

Linux Process

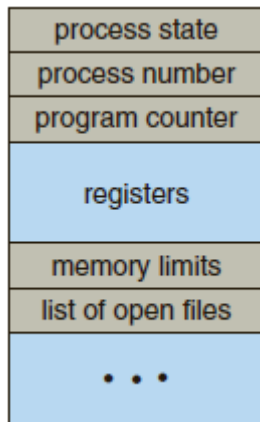
國立臺北科技大學資訊工程系
郭忠義教授

jykuo@ntut.edu.tw

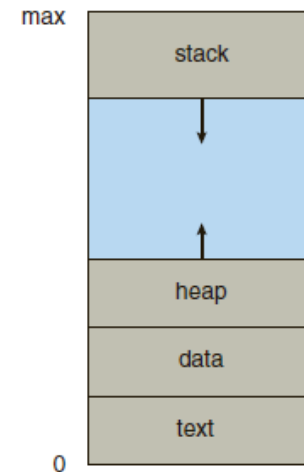
Process

□ 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

Process

❑ 程序(Process) – fork()

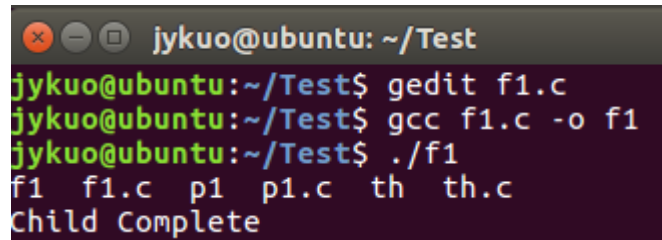
```
#include <sys/types.h>          // f1.c
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
int main() {
    pid_t pid;
    pid = fork();    //fork a child process
    if (pid < 0) {    // error occurred
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 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

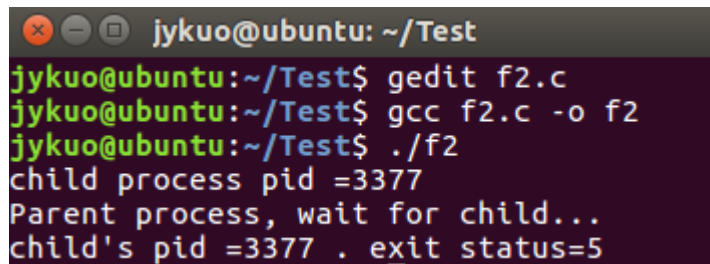
```
#include <stdio.h>      // f2.c
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main() {
    pid_t pid;
    int status, i;
    if(fork() == 0) {
        printf("child process pid =%d\n", getpid());
        exit(5);
    }
    else {
        sleep(1);
        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>      // 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());
        // 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
```


程序操作指令

□ & 與 [Ctrl]+[z] 背景執行

- 需長時執行程式，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
```

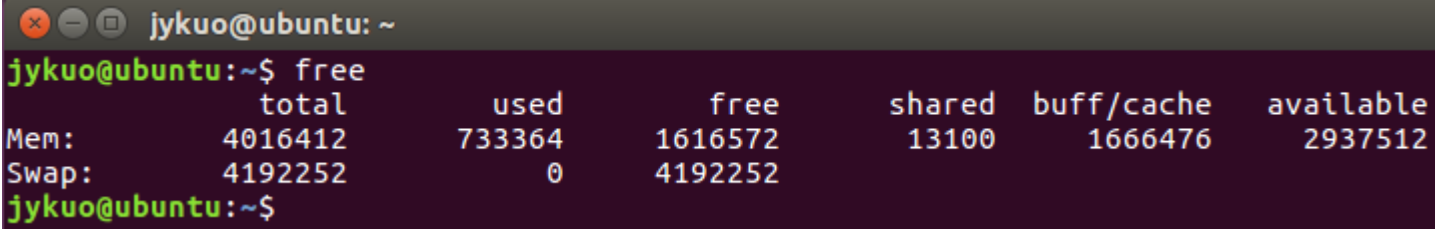
○關閉Terminal，重新開啟新的Terminal，仍然有sleep 300

```
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



```
jykuo@ubuntu: ~  
jykuo@ubuntu:~$ free  
              total        used        free      shared  buff/cache   available  
Mem:          4016412       733364       1616572        13100       1666476       2937512  
Swap:          4192252           0        4192252  
jykuo@ubuntu:~$
```

The screenshot shows a terminal window with the title 'jykuo@ubuntu: ~'. The user has entered the command 'free'. The output displays memory usage statistics in a table format. The first row shows the total, used, free, shared, buff/cache, and available memory. The second row shows the same statistics for memory (Mem). The third row shows the same statistics for swap space (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
}
```

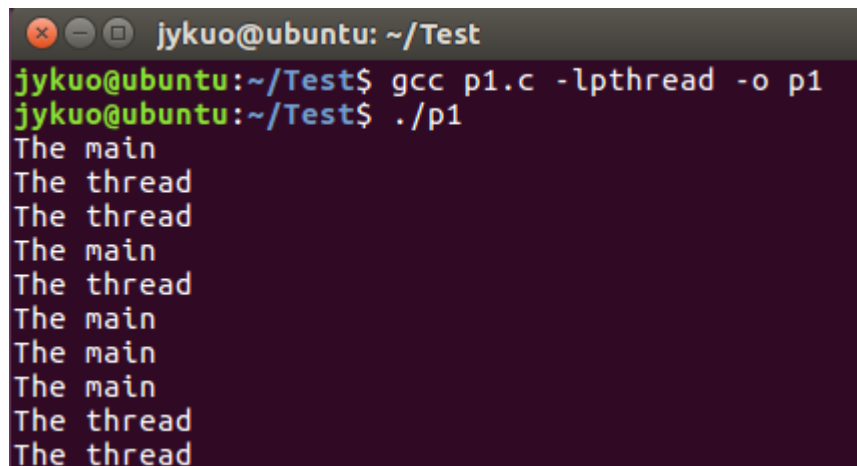
```
int main(){
    int i, p;
    time_t t;
    pthread_t id;
    srand((unsigned) time(&t));
    // 建立子執行緒
    p = pthread_create(&id, NULL, (void *) thread, NULL);
    if(p!=0){
        printf ("Create pthread error!\n");
        exit(1);
    }
    message("The main\n");
    pthread_join(id, NULL);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

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* calculateMinimum(void* data){
    int* input = (int*) data;
    max = input[0];
    for (int i = 0; i < size; i++)
        if(input[i] > max)
            max = input[i];
}
```

```
void* calculateMaximum(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 Thread執行緒

□ 建置子程序,

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

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

□ 與沒有使用子程序比較執行時間

Exercise Thread執行緒

- ❑ 建置3個Thread,
 - thread分配計算3~12, 13~22, 23~32
 - 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("%d ms\n", (end-begin)*100/CLK_TCK); //毫秒
    return 0;
}
```