp -R）
mv	搬移檔案或更名
ln	連結檔案（類似建立捷徑的意思）
觀看檔案類	
cat	觀看檔案內容
less	觀看檔案內容（以頁顯示，可以上下捲動）
more	觀看檔案內容（以頁顯示，不能上下捲動）
head	觀看檔案的開頭起始行（預設為開頭 10 行）
tail	觀看檔案的結尾倒數行（預設為倒數 10 行）
查詢手冊類	
man	查看指令的用法（manpage）
info	查看指令的用法（同 manpage，但具有超連結）

指令

尋找系統檔案	
find	搜尋系統中的檔案
locate	搜尋系統中的檔案 (透過已整理的資料庫)
查詢指令相關檔案	
whereis	搜尋指令名稱的相關檔案
which	搜尋指令名稱所存在的位置 (根據 PATH)
type	查詢指令名稱所存在的位置以及種類

⚡ 【find、locate】搜尋檔案示意圖：



指令

更改檔案權限

chmod	更改檔案的存取權限
chown	更改檔案的擁有者或擁有群組
chgrp	更改檔案的擁有群組



「chmod」基礎用法：

```
$ chmod [選項] [新的權限] [檔案]
$ chmod u=rwx,g=rx,o=rx File.txt
$ chmod -Rf 755 Directory
```

「遞迴式、強制」的將目錄底下所有的檔案更改為新的權限。

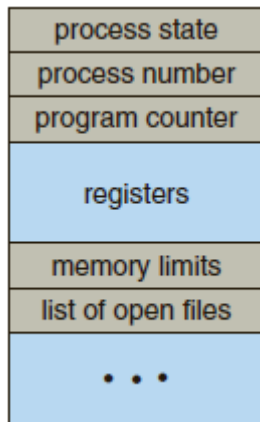
權限設定有兩種方式，分別為「符號表示法」與「數字表示法」，筆者在下面為各位做說明與示範。

Linux Process

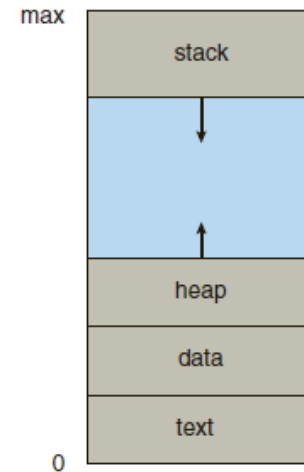
Process

□ 程序(Process)

- 在系統的工作單元。
- 一個程式(Program)被載入記憶體執行。
 - Program是passive，Process是active
- 在記憶體有text (code), data, stack (function call), heap (variable)。
- 作業系統有PCB，紀錄Process執行資訊
 - 程式計數器PC (program counter)，紀錄下一個要執行的指令



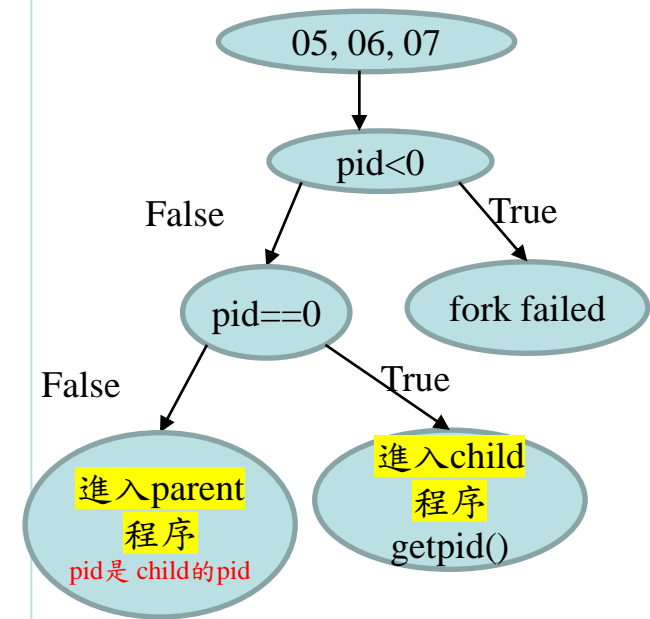
Process control block (PCB)



Process在記憶體的內容

程序(Process) – fork()

```
01 #include <sys/types.h>      // f0.c
02 #include <sys/wait.h>
03 #include <stdio.h>
04 #include <unistd.h>
05 int main() {
06     pid_t pid;
07     pid = fork();    //fork a child process
08                     //若 pid>0, 進入parent process, pid 表示child 的pid
09                     //若 pid==0, 進入child process
10     if (pid < 0) {    //error occurred
11         fprintf(stderr, "Fork Failed");
12         return 1;
13     }
14     else if (pid == 0) { // 處於 child process
15         printf("0=> Child pid=%d\n", getpid());
16     }
17     else {            // 處於 parent process
18         wait(NULL);    // parent wait child complete
19         printf("1=> Child pid=%d\n", pid);
20         printf("1=> parent pid=%d\n", getpid());
21         printf("Child Complete\n");
22     }
23     return 0;
24 }
```



Process

❑ 程序(Process) – fork()

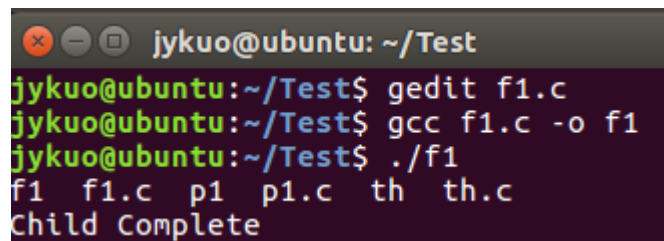
```
#include <sys/types.h>          // f1.c
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
int main() {
    pid_t pid;
    pid = fork();    //fork a child process
    if (pid < 0) {    // error occurred
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { // child process
        execlp("/bin/ls", "ls", NULL);
    }
    else {            // parent process
        wait(NULL);    // parent wait child complete
        printf("Child Complete\n");
    }
    return 0;
}
```

Process

❑ 程序(Process) – fork()

- gcc f1.c -o f1

- ./f1



```
jykuo@ubuntu: ~/Test
jykuo@ubuntu:~/Test$ gedit f1.c
jykuo@ubuntu:~/Test$ gcc f1.c -o f1
jykuo@ubuntu:~/Test$ ./f1
f1 f1.c p1 p1.c th th.c
Child Complete
```

Process

□ `pid = wait(NULL); pid = wait(&status);`

○ 某程序呼叫`wait`，立即暫停自己，判斷是否某子程序已跳出。

➢ 若有，`wait`會收集此子程序資訊，銷毀後返回；

➢ 若無，`wait`會一直等待直到有一個出現。

○ `status`儲存程序跳出時狀態 (int型別指標)。

➢ 若只要刪除子程序，不管跳出狀態，就設定`NULL`。

○ `wait()`呼叫成功，會回傳子程序ID

➢ 若無子程序，呼叫失敗，回傳-1，`errno`為`ECHILD`。

○ `WIFEXITED(status)` 巨集顯示子程序是否正常跳出，若是，回傳一個非零值。

○ `WEXITSTATUS(status)`

➢ 當`WIFEXITED`回傳非零值，此巨集取得子程序回傳值

➢ 子程序使用`exit(5)`跳出，`WEXITSTATUS(status)`回傳5；

➢ 若子程序非正常跳出，`WIFEXITED`返回0，此值無意義。

Exercise

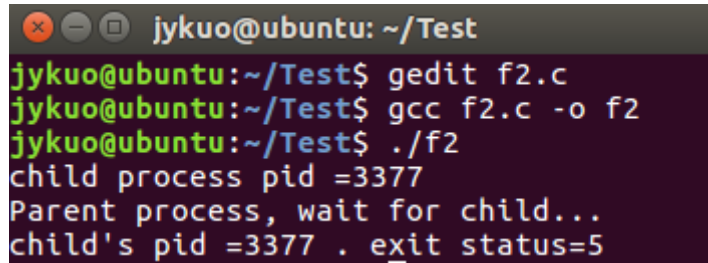
❑ 程序(Process) – 執行以下程式，查看 ps -aux

```
#include <stdio.h>      // f2.c
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main() {
    pid_t pid;
    int status, i;
    if(fork() == 0) {
        printf("child process pid =%d\n", getpid());
        exit(5);
    }
    else {
        sleep(1);
        printf("Parent process, wait for child...\n");
        pid = wait(&status);      //回傳等待的child程序的pid, 回傳值
        i = WEXITSTATUS(status);
        printf("child's pid =%d . exit status=%d\n", pid, i);
    }
    return 0;
}
```

Process

❑ 程序(Process) – fork()

- gcc f2.c -o f2
- ./f2



```
jykuo@ubuntu: ~/Test
jykuo@ubuntu:~/Test$ gedit f2.c
jykuo@ubuntu:~/Test$ gcc f2.c -o f2
jykuo@ubuntu:~/Test$ ./f2
child process pid =3377
Parent process, wait for child...
child's pid =3377 . exit status=5
```

Exercise

```
#include <stdio.h>      // f1.c
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main() {
    pid_t pid;
    int status, i;
    if(fork() == 0) {
        printf("child process pid =%d\n", getpid());
        // 第二版修改 lsxx 變成錯誤執行
        if(execlp("ls", "ls_process", "-l", NULL) < 0) {
            printf("after execlp fail, pid=%d\n", getpid());
            exit(8);
        }
        exit(5);
    }
    else {
        sleep(1);
        printf("Parent process, wait for child...\n");
        pid = wait(&status);          // 回傳等待的child程序的pid, 回傳值
        if (WIFEXITED(status) > 0) {
            printf("child's pid =%d, %d, exit status=%d\n", pid, WIFEXITED(status), WEXITSTATUS(status));
        }
        else
            printf("child's pid =%d, WIF status=%d\n", pid, WIFEXITED(status));
    }
    return 0;
}
```

```
u20@ubuntu:~/Test$ gedit f1.c
u20@ubuntu:~/Test$ gcc f1.c -o f1
u20@ubuntu:~/Test$ ./f1
child process pid =8706
total 24
-rwxrwxr-x 1 u20 u20 17048 Oct 23 16:00 f1
-rw-rw-r-- 1 u20 u20 930 Oct 23 16:00 f1.c
Parent process, wait for child...
child's pid =8706, 1, exit status=0
u20@ubuntu:~/Test$ gedit f1.c
u20@ubuntu:~/Test$ gcc f1.c -o f1
u20@ubuntu:~/Test$ ./f1
child process pid =8721
after execlp fail, pid=8721
Parent process, wait for child...
child's pid =8721, 1, exit status=8
```

Exercise 產生多個process

❑ `pstree -p {pid}`

```
#include <stdio.h>          // nf.c
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main() {
    pid_t child[2], self = getpid(), p;
    child[0] = fork();
    if (child[0]>0) child[1] = fork();
    if (child[0]>0 && child[1]>0) {
        sleep(10);
        printf("\nParent process, wait for child...\n");
        p = wait(NULL);           //等待child程序
        printf("child's pid1 =%d, pid2=%d, p=%d\n", child[0], child[1], p);
        printf("parent pid =%d\n", self);
    }
    else
        sleep(10);
    printf("child process pid =%d\n", getpid());
    return 0;
}
```

```
u20@ubuntu:~/Test$ gedit nf.c
u20@ubuntu:~/Test$ gcc nf.c -o nf
u20@ubuntu:~/Test$ ./nf&
[1] 2819
u20@ubuntu:~/Test$ pstree -p 2819
nf(2819)─nf(2820)
        └nf(2821)
u20@ubuntu:~/Test$
Parent process, wait for child...
child process pid =2821
child process pid =2820
child's pid1 =2820, pid2=2821, p=2820
parent pid =2819
child process pid =2819

[1]+  Done                  ./nf
```


Process Fork 共用變數 pipe

□ pipe解決共用變數

○ 創立 pipe 的 file descriptor

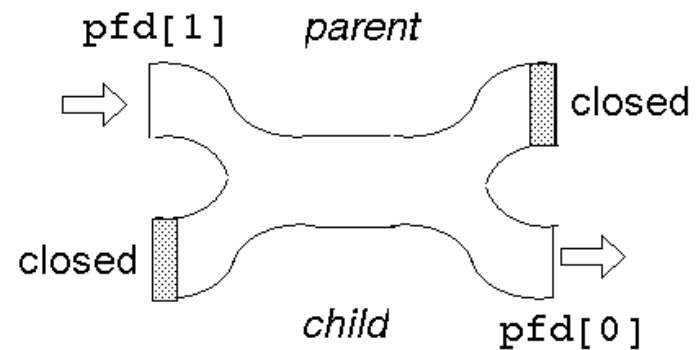
- `int fd[2]`，`fd[0]`讀資料，`fd[1]`寫資料
- `pipe(fd)` 呼叫pipe作使用

○ 傳送端code

- `close(fd[0])` // 傳送端沒有要接收資料，故關閉讀取
- `write(fd[1], %value, sizeof(value))` // 參數fd寫入端，傳送資料的buffer，傳送資料大小
- `close(fd[1])` // 寫完後關閉

○ 接收端

- `close(fd[1])` // 接收端沒有接收資料，關閉接收
- `read(fd[0], %value, sizeof(value))` // 參數為fd傳送端，接收資料的buffer，接收資料大小
- `close(fd[0])` // 接收結束後關閉



Process Fork 共用變數 pipe

□ pipe解決共用變數

```
#include <sys/types.h>           // f3.c
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main() {
    int value=0;
    pid_t pid;
    int fd[2];
    pipe(fd);
    pid = fork();    //fork a child process
    if (pid < 0) {    // error occurred
        fprintf(stderr, "Fork Failed");
        return 1;
    }
}
```

```
kjy@ubuntu:~/Test$ gcc f3.c -o f3
kjy@ubuntu:~/Test$ ./f3
child value=15
Child Complete
parent value=15
```

```
else if (pid == 0) { // child process
    close(fd[0]);
    value = 15;
    printf("child value=%d\n", value);
    write(fd[1], &value, sizeof(value));
    close(fd[1]);
}
else {                // parent process
    wait(NULL);        // parent wait child complete
    read(fd[0], &value, sizeof(value));
    printf("Child Complete\n");
    printf("parent value=%d\n", value);
    close(fd[0]);
}
return 0;
}
```

IPC 共享記憶體

```
// shm1.c - server
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define SHM_SIZE 50
int main() {
    char c;
    int shm_id;
    key_t key;
    char* shm, *s;
    // 命名共享記憶體(shared memory segment) "567".
    key = 567;
    // Create the segment.
    shm_id = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
    if (shm_id < 0) {
        perror("shmget");
        printf("creat fail");
        exit(1);
    }
}
```

```
//attach 共享記憶體到本 process的變數
if ((shm=shmat(shm_id, NULL, 0))!=(char*)-1){
    perror("shmat");
    printf("attach fail");
    exit(1);
}
// 寫入資料到共用記憶體，等待其他 process 讀取
s = shm;
// 第一個字元表示要寫入資料 byte 個數
*s++ = 20;
for (c = 'a'; c <= 'z'; c++) {
    *s++ = c;
}
//等待其他 process 修改記憶體第一個字元 '*',
// 表示已經讀取本 process 寫入資料
while (*shm != '*') {
    sleep(1);
}
return 0;
}
```

IPC 共享記憶體

```
// shm2.c - client
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define SHM_SIZE 27
int main() {
    int shm_id;
    int size=0;
    key_t key;
    char* shm, *s;
    //取得共享記憶體名稱 "567", 由 server 造出
    key = 567;
    // 連結到共享記憶體名稱.
    if ((shm_id = shmget(key, SHM_SIZE, 0666)) < 0) {
        perror("shmget");
        exit(1);
    }
    // attach 共享記憶體到本 process 的變數
    if ((shm = shmat(shm_id, NULL, 0)) == (char*) - 1) {
        perror("shmat");
        exit(1);
    }
}
```

```
// 讀取 server 在共享記憶體寫入的資料
s = shm;
// 第一個字元表示server寫入資料 byte 個數
size = *s++;
for (int i = 0; i < size; i++, s++) {
    putchar(*s);
}
putchar('\n');
// 修改記憶體第一個字元 '*',
// 表示已經讀取本 process 寫入資料
*shm = '*';
return 0;
}
```

```
u20@ubuntu:~/Test$ gcc shm1.c -o shm1
u20@ubuntu:~/Test$ gcc shm2.c -o shm2
u20@ubuntu:~/Test$ ./shm1&
[3] 4080
u20@ubuntu:~/Test$ ./shm2&
[4] 4081
u20@ubuntu:~/Test$ abcdefghijklmnopqrst
[3]- Done ./shm1
[4]+ Done ./shm2
```

浮點數轉byte

```
#include <stdio.h>
typedef unsigned char byte;
// 將 double (64bit) 轉成 8 個 byte 存放記憶體空間
// 以利寫入共享記憶體
int main() {
    double k = 17.62538912;
    double *kp;
    //宣告 byte 指標 p 指向 double 變數的記憶體
    byte *p = (byte *)&k;
    // 指標 p, p[0]~p[7]為 double 每一個 byte 的資料
    printf("%x %x %x %x\n", p[0], p[1], p[2], p[3]);
    printf("%x %x %x %x\n", p[4], p[5], p[6], p[7]);
    // 宣告 double 指標指向 p (要轉型)
    kp = (double*) p;
    // 取出 double
    printf("%.10f\n", *kp);
    return 0;
}
```

```
9a ac 59 80
19 a0 31 40
17.6253891200
```

```
// 改寫成 function 版本
#include <stdio.h>
typedef unsigned char byte;
byte * getBytes(double *pValue) {
    byte *p = (byte *)(pValue);
    return p;
}
double getValue(byte *p) {
    double *pValue = (double*) p;
    return *pValue;
}
int main() {
    double k = 17.62538912;
    byte * p = getBytes(&k);
    printf("%x %x %x %x\n", p[0], p[1], p[2], p[3]);
    printf("%x %x %x %x\n", p[4], p[5], p[6], p[7]);
    double v = getValue(p);
    printf("%.10f\n", v);
    return 0;
}
```

共享記憶體傳遞double

```
#include <sys/types.h> // sf1.c
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define SHM_SIZE 50
#define KEY 567
typedef unsigned char byte;
byte * getBytes(double *pValue) {
    byte *p = (byte *)(pValue);
    return p;
}
int main() {
    int shm_id;
    key_t key;
    char* shm, *s;
    byte *p;
    double k = 17.62513782;
    // 命名共享記憶體(shared memory segment) "567".
    // Create the segment.
    if ((shm_id = shmget(KEY, SHM_SIZE, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        printf("creat fail");
        exit(1);
    }

    //attach 共享記憶體到本 process的變數
    if ((shm = shmat(shm_id, NULL, 0)) == (char*) - 1) {
        perror("shmat");
        printf("attach fail");
        exit(1);
    }
    // 寫入資料到共享記憶體，等待其他 process 讀取
    p = getBytes(&k);
    s = shm;
    // 寫入double (8 bit) 到共享記憶體，
    // 等待其他 process 讀取
    for (int i=0; i<8; i++) {
        *s++ = p[i];
    }
    //等待其他 process 修改記憶體第一個字元 '*',
    // 表示已經讀取本 process 寫入資料
    while (*shm != '*') {
        sleep(1);
    }
    return 0;
}
```

共享記憶體傳遞double

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define SHM_SIZE 50
#define KEY 567
typedef unsigned char byte;
double getValue(byte *p) {
    double *pValue = (double*) p;
    return *pValue;
}
int main() {
    int shm_id, size=8;
    char* shm, *s;
    byte * p;
    double value;
    //get the segment named "567", created by the server.
    // Locate the segment.
    if ((shm_id = shmget(KEY, SHM_SIZE, 0666))<0){
        perror("shmget");
        exit(1);
    }
```

```
// attach the segment to our data space.
if ((shm = shmat(shm_id, NULL, 0)) == (char*) - 1) {
    perror("shmat");
    exit(1);
}
// read what the server put in the memory.
s = shm;
for (int i = 0; i<size; i++) {
    p[i]=s[i];
}
value = getValue(p);
printf("\nvalue = %.10f\n", value);
// change the first character of the segment to '*',
// indicating read the segment.
*shm = '*';
return 0;
}
```

```
u20@ubuntu:~/Test$ gcc sf1.c -o sf1
u20@ubuntu:~/Test$ gcc sf2.c -o sf2
u20@ubuntu:~/Test$ ./sf1&
[3] 4373
u20@ubuntu:~/Test$ ./sf2&
[4] 4374
u20@ubuntu:~/Test$
value = 17.6251378200

[3]-  Done                  ./sf1
[4]+  Done                  ./sf2
```

程序操作指令

❑ & 與 [Ctrl]+[z] 背景執行

- 需長時間執行程式，加&或按 Ctrl+z 將程式置於背景執行。
- 提供終端機命令模式同時做許多事情。
- 例如執行 `sudo find "/" -name grep&`，表示尋找 grep 檔案的指令放置背景執行。

```
kjy@ubuntu:~$ sudo find / -name grep&
[2] 2166
kjy@ubuntu:~$ /bin/grep
/snap/core18/1668/bin/grep
/snap/core18/1668/usr/share/doc/grep
/snap/core/8268/bin/grep
/snap/core/8268/usr/share/doc/grep
find: '/run/user/1000/gvfs': Permission denied
/usr/share/doc/grep

[2]+  Exit 1                  sudo find / -name grep
kjy@ubuntu:~$
```


程序操作指令

❑ sleep 500&，執行睡眠500秒

○ [1] 代表指定給該工作的序號

○ 2187 代表 PID (process ID)

```
File Edit View Search Terminal Help
k jy@ubuntu:~$ sleep 300&
[1] 2187
k jy@ubuntu:~$ sleep 500&
[2] 2188
k jy@ubuntu:~$
```

❑ 查詢當前的背景工作可使用 jobs

```
k jy@ubuntu:~$ jobs
[1]-  Running                  sleep 300 &
[2]+  Running                  sleep 500 &
k jy@ubuntu:~$ jobs -l
[1]-  2187 Running              sleep 300 &
[2]+  2188 Running              sleep 500 &
```

❑ fg

○ 將程式叫回前景，沒有背景程式執行，系統顯示無執行中程式。

○ 若背景堆積好幾個命令，可用工作序號挑選

```
k jy@ubuntu:~$ fg %1
sleep 300
^Z
[1]+  Stopped                  sleep 300
```

程序操作指令

- top：對程序執行時間監控；

```
jykuo@ubuntu:~$ top
```

top - 21:56:33 up 3:49, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 217 total, 1 running, 216 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.4 us, 0.3 sy, 0.0 ni, 98.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2018048 total, 127808 free, 777388 used, 1112852 buff/cache
KiB Swap: 2094076 total, 2090480 free, 3596 used. 998820 avail Mem

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3389	root	20	0	201536	4412	3104	S	2.3	0.2	1:31.49	vmtoolsd
925	root	20	0	377252	37096	8320	S	0.3	1.8	0:31.21	Xorg
4765	jykuo	20	0	541148	20068	14296	S	0.3	1.0	0:36.28	vmtoolsd
90323	jykuo	20	0	41800	3704	3052	R	0.3	0.2	0:00.74	top

程序操作指令

□ ps -aux

- 查執行中的程式，可配合參數 -aux 執行
- 列出連同系統服務的程式，輸出第一列會出現 PID，是每個程式執行的編碼。

```
kjy@ubuntu:~$ ps -aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.1	0.2	159828	9152	?	Ss	18:46	0:02	/sbin/init au
root	2	0.0	0.0	0	0	?	S	18:46	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	I<	18:46	0:00	[rcu_gp]
root	4	0.0	0.0	0	0	?	I<	18:46	0:00	[rcu_par_gp]
root	6	0.0	0.0	0	0	?	I<	18:46	0:00	[kworker/0:0H
root	9	0.0	0.0	0	0	?	I<	18:46	0:00	[mm_percpu_wq
root	10	0.0	0.0	0	0	?	S	18:46	0:00	[ksoftirqd/0]
root	11	0.0	0.0	0	0	?	I	18:46	0:00	[rcu_sched]
root	12	0.0	0.0	0	0	?	S	18:46	0:00	[migration/0]
root	13	0.0	0.0	0	0	?	S	18:46	0:00	[idle_inject/

程序操作指令

❑ ps：顯示瞬間程序的狀態，非動態連續監控；

ps 參數

l 長格式輸出；
u 按使用者名和啟動時間順序顯示程序；
j 用任務格式來顯示程序；
f 用樹形格式來顯示程序；
a 顯示所有使用者的所有程序；
x 顯示無控制終端的程序；
r 顯示執行中的程序；
-A 列出所有的程序
-au 顯示較詳細的資訊
-aux 顯示所有包含其他使用者的程序
-e 顯示所有程序,環境變數
-f 全格式
-h 不顯示標題
-l 長格式

欄位

USER: 程序所有者
PID: 程序ID
%CPU: 占用的 CPU 使用率
%MEM: 占用的記憶體使用率
VSZ: 占用的虛擬記憶體大小
RSS: 占用的記憶體大小
TTY: 終端的次要裝置號碼
STAT: 程序狀態:
START: 啟動程序的時間；
TIME: 程序消耗CPU的時間；
COMMAND: 命令的名稱和參數；

程序操作指令

□ ps：顯示瞬間程序的狀態，非動態連續監控；

STAT狀態

D 無法中斷的休眠狀態（通常為 IO 的程序）；

R 正在執行，在可中斷隊列中；

S 處於休眠狀態，靜止狀態；

T 停止或被追蹤，暫停執行；

X 死掉的程序；

Z 僵屍程序不存在但暫時無法刪除；

W: 沒有足夠的記憶體分頁可分配

WCHAN 正在等待的程序資源；

<: 高優先級程序

N: 低優先序程序

L: 有記憶體分頁分配並鎖在記憶體內（即時系統或短 I/O）

s 程序的領導者（在它之下有子程序）；

l 多程序的（使用 CLONE_THREAD, 類似 NPTL pthreads）；

+ 位於後臺的程序組；

程序操作指令

- ps：顯示瞬間程序的狀態，非動態連續監控；

```
jykuo@ubuntu: ~  
jykuo@ubuntu:~$ ps -Al  
F S      UID      PID      PPID      C  PRI   NI   ADDR  SZ  WCHAN    TTY          TIME CMD  
4 S      0         1         0      0   80    0   - 46306 -      ?           00:00:06 systemd  
1 S      0         2         0      0   80    0   -    0 -      ?           00:00:00 kthreadd  
1 S      0         4         2      0   60  -20   -    0 -      ?           00:00:00 kworker/0:0H  
1 S      0         6         2      0   60  -20   -    0 -      ?           00:00:00 mm_percpu_wq  
1 S      0         7         2      0   80    0   -    0 -      ?           00:00:06 ksoftirqd/0  
1 S      0         8         2      0   80    0   -    0 -      ?           00:00:01 rcu_sched  
1 S      0         9         2      0   80    0   -    0 -      ?           00:00:00 rcu_bh
```

```
jykuo@ubuntu: ~  
jykuo@ubuntu:~$ ps -Alf  
F S  UID      PID      PPID      C  PRI   NI   ADDR  SZ  WCHAN    STIME TTY          TIME CMD  
4 S  root         1         0      0   80    0   - 46306 -      18:06 ?           00:00:06 /lib/systemd/sy  
1 S  root         2         0      0   80    0   -    0 -      18:06 ?           00:00:00 [kthreadd]  
1 S  root         4         2      0   60  -20   -    0 -      18:06 ?           00:00:00 [kworker/0:0H]  
1 S  root         6         2      0   60  -20   -    0 -      18:06 ?           00:00:00 [mm_percpu_wq]  
1 S  root         7         2      0   80    0   -    0 -      18:06 ?           00:00:06 [ksoftirqd/0]  
1 S  root         8         2      0   80    0   -    0 -      18:06 ?           00:00:01 [rcu_sched]  
1 S  root         9         2      0   80    0   -    0 -      18:06 ?           00:00:00 [rcu_bh]
```

計算資源監控

❑ glances：顯示動態連續監控計算資源

○ sudo apt install glances

```
ubuntu (Ubuntu 20.04 64bit / Linux 5.15.0-52-generic) - IP 192.168.182.140/24 Pub 60.250.162.107 Uptime: 0:04:31

CPU [ 5.0%] CPU / 5.0% nice: 0.0% ctx_sw: 1K MEM - 31.7% SWAP - 0.0% LOAD 4-core
MEM [ 31.7%] user: 2.7% irq: 0.0% inter: 572 total: 3.80G total: 2.00G 1 min: 0.26
SWAP [ 0.0%] system: 2.1% iowait: 0.0% sw_int: 300 used: 1.20G used: 0 5 min: 0.21
idle: 95.0% steal: 0.0% free: 2.60G free: 2.00G 15 min: 0.10

NETWORK Rx/s Tx/s TASKS 329 (557 thr), 1 run, 209 slp, 119 oth sorted automatically by CPU consumption
ens33 0b 0b
lo 0b 0b

DefaultGateway 4ms

FILE SYS Used Total CPU% MEM% VIRT RES PID USER TIME+ THR NI S R/s W/s Command
/ (sda5) 9.13G 58.3G 0.7 1.4 432M 53.0M 4126 u20 0:01 1 0 R 0 0 /usr/bin/py
0.7 1.7 277M 64.4M 1501 u20 0:02 2 0 S 0 0 /usr/lib/xo
0.7 1.3 805M 50.5M 1992 u20 0:01 5 0 S 0 0 /usr/libexe
0.7 1.1 295M 41.2M 1806 u20 0:01 4 0 S 0 0 /usr/bin/vm
0.3 0.3 320M 11.1M 1472 u20 0:00 3 0 S 0 0 /usr/libexe
0.3 0.0 0 0 1950 root 0:00 1 0 I ? ? [kworker/1:
0.0 1.5 708M 58.2M 1818 u20 0:00 6 0 S 0 0 /usr/libexe
0.0 1.3 430M 50.6M 3065 root 0:01 1 0 S ? ? /usr/bin/py
0.0 1.0 1.05G 39.9M 762 root 0:02 14 0 S ? ? /usr/lib/sn
0.0 0.9 545M 33.9M 1464 u20 0:00 4 0 S 0 0 /usr/libexe
0.0 0.9 281M 33.7M 1657 u20 0:01 4 0 S 0 0 /usr/libexe
0.0 0.8 683M 32.3M 1761 u20 0:00 4 0 S 0 0 /usr/libexe
0.0 0.8 349M 31.5M 1814 u20 0:00 4 0 S 0 0 /usr/libexe
0.0 0.8 421M 31.0M 2060 u20 0:00 4 0 S 4K 2M update-noti
0.0 0.8 422M 30.5M 1764 u20 0:00 4 0 S 0 0 /usr/libexe
0.0 0.8 566M 30.4M 1750 u20 0:00 4 0 S 0 0 /usr/libexe
0.0 0.8 349M 30.0M 1755 u20 0:00 4 0 S 0 0 /usr/libexe
0.0 0.8 205M 29.7M 1661 u20 0:00 3 0 S 0 0 /usr/libexe
0.0 0.8 349M 29.5M 1812 u20 0:00 3 0 S 0 0 /usr/libexe
0.0 0.7 828M 29.1M 1698 u20 0:00 9 0 S 0 0 /usr/libexe
0.0 0.7 738M 28.2M 1710 u20 0:00 6 0 S 0 0 /usr/libexe
```

程序操作指令

□ kill

- 刪除執行中程式，先使用 ps 指令查詢PID
- 當執行 ftp 程式，出現當機時，ps -aux 可查出 ftp 的PID，假設 PID 為 110，輸入：# kill 110，可刪除這個 ftp 程式。

```
kjy@ubuntu:~$ ftp
ftp> quit
kjy@ubuntu:~$ ftp&
[3] 2219
```

```
kjy@ubuntu:~$ ps -aux |grep ftp
kjy      2219  0.0  0.0  27848  2516 pts/0    T   19:04   0:00 ftp
kjy      2222  0.0  0.0  21532  1148 pts/0    S+  19:05   0:00 grep --color=
auto ftp
```

```
kjy@ubuntu:~$ kill -9 2219
kjy@ubuntu:~$ ps -aux |grep ftp
kjy      2231  0.0  0.0  21532  1004 pts/0    S+  19:07   0:00 grep --color=
auto ftp
[3]+  Killed                  ftp
```


程序操作指令

□ kill

kill -STOP [pid]

發送SIGSTOP (17,19,23)停止一個程序，而並不刪除這個程序。

kill -CONT [pid]

發送SIGCONT (19,18,25)重新開始一個停止的程序。

kill -KILL [pid]

發送SIGKILL (9)強迫程序立即停止，並且不實施清理操作。

kill -9 -1

終止擁有的全部程序。

程序操作指令

❑ nohup (no hang up, 不要掛斷)。

- 使用者用 ssh 等指令登入主機後，想要執行某指令，但登出或關掉 ssh，背景執行的工作會跟著消失，因它的父行程被關閉。
- nohup 強制保存背景工作，即便父行程被關閉。

```
kjy@ubuntu:~$ sleep 300&
[1] 2996
kjy@ubuntu:~$ jobs
[1]+  Running                  sleep 300 &
kjy@ubuntu:~$ ps -fc sleep
UID          PID    PPID  C STIME TTY          TIME CMD
kjy           2996     2894  0 19:17 pts/0    00:00:00 sleep 300
```

- 關閉Terminal，重新開啟新的Terminal，jobs是空的

```
kjy@ubuntu:~$ jobs
kjy@ubuntu:~$ ps -fc sleep
UID          PID    PPID  C STIME TTY          TIME CMD
kjy@ubuntu:~$
```

程序操作指令

❑ nohup (no hang up, 不要掛斷)。

○再執行一次 nohup sleep 300& ,

```
kjy@ubuntu:~$ nohup sleep 300&
[1] 3056
kjy@ubuntu:~$ nohup: ignoring input and appending output to 'nohup.out'

kjy@ubuntu:~$ jobs
[1]+  Running                  nohup sleep 300 &
kjy@ubuntu:~$ ps -fc sleep
UID          PID    PPID  C STIME TTY          TIME CMD
kjy          3056    3028  0 19:19 pts/0      00:00:00 sleep 300
```

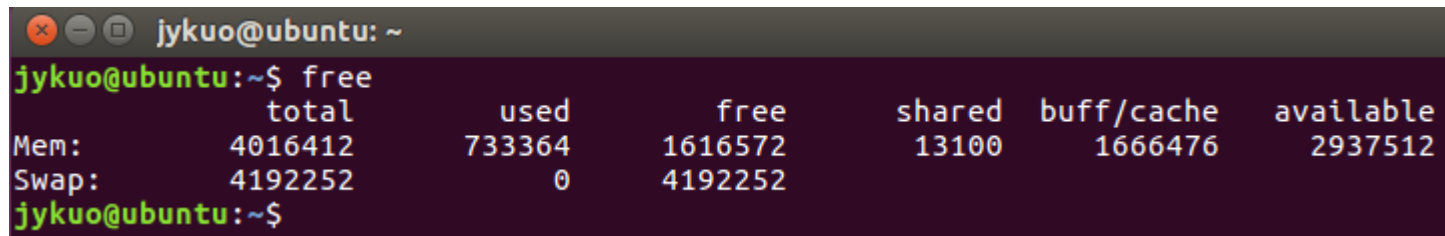
○關閉Terminal，重新開啟新的Terminal，仍然有sleep 300

```
kjy@ubuntu:~$ jobs
kjy@ubuntu:~$ ps -fc sleep
UID          PID    PPID  C STIME TTY          TIME CMD
kjy          3056    2335  0 19:19 ?          00:00:00 sleep 300
```

程序操作指令

❑ free

- 查看記憶體使用狀況



A terminal window titled 'jykuo@ubuntu: ~' showing the output of the 'free' command. The output is a table with 7 columns: total, used, free, shared, buff/cache, and available. The rows are for Mem: and Swap:.

	total	used	free	shared	buff/cache	available
Mem:	4016412	733364	1616572	13100	1666476	2937512
Swap:	4192252	0	4192252			

程序操作指令

❑ exit

- 離開 Linux 系統，相當於 login out。

❑ shutdown

- 關機，只有 root 有權限

- # shutdown <==系統在兩分鐘後關機，並傳送訊息給在線上的人
- # shutdown -h now <==系統立刻關機
- # shutdown -r now <==系統立刻重新開機
- # shutdown -h 20:30 <==系統在今天 20:30 關機
- # shutdown -h +10 <==系統在 10 分鐘後關機

❑ reboot

- 重新開機，可以配合寫入緩衝資料的 sync 指令動作，如下：
- # sync; sync; sync; reboot

Thread執行緒

- ❑ 單一執行緒的 Process 有一個PC (program counter)
- ❑ 多執行緒的Process有多個PC，每一個指向一個執行緒要執行的下一個指令。
 - 多執行緒可以**利用多核心CPU**平行執行。
 - 每個執行緒具有: ID, PC, 暫存器組、stack
 - 同一個Process的所有執行緒，**共享被分配的記憶體、code, data, file, OS signal**。
 - OS造一個執行緒比造一個Process較經濟有效率。

Thread執行緒

```
int pthread_create(pthread_t *thread,  
                  const pthread_attr_t *attr,  
                  void *(*start_routine) (void *),  
                  void *arg);
```

- `pthread_t *thread`： `pthread_t` 是執行緒的資料型別。
- `const pthread_attr_t *attr`：設定呼叫策略、能使用的記憶體大小等。大部分設為 `NULL`。
- 3) `void *(*start_routine) (void *)`：新建執行緒的函數，該函數的參數最多有 1 個（可以省略不寫），參數和回傳值類型須為 `void*`。若該有回傳值，由 `pthread_join()`接收。
- `void *arg`：指定傳遞給 `start_routine` 函數的參數，不需資料時，設為 `NULL`。
- 成功創建執行緒，`pthread_create()` 回傳 0，反之返回非零。

Thread執行緒

□ 多執行緒的Process

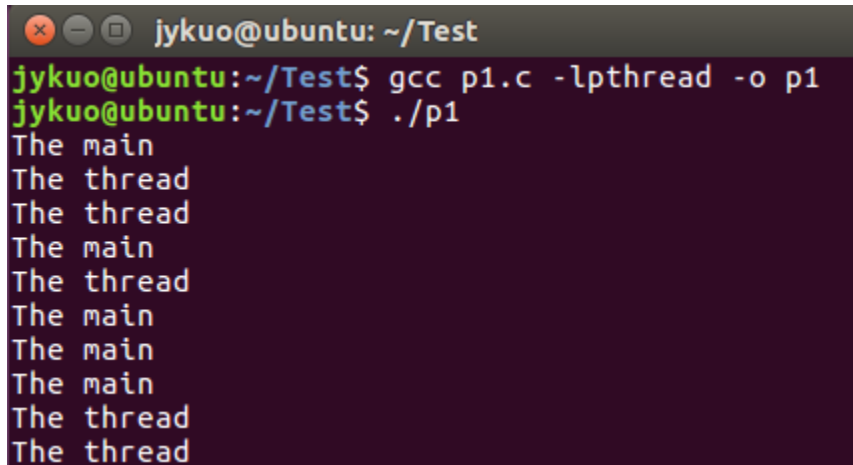
```
// p1.c      gcc p1.c -lpthread -o p1
//          ./p1
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
void message(char *s) {
    for(int i=0;i<5;i++) {
        printf("%s", s);
        sleep(rand()%3);    //單位秒
        // usleep(rand()%3); 單位微秒
    }
}
void thread(void) {
    message("The thread\n");
    pthread_exit(NULL); // 離開子執行緒
}
```

```
int main(){
    int i, p;
    time_t t;
    pthread_t id;
    srand((unsigned) time(&t));
    // 建立子執行緒
    p = pthread_create(&id, NULL, (void *) thread, NULL);
    if(p!=0){
        printf ("Create pthread error!\n");
        exit(1);
    }
    message("The main\n");
    pthread_join(id, NULL); // 等待子執行緒執行完成
    return 0;
}
```


Thread執行緒

❑ 多執行緒的Process，編譯執行

- gcc p1.c -lpthread -o p1
- ./p1



```
jykuo@ubuntu: ~/Test
jykuo@ubuntu:~/Test$ gcc p1.c -lpthread -o p1
jykuo@ubuntu:~/Test$ ./p1
The main
The thread
The thread
The main
The thread
The main
The main
The main
The thread
The thread
```

A terminal window with a dark purple background. The title bar shows window control icons and the text 'jykuo@ubuntu: ~/Test'. The terminal displays the compilation of 'p1.c' using 'gcc' with the '-lpthread' flag, followed by the execution of './p1'. The output shows interleaved messages from the main thread and several spawned threads, demonstrating concurrent execution.

Exercise Thread執行緒

❑ 多執行緒的Process

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
double avg;
int min, max, size;
void* calculateAverage(void* data){
    int* input = (int*) data;
    int i, sum = 0;
    for (i = 0; i < size; i++)
        sum += input[i];
    avg = (double)sum / size;
}
void* calculateMaximum(void* data){
    int* input = (int*) data;
    max = input[0];
    for (int i = 0; i < size; i++)
        if(input[i] > max)
            max = input[i];
}
```

```
void* calculateMinimum(void* data){
    int* input = (int*) data;
    min = input[0];
    for (int i = 0; i < size; i++)
        if(input[i] < min)
            min = input[i];
}
int main(int argc, char *argv[]){
    int i;
    int data[argc - 1];
    int t1, t2, t3;
    pthread_t thread1, thread2, thread3;
    if(argc <= 1) {
        printf("Incorrect. Please enter more integers.\n");
        exit(0);
    }
    for (i = 0; i < (argc - 1); i++) {
        data[i] = atoi(argv[i + 1]);
        size++;
    }
```

Exercise Thread執行緒

□ 多執行緒的Process

```
t1 = pthread_create(&thread1, NULL, (void*) calculateAverage, (void*) data);
if(t1) {
    fprintf(stderr, "Error creating thread(calculateAverage), return code: %d\n", t1);
    exit(EXIT_FAILURE);
}
t2 = pthread_create(&thread2, NULL, (void*) calculateMinimum, (void*) data);
if(t2) {
    fprintf(stderr, "Error creating thread(calculateMinimum), return code: %d\n", t2);
    exit(EXIT_FAILURE);
}
t3 = pthread_create(&thread3, NULL, (void*) calculateMaximum, (void*) data);
if(t3) {
    fprintf(stderr, "Error creating thread(calculateMaximum), return code: %d\n", t3);
    exit(EXIT_FAILURE);
}
pthread_join(thread1, NULL); pthread_join(thread2, NULL); pthread_join(thread3, NULL);
printf("The average : %f\n", avg); printf("The minimum : %d\n", min);
printf("The maximum : %d\n", max);
exit(EXIT_SUCCESS);
}
```

Exercise Process

□ 建置子程序(Process),

- 子程序(Process)分配計算，例如 31~60
- 父程序(Process)分配計算，例如 1~30和加總，再*4
- (計算 PI 精確到小數N位，即輸入N)

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \frac{1}{15} \dots$$

□ 與沒有使用子程序(Process)比較執行時間

Exercise Thread

- ❑ 建置3個Thread,
 - thread分配計算3~22, 23~42, 43~62
 - main 做加總，再*4
 - (計算 PI 精確到小數N位)

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \frac{1}{15} \dots$$

- ❑ 與沒有使用thread比較執行時間

程式執行時間

□ 計算程式執行時間

```
#include <stdio.h>
#include <time.h>
long g(int n) {
    if (n<1) return 1;
    else return (g(n-1)+g(n-2)+g(n-3));
}
int main() {
    long begin, end;
    begin = clock();
    g(32); g(32);
    end = clock();
    printf("%ld ms\n", (end-begin)*100/CLOCKS_PER_SEC); //毫秒
    return 0;
}
```

fork() 程式執行時間

❑ fork

- 計算程式執行時間
- clock() 會被歸0

```
struct timespec {  
    time_t  tv_sec;      /* seconds */  
    long    tv_nsec;     /* nanoseconds */  
};
```

```
#include <stdio.h>  
#include <time.h>  
long g(int n) {  
    if (n<1) return 1;  
    else return (g(n-1)+g(n-2)+g(n-3));  
}  
int main() {  
    struct timespec st = {0, 0};  
    struct timespec et = {0, 0};  
    clock_gettime(CLOCK_REALTIME, &st);  
    g(32); g(32);  
    clock_gettime(CLOCK_REALTIME, &et);  
    printf("%ld ms\n", (et.tv_sec-st.tv_sec)*1000+(et.tv_nsec-st.tv_nsec)/1000000); //毫秒  
    return 0;  
}
```

Exercise 產生多個process

```
#include <stdio.h>      // nf.c
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main() {
    pid_t child[2], self = getpid(), p;
    child[0] = fork();
    if (child[0]>0) child[1] = fork();
    if (child[0]>0 && child[1]>0) {
        sleep(10);
        printf("\nParent process, wait for child...\n");
        p = wait(NULL);           //等待child程序
        printf("child's pid1 =%d, pid2=%d, p=%d\n", child[0], child[1], p);
        printf("parent pid =%d\n", self);
    }
    else
        sleep(10);
    printf("child process pid =%d\n", getpid());
    return 0;
}
```


Exercise 產生多個process 計算PI

```
#include <stdio.h>      // nf.c
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <time.h>
#define SHM_SIZE 50
#define KEY 567
typedef unsigned char byte;
byte * getBytes(double *pValue) {
    byte *p = (byte *) (pValue);
    return p;
}
void writeSHM(double magic, int loc) {
    int shm_id;
    char* shm, *s;
    byte *p;
    double k = magic;
    // 共享記憶體(shared memory segment) "567".
    if ((shm_id = shmget(KEY, SHM_SIZE, 0666)) < 0) {
        printf("get fail");
        exit(1);
    }
    //attach 共享記憶體到本 process的變數
    if ((shm = shmat(shm_id, NULL, 0)) == (char*) - 1) {
        printf("attach fail");
        exit(1);
    }
}
```

Exercise 產生多個process 計算PI

```
// 寫入資料到共享記憶體，等待其他 process 讀取
p = getBytes(&k);
s = shm+loc*8;
// 寫入double到共享記憶體，等待parent process 讀取
//printf("\n %f \n", magic);
for (int i=0; i<8; i++) {
    *s++ = p[i];
    printf("%x", p[i]);
}
}
void getResult() {
    int shm_id;
    char* shm, *s;
    byte * p;
    double value;
    //get the segment named "567", created by the server.
    if ((shm_id = shmget(KEY, SHM_SIZE, 0666))<0){
        printf("get fail");
        exit(1);
    }
    // attach the segment to our data space.
    if ((shm = shmat(shm_id, NULL, 0)) == (char*) - 1) {
        printf("attach fail");
        exit(1);
    }
}
```

Exercise 產生多個process 計算PI

```
// read what the server put in the memory.
s = shm;
for (int i = 0; i<8; i++) {
    p[i]=s[i];
    printf("%x-", p[i]);
}
printf("\n==>value = %.16f\n", *((double*)p));
for (int i = 0; i<8; i++) {
    p[i]=s[i+8];
    printf("%x-", p[i]);
}
printf("\n==>value = %.16f\n", *((double*)p));
}

double compute(int loc) {
    double c = 1000000000;
    double r = 0.0, start=0.0, end=0.0;
    int sign = 1;
    if (loc == 0 || loc == 1) {
        start = c*loc;
        end = c + start;
    }
    else if (loc==2) {
        start = 0.0;
        end = c*2;
    }
    for (double i=start;i<end; i++) {
        r = r + sign/(1+2*i);
        sign = (-1)*sign;
    }
    return 4*r;
}
```

Exercise 產生多個process 計算PI

```
int main() {
    int shm_id;
    double r = 0;
    pid_t wpid;
    pid_t child[2] = {-1, -1}, self = getpid();
    struct timespec st = {0, 0};
    struct timespec et = {0, 0};
    clock_gettime(CLOCK_REALTIME, &st);
    if ((shm_id = shmget(KEY, SHM_SIZE, IPC_CREAT | 0666)) < 0) {
        printf("creat fail");
        exit(1);
    }
    child[0] = fork();
    if (child[0] > 0) child[1] = fork();
    if (child[0] > 0 && child[1] > 0) {
        //printf("===>%.16f\n", compute(1));
        printf("parent pid = %d\n", self);
        printf("Parent process, wait for child...\n");
        while ((wpid = wait(NULL)) > 0);          //等待child程序
        getResult();
        r = compute(2);                          //
        printf("\nParent process, wait for child end.\n");
        clock_gettime(CLOCK_REALTIME, &et);
        printf("%ld ms\n", (et.tv_sec - st.tv_sec) * 1000 + (et.tv_nsec - st.tv_nsec) / 1000000);
    }
}
```

Exercise 產生多個process 計算PI

```
else {  
    //sleep(1);  
    if (child[0]==0) {  
        r = 1;  
        //r = compute(0);  
        writeSHM(r,0);  
        printf("\n0:child process pid =%d\n", getpid());  
    }  
    else if (child[1]==0) {  
        r = 1;  
        //r = compute(1);  
        writeSHM(r,1);  
        printf("\n1:child process pid =%d\n", getpid());  
    }  
}  
return 0;  
}
```

Exercise Process

- ❑ 建置N個子程序(Process), $2 < N < 6$
 - 輸入C，計算 PI 精確到小數C位
 - 子程序(Process)負責計算，例如 31~60
 - 父程序(Process)分配計算，加總

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \frac{1}{15} \dots$$

- ❑ 與沒有使用子程序(Process)比較執行時間