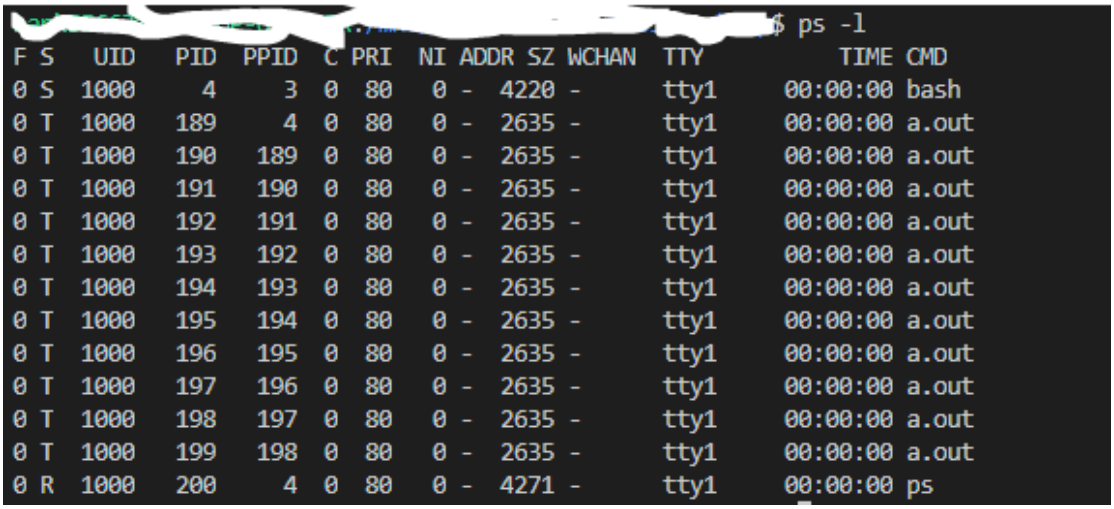
Project #2. Process creation cleanup and IPC

Process creation and cleanup

Process 意旨已經執行並且 load 進 memory 的 program，程式中每一行程式碼隨時可能會被 CPU 執行。

在 linux 中我們可以使用 ps -l，觀察現在在系統中的 process。



```
$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000     4     3  0  80   0 -  4220 -      tty1     00:00:00 bash
0 T  1000   189     4  0  80   0 -  2635 -      tty1     00:00:00 a.out
0 T  1000   190   189  0  80   0 -  2635 -      tty1     00:00:00 a.out
0 T  1000   191   190  0  80   0 -  2635 -      tty1     00:00:00 a.out
0 T  1000   192   191  0  80   0 -  2635 -      tty1     00:00:00 a.out
0 T  1000   193   192  0  80   0 -  2635 -      tty1     00:00:00 a.out
0 T  1000   194   193  0  80   0 -  2635 -      tty1     00:00:00 a.out
0 T  1000   195   194  0  80   0 -  2635 -      tty1     00:00:00 a.out
0 T  1000   196   195  0  80   0 -  2635 -      tty1     00:00:00 a.out
0 T  1000   197   196  0  80   0 -  2635 -      tty1     00:00:00 a.out
0 T  1000   198   197  0  80   0 -  2635 -      tty1     00:00:00 a.out
0 T  1000   199   198  0  80   0 -  2635 -      tty1     00:00:00 a.out
0 R  1000   200     4  0  80   0 -  4271 -      tty1     00:00:00 ps
```

PID:該 process 的 ID。

PPID:該 process 的 parent ID。

PGID:上面未顯示，為該 process 的 process group ID，在 Linux 中可使用，ps xao pgid,comm 顯示。

接下來介紹一些有關 Process creation 的 function。

pid_t fork(void);：標頭檔<unistd.h>，用於複製出一個 child process，執行之後會回傳一個整數值，以下是個數值代表意義：

- 負值(<0)：建立 child process 失敗。
- 零(=0)：代表現在位於新建立的 child process 中。
- 正值(>0)：代表這個程式位於原本的 process 中，該值為創造出來的 child process 的 PID。

pid_t wait(int *status);：標頭檔<sys/types.h>/<sys/wait.h>，會暫停目前 process 的執行，直到有信號來或是某一個 child process 結束，如在執行 wait()時 child process 已結束，則 wait()會立即返回 child process 結束狀態值，結束狀態會由參數 status 返回，而 child process 的 PID 會藉由 return value 返回。

pid_t waitpid(pid_t pid, int *status, int options)：
標頭檔<sys/types.h> /<sys/wait.h>，跟 wait 作用類似，但可指定等待任一個 process 結束，wait()相當於 waitpid()的特例，

參數 pid 數值意義如下：

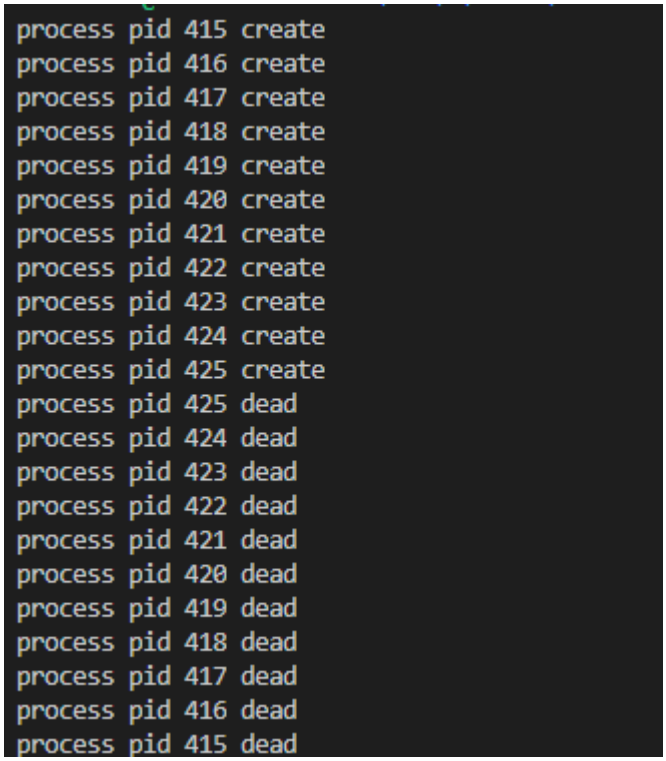
- pid<-1 等待 process pid 為 pid 絕對值的任何 child process.

- pid=-1 等待任何 child process, 相當於 wait().
- pid=0 等待 PGID 與目前 process 相同的任何 child process.
- pid>0 等待任何 child process pid 為 pid 的 child process

參數 option 可以為 0 或下面的 OR 組合:

- WNOHANG 如果沒有任何已經結束的 child process 則馬上返回, 不予以等待。
- WUNTRACED 如果 child process 進入暫停執行情況則馬上返回,但結束狀態不予以理會。
- 作業 1:

請 create 出 10 個 child process，並確保每個 process 在他所有 child process 結束後，才結束，請以在 terminal 中輸出每個 process 何時 create，何時 dead，以方便批改，如下圖。



- 作業 2:

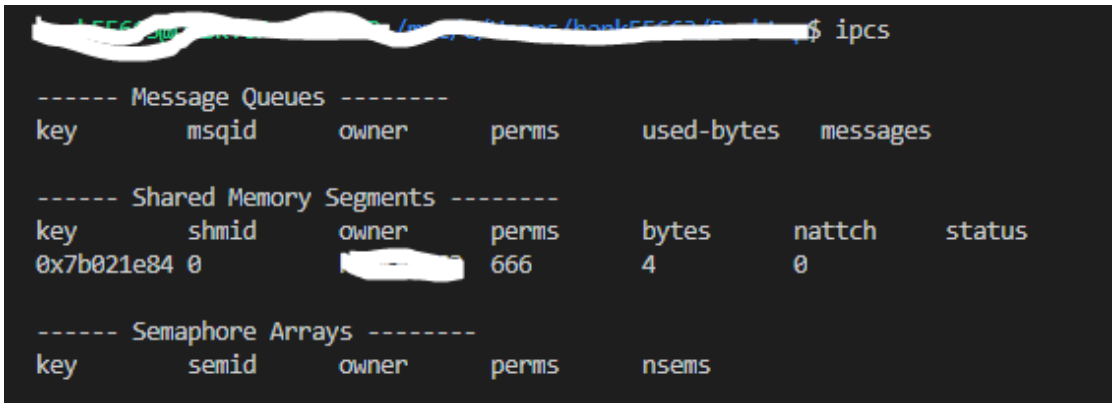
請參考 process_2.cpp，請修改成確保每個 process 在他所有 child process 結束後才結束，並輸出該 process 所 create 的 child process pid，0 視為沒有，以下為一個合法的輸出。

```
create main process 444
process 444 create process 445
process 445 create process 0
process 444 create process 446
process 446 create process 0
process 445 create process 447
process 447 create process 0
process 444 create process 448
process 446 create process 449
process 448 create process 0
process 449 create process 0
process 448 deaded its child process 0 0 0
process 449 deaded its child process 0 0 0
process 446 deaded its child process 0 0 449
process 447 create process 451
process 445 create process 450
process 451 create process 0
process 450 create process 0
process 451 deaded its child process 0 0 0
process 450 deaded its child process 0 0 0
process 447 deaded its child process 0 0 451
process 445 deaded its child process 0 447 450
process 444 deaded its child process 445 446 448
```

IPC

IPC 全名為 Interprocess Communication，意旨跨 process 的溝通，有許多種方式，例如使用 Clipboard、Pipes、File Mapping、shared memory 等，這次主要需要你們練習 shared memory.

在 linux 中我們可以使用 ipcs 這個指令去觀察 shared memory 的狀態，如下圖：



而 shared memory 主要會用到以下 function:

- int shmget(key_t key, size_t size, int shmflg);
  - key 為 process 間事先約定的 key。
  - size 為 shared memory 的大小，當創建一個新的 shared memory segment 時，size 必須大於 0，若是訪問一個已經存在的 shared memory segment，size 可為 0。
  - shmflg: IPC_CREATE、IPC_EXCL

        - IPC_CREATE：調用 shmget 時，系統將此值與其他 shared memory segments 的 key 進行比較。如果存在相同的 key ，說明 shared memory segment 已存在，此時返回該 shared memory segment 的 shmid；如果不存在，則新建一個 shared memory segment 並返回其 shmid。
        - IPC_EXCL：必須和 IPC_CREATE 一起使用，否則沒意義。當 shmflg 取 IPC_CREATE | IPC_EXCL 時，表示如果發現 shared memory segment 已經存在則返回-1，錯誤代碼為 EEXIST 。
    - 返回值：成功返回 shmid；失敗返回 -1
- void *shmat(int shmid, const void *shmaddr, int shmflg);
    - shmid：shmget 的返回值
    - shmaddr：shared memory 連接到本 process 後在本 process address space 的 memory address。如果為 NULL，則由 kernel 選擇一塊空閒的 address。如果非空，則由該參數指定 address。一般傳入 NULL。
    - shmflg：一般為 0 ，則不設置任何限制權限。
        - SHM_RDONLY：只讀
        - SHM_RND：如果指定了 shmaddr，那麼設置該標誌位後，連接的內存地址會對 SHMLBA 向下取整對齊。
- 返回值：成功後返回一個指向 shared memory 的 pointer；否則返回 (void *) -1
- int shmdt(const void *shmaddr);
    - shmaddr：shmat 函數的返回值
    - 返回值：成功返回 0；失敗返回 -1
- 調用 shmdt 後，shared memory 的 shmid_ds 結構會變化。其中 shm_dtime 指向當前時間，shm_lpid 設置為當前 pid，shm_nattch 減一。如果 shm_nattch 變為 0，並且該 shared memory 已被標記為刪除，那麼該 shared memory 會被刪除。(也就是說，光調用 shmdt，但是沒調用刪除，就不會刪除。並且，只調用刪除，沒調用 shmdt 使得 nattch=0，也不會刪除）
- int shmctl(int shmid, int cmd, struct shmid_ds *buf);
    - shmid：shmget 的返回值
    - cmd：
        - IPC_STAT：把 shmid_ds 從 kernel 中複製到使用者的 buffer
        - IPC_RMID 標記刪除。只有當 nattch=0 時才會真正刪除。
        - IPC_INFO 返回系統範圍內的 shared memory 設置
- 如未使用 shmdt 及 shmctl 讓 shared memory 刪除，則會讓此 shared memory 持續存在在系統中。
- 有關其他的訊息可以去參考<sys/shm.h>及<sys/ipc.h>裡面會有更詳細的結構。

- 作業 3:
  - 請你寫一隻程式跟助教的程式互動玩猜拳遊戲，總共會進行 100 回合，每一回合獲勝即獲得 1 分，剛開始助教的程式會傳送一個 shared memory 的 key 給你，請你用 standard input 讀入(cin or scanf)，之後每次猜拳前，助教都會先將該回合要出的拳以一個 int 代表放入此 key 所創建的 shared memory，0 為 Paper(布)，1 為 Scissor(剪刀)，2 為 Stone(石頭)，當助教決定好要出甚麼時，會輸出"OK"，之後換你出拳，你出完拳後，助教會輸出助教程式所出的拳，保證與 shared memory 中所代表的拳相同，你可以用來判斷你的分數，請盡你所能的打敗助教吧，出拳請使用 standard output(cout or printf)每次 output 結束請使用(fflush(stdout) or cout.flush())來清空 buffer，否則可能造成不可預期結果。
  - Os_fake_sol.cpp 為一個能正常跟助教玩猜拳遊戲的程式。
  - Os_fake_judge.cpp 為一個只會出布的程式，其餘操作與助教程式一模一樣，也會將布所代表的數字 0 存於 shared memory 中，可用於協助你完成作業，請將 ftok 函數裡的第一個字串換成任一個你電腦裡存在的文件。
  - interactive.sh 為一個互動的 script，使用方法為先將你的程式及 Os_fake_judge.cpp 編譯成可執行檔，接著使用下列指令 ./interactive.sh ./{Os_fake_judge.cpp 的執行檔} ./{your solution 的執行檔}，最後顯示出來的即為你的得分。