Name: Phạm Nguyễn Hải Anh

ID: 21520586

Class: IT007.N21.ANTN

OPERATING SYSTEM
LAB 3'S REPORT

SUMMARY

Task		Status	Page
Section 3.5	Bài 1	Hoàn thành	2
	Bài 2	Hoàn thành	3
	Bài 3	Hoàn thành	4
	Bài 4	Hoàn thành	15
…	…		
	…		

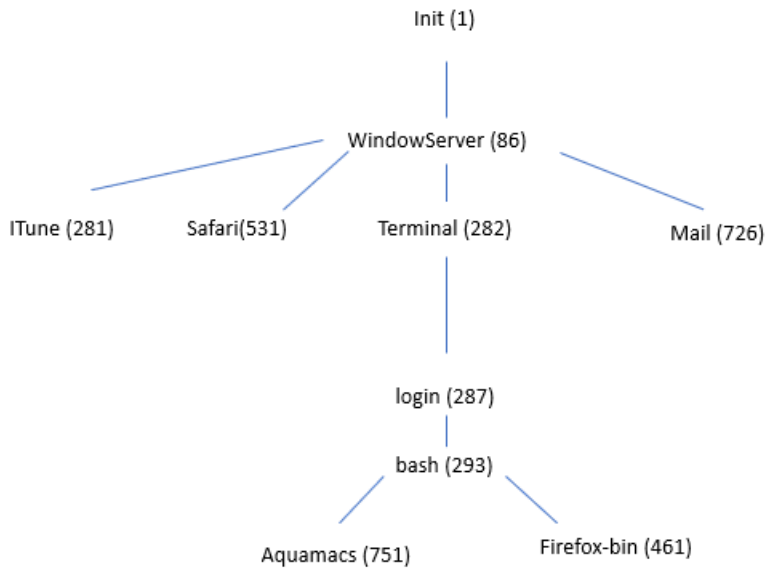
Self-scrores: 9.5

Note: Export file to **PDF and name the file by following format:*
Student ID_LABx.pdf

Section 1.5

1. Mối quan hệ cha-con giữa các tiến trình

a. Vẽ cây quan hệ parent-child của các tiến trình bên dưới:



b. Trình bày cách sử dụng lệnh ps để tìm tiến trình cha của một một tiến trình dựa vào PID của nó

Lệnh để thực hiện yêu cầu nây sẽ là:

ps -f $(ps -o ppid= <child-process-id>)

Giải thích:

- ps -f <process-id>: in ra tiến trình có <process-id> đang chạy ở dạng chi tiết (PID, PPID, TIME, …)

- ps -o ppid= <child-process-id>: sẽ in ra ppid của tiến trình có pid = <child-process-id>, nói cách khác là in ra pid của tiến trình cha khi cho trước pid tiến trình con,

- Cho lệnh thứ 2 vào dấu $(), ppid thay vì xuất ra màn hình sẽ trở thành tham số cho option -f của ps. Từ đó ps -f sẽ xuất ra thông tin của tiến trình cha.

```
pngha2@pngha3:~$ ps
    PID TTY          TIME CMD
  84717 pts/0    00:00:00 bash
  98795 pts/0    00:00:00 ps
pngha2@pngha3:~$ ps -f 84717
UID          PID    PPID  C STIME TTY      STAT   TIME CMD
pngha2     84717   84716  0 18:29 pts/0    Ss     0:00 -bash
pngha2@pngha3:~$ ps -f $(ps -o ppid= 84717)
UID          PID    PPID  C STIME TTY      STAT   TIME CMD
pngha2     84716   84568  0 18:29 ?        S      0:00 sshd: pngha2@pts/0
```

c. Tìm hiểu và cài đặt lệnh pstree (nếu chưa được cài đặt), sau đó trình bày cách sử dụng lệnh này để tìm tiến trình cha của một tiến trình dựa vào PID của nó.

Dùng lệnh man pstree để tìm hiểu cách sử dụng lệnh pstree, có 2 option hữu ích:

- p: hiển thị pid của 1 tiến trình và các pid của tiến trình con của nó
- s: hiển thị những tiến trình cha của 1 tiến trình con

```
  -p     Show PIDs.  PIDs are shown as decimal numbers in parentheses after each process  name.   -p  implicitly
         disables compaction.

  -s     Show parent processes of the specified process.
```

Vậy lệnh để thực hiện yêu cầu c là: ps -p -s <procees id>

Giả sử chọn pid = 1963. Câu lệnh ps -p -s 1963 sẽ in ra các tiến trình cha của tiến trình 1963.

```
pngha2@pngha3:~$ pstree -p -s 1963
systemd(1)───containerd-shim(1304)───entrypoint.sh(1404)───kubelet(1963)─┬─{kubelet}(1973)
                                                                         ├─{kubelet}(1974)
                                                                         ├─{kubelet}(1975)
                                                                         ├─{kubelet}(1976)
                                                                         ├─{kubelet}(1977)
                                                                         ├─{kubelet}(1978)
                                                                         ├─{kubelet}(1979)
                                                                         ├─{kubelet}(1980)
                                                                         ├─{kubelet}(1990)
                                                                         ├─{kubelet}(2006)
                                                                         ├─{kubelet}(2008)
                                                                         ├─{kubelet}(2010)
                                                                         └─{kubelet}(11477)
```

2. Chương trình bên dưới in ra kết quả gì? Giải thích tại sao?

Trên lý thuyết, chương trình sẽ in ra "I see 17 coconuts!". Giải thích: Chương trình tạo ra 2 tiến trình, 1 cha 1 con. Khi tiến trình con thực thi (vế if), biến num_coconuts trong tiến trình con thay đổi thành 42, rồi exit, giải phóng giá trị đã đổi trong biến num_coconuts. Còn tiến trình cha (vế else) ngồi đợi tiến trình con thực thi, rồi in ra giá trị biến num_coconuts. Lúc này giá trị đã được trả về trước khi vào tiến trình con, là 17.

Chương trình:

```c
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
int main(){
        pid_t pid;
        int num_coconuts = 17;
        pid = fork();
        if(pid == 0) {
                num_coconuts = 42;
                exit(0);
        } else {
                wait(NULL); /*wait until the child terminates */
        }
        printf("I see %d coconuts!\n", num_coconuts);
        exit(0);
}
```

Chạy chương trình:

```
pngha2@pngha3:~$ gcc -o bt2 bt2.c
pngha2@pngha3:~$ ./bt2
I see 17 coconuts!
```

3. **Trong phần thực hành, các ví dụ chỉ sử dụng thuộc tính mặc định của pthread, hãy tìm hiểu POSIX thread và trình bày tất cả các hàm được sử dụng để làm thay đổi thuộc tính của pthread, sau đó viết các chương trình minh họa tác động của các thuộc tính này và chú thích đầy đủ cách sử dụng hàm này trong chương trình. (Gợi ý các hàm liên quan đến thuộc tính của pthread đều bắt đầu bởi: pthread_attr_*)**

Các hàm thay đổi thuộc tính của pthread: (nguồn: http://www.qnx.com/developers/docs/qnxcar2/index.jsp?topic=%2Fcom.qnx.doc.neutrino.getting_started%2Ftopic%2Fs1_procs_thread_attr.html)

- pthread_attr_init (pthread_attr_t *attr). - pthread_attr_destroy(pthread_attr_t *attr).

- pthread_attr_setdetachstate(pthread_attr_t *attr , int detachstate).

- pthread_attr_getdetachstate(const pthread_attr_t *attr , int *detachstate).

- pthread_attr_getguardsize(const pthread_attr_t * restrict attr, size_t * restrict guardsize).

- pthread_attr_setguardsize(pthread_attr_t * attr, size_t).

- pthread_attr_getinheritsched(const pthread_attr_t * restrict attr, int * restrict inheritsched).

- pthread_attr_setinheritsched(pthread_attr_t * attr, int inheritsched).

- pthread_attr_getschedparam(const pthread_attr_t* restrict attr, struct sched_param *restrict param).

- pthread_attr_setschedparam(pthread_attr_t * attr , const struct ched_param *param).

- pthread_attr_getschedpolicy(const pthread_attr_t* restrict attr, int restrict policy).

- pthread_attr_setschedpolicy(pthread_attr_t *attr , int policy).

- pthread_attr_getscope(const pthread_attr_t *restrict attr, int * restrict contentionscope).

- pthread_attr_setscope(pthread_attr_t *attr , int contentionscope).

- pthread_attr_getstack(const pthread _attr_t *restrict attr, void **restrict stackaddr, size_t stacksize).

- pthread_attr_setstack(pthread _attr_t *restrict attr, void *stackaddr, size_t stacksize).

- pthread_attr_getstacksize(const pthread _attr_t *restrict attr, size_t restrict stacksize).

- pthread_attr_setstacksize(pthread _attr_t *restrict attr, size_t stacksize).

Chương trình minh họa Init và Destroy:

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
int main(){
        pthread_attr_t tattr;
        int ret;
        if (pthread_attr_init (&tattr) == 0)
        {
                printf("int successed\n");
        }
        else
        {
                printf("init failed\n");
        }
        if (pthread_attr_destroy(&tattr) == 0)
        {
                printf("destroy successed\n");
        }
        else
        {
                printf("destroy failed\n");
        }
        return 1;
}
```

Chú thích (lượt bỏ cụm “pthread_attr_” trước tên hàm):

Hàm init: tạo các thuộc tính của thread với giá trị mặc định. Trả về 0 nếu tạo thành công.

```
The pthread_attr_init() function initializes the thread attributes object pointed to
by attr with default attribute values. After this call, individual attributes of the
object can be set using various related functions (listed under SEE ALSO), and then
the object can be used in one or more pthread_create(3) calls that create threads.
```

Hàm destroy: hủy object, thu hồi vùng nhớ của các thuộc tính. Trả về 0 nếu hủy thành công.

```
When a thread attributes object is no longer required, it should be destroyed using
the pthread_attr_destroy() function. Destroying a thread attributes object has no
effect on threads that were created using that object.
```

Chạy chương trình:

```
pngha2@pngha3:~$ ./bt3_ini_des
int successed
destroy successed
```

Logic chương trình: Hàm init tạo các thuộc tính của thread, và dùng hàm destroy để thu hồi vùng nhớ của thread đó khi không dùng nữa. Vì chương trình tạo và hủy thuộc tính thành công nên trả về 0, khi đó thông báo “init successed” và “destroy successed”.

Minh họa getdetachstate và setdetachstate:

Thuộc tính trạng thái detach: quyết định thread tạo từ các thuộc tính của thread attributes object sẽ ở trạng thái joinable hoặc detach.

Chú thích (lược bỏ cụm từ “pthread_attr”):

```
SYNOPSIS
       #include <pthread.h>

       int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
       int pthread_attr_getdetachstate(const pthread_attr_t *attr, int *detachstate);
```

Hàm setdetachstate: thay đổi thuộc tính trạng thái detach của thread bằng giá trị *detachstate* .

Hàm getdetachstate: lấy ra thuộc tính trạng thái detach của thread, lưu vào biến của *detachstate*.

```
The pthread_attr_setdetachstate() function sets the detach state attribute of the
thread attributes object referred to by attr to the value specified in detachstate.
The detach state attribute determines whether a thread created using the thread at-
tributes object attr will be created in a joinable or a detached state.
```

Chương trình:

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <limits.h>
int main(){
        pthread_attr_t tattr;
        int ret;
        int date;
        if (pthread_attr_init (&tattr) == 0)
        {
                printf ("Init successed\n");
                pthread_attr_getdetachstate(&tattr, &date);
                printf ("Detach hien tai %d\n", date);

                pthread_attr_setdetachstate(&tattr, PTHREAD_CREATE_DETACHED);
                pthread_attr_getdetachstate(&tattr, &date);
                printf("Detach sau %d\n", date);
        }
        else {
                printf ("init failed\n");
                return 1;
        }

        if (pthread_attr_destroy(&tattr) == 0){
                printf ("Destroy successed\n");
        }
        else {
                printf ("Destroy failed");
        }
        return 0;
}
```

Chạy chương trình:

```
pngha2@pngha3:~$ ./bt3_get_set_detachstate
Init successed
Detach hien tai 0
Detach sau 1
Destroy successed
```

Logic chương trình (lược bỏ cụm từ “pthread_attr”):

Khởi tạo các thuộc tính của thread bằng hàm init. Nếu tạo thành công, lấy trạng thái detach của thread bằng hàm get, gán vào biến date. Sau đó thay đổi trạng thái detach bằng hàm set thành trạng thái “CREATE_DETACHED”; xem trạng thái mới này bằng hàm get.

Sau cùng, thu hồi vùng nhớ bằng hàm destroy.

Minh họa getguardsize và setguardsize:

Thuộc tính guardsize: cung cấp khả năng bảo vệ tránh trường hợp tràn của stack pointer.

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <limits.h>
int main(){
        pthread_attr_t tattr;
        int ret;
        long unsigned int get;
        size_t set;
        set = 35000;
        if (pthread_attr_init (&tattr) == 0)
        {
                printf ("Init successed\n");
                pthread_attr_getguardsize(&tattr, &get);
                printf ("Current guardsize: %ld\n", get);

                pthread_attr_setguardsize(&tattr,set);
                pthread_attr_getguardsize(&tattr, &get);
                printf("Changed guardsize: %ld\n", get);
        }
        else {
                printf ("init failed\n");
                return 1;
        }

        if (pthread_attr_destroy(&tattr) == 0){
                printf ("Destroy successed\n");
        }
        else {
                printf ("Destroy failed");
        }
        return 0;
}
```

Chú thích (lược bỏ cụm từ “pthread_attr”):

Biến set: thay đổi giá trị cho guardsize

Hàm getguardsize, setguardsize: lấy và thay đổi guardsize (cách hoạt động tương tự phần detachstate)

Logic chương trình: Cũng giống như phần minh họa detachstate. Về logic chương trình: khởi tạo thuộc tính mới của thread, lấy thông số guardsize bằng hàm get và thay đổi nó bằng hàm set.

Chạy chương trình:

```
pngha2@pngha3:~$ gcc -o bt3_get_set_guardsize bt3_get_set_guardsize.c -pthread
pngha2@pngha3:~$ ./bt3_get_set_guardsize
Init successed
Current guardsize: 4096
Changed guardsize: 35000
Destroy successed
```

Minh họa getinheritsched và setinheritsched:

Thuộc tính inheritsched: quyết định liệu cái thread được tạo ra sẽ kế thừa các thuộc tính lập lịch từ thread gọi nó hay từ thread attr.

```
int pthread_attr_setinheritsched(pthread_attr_t *attr,
                                 int inheritsched);
int pthread_attr_getinheritsched(const pthread_attr_t *attr,
                                 int *inheritsched);
```

The **pthread_attr_setinheritsched**() function sets the inherit-scheduler attribute of the thread attributes object referred to by *attr* to the value specified in *inheritsched*. The inherit-scheduler attribute determines whether a thread created using the thread attributes object *attr* will inherit its scheduling attributes from the calling thread or whether it will take them from *attr*.

Chú thích về hàm get, set và logic chương trình: tương tự phần minh họa detach state. Chương trình sẽ set thuộc tính inheritsched thành PTHREAD_INHERIT_SCHED

Chương trình:

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <limits.h>
int main(){
        pthread_attr_t tattr;
        int get;
        if (pthread_attr_init (&tattr) == 0)
        {
                printf ("Init successed\n");
                pthread_attr_getinheritsched (&tattr, &get);
                printf ("Current scope: %d\n", get);

                pthread_attr_setinheritsched (&tattr,PTHREAD_INHERIT_SCHED);
                pthread_attr_getinheritsched (&tattr, &get);
                printf("Changed scope: %d\n", get);
        }
        else {
                printf ("init failed\n");
                return 1;
        }

        if (pthread_attr_destroy(&tattr) == 0){
                printf ("Destroy successed\n");
        }
        else {
                printf ("Destroy failed");
        }
        return 0;
}
```

Chạy chương trình:

```
pngha2@pngha3:~$ ./bt3_get_set_inheritsched
Init successed
Current scope: 0
Changed scope: 0
Destroy successed
```

Minh họa getschedparam và setschedparam:

Thuộc tính schedparam: tạo tham số lập lịch của 1 thread.

```c
int pthread_attr_setschedparam(pthread_attr_t *attr,
                               const struct sched_param *param);
int pthread_attr_getschedparam(const pthread_attr_t *attr,
                               struct sched_param *param);
```

```
The pthread_attr_setschedparam() function sets the scheduling parameter at-
tributes of the thread attributes object referred to by attr to the values
specified in the buffer pointed to by param. These attributes determine the
scheduling parameters of a thread created using the thread attributes object
attr.

The pthread_attr_getschedparam() returns the scheduling parameter attributes
of the thread attributes object attr in the buffer pointed to by param.

Scheduling parameters are maintained in the following structure:

    struct sched_param {
        int sched_priority;     /* Scheduling priority */
    };
```

Chú thích:

Hàm set: tạo các thông số lập lịch trong tattr bằng cách sử dụng giá trị từ 1 biến kiểu cấu trúc sched_param..

Hàm get: nhận thuộc tính ưu tiên lập lịch từ attr và lưu trữ nó vào 1 biến kiểu cấu trúc sched_param.

Nếu hàm chạy thành công, trả về 0. Ngược lại sẽ trả về giá trị khác 0.

Chương trình:

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
int main(){
        pthread_attr_t tattr;
        int ret;
        struct sched_param param;
        param.sched_priority = 999;
        if (pthread_attr_setschedparam (&tattr, &param) == 0){
                printf("Set successed\n");
        } else {
                printf("Set failed\n");
        }
        if (pthread_attr_getschedparam (&tattr, &param) == 0){
                printf("Get successed, param is: %d \n", param.sched_priority);
        } else {
                printf("Get failed\n");
        }
        return 0;
}
```

Logic chương trình: Tạo ra 1 thread attributes. Sau đó tạo 1 biến param có kiểu cấu trúc là sched_param với độ ưu tiên = 999. Thay đổi schedparam của thread atributes bằng với param và lấy schedparam mới từ thread attributes ra.

Chạy chương trình:

```
pngha2@pngha3:~$ gcc -o bt3_get_set_schedparam bt3_get_set_schedparam.c
pngha2@pngha3:~$ ./bt3_get_set_schedparam
Set failed
Get successed, param is: 1490642400
```

Theo chương trình trên, set độ ưu tiên thành 999 cho thread đã không thành công. Lấy thông số ra và lưu vào biến param thì thành công, với độ ưu tiên = 1490642400.

Minh họa getschedpolicy và setschedpolicy:

Thuộc tính schedpolicy: tạo chính sách lập lịch của 1 thread.

```
The pthread_attr_setschedpolicy() function sets the scheduling policy attribute of the thread  attributes  ob-
ject referred to by attr to the value specified in policy.  This attribute determines the scheduling policy of
a thread created using the thread attributes object attr.
```

Chú thích:

```
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
int pthread_attr_getschedpolicy(const pthread_attr_t *attr, int *policy);
```

Hàm set schedpolicy dùng để đặt lịch định thời theo biến policy.

Hàm get schedpolicy dùng để truy xuất lịch định thời và lưu vào biến policy.

Nếu thành công, trả về 0. Ngược lại sẽ trả về nột giá trị khác 0.

Chương trình:

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <limits.h>
int main(){
        pthread_attr_t tattr;
        int poll;
        if (pthread_attr_init(&tattr) == 0){
                printf("Init successed\n");
                pthread_attr_getschedpolicy (&tattr, &poll);
                printf("current Policy: %d\n", poll);
                pthread_attr_setschedpolicy (&tattr, SCHED_OTHER);
                pthread_attr_getschedpolicy (&tattr, &poll);
                printf("Changed Policy: %d\n", poll);
        } else {
                printf("Init failed\n");
                return 1;
        }
        if (pthread_attr_destroy(&tattr) == 0){
                printf ("Destroy successed\n");
        }
        else {
                printf ("Destroy failed\n");
        }
        return 0;
}
```

Logic chương trình:

Khởi tạo 1 đối tượng thread attributes bằng hàm init. Lấy sched_policy của đối tượng này bằng hàm get, lưu vào biến poll. Thay đổi sched_policy bằng hàm set với giá trị là SCHED_OTHER; rồi lấy sched_policy vừa thay đổi bằng hàm get.

Chạy chương trình:

```
pngha2@pngha3:~$ ./bt3_schedpolicy
Init successed
current Policy: 32764
Changed Policy: 32764
Destroy successed
```

Minh họa getstacksize và setstacksize:

Thuộc tính stacksize: kích cỡ tối thiểu (byte) được cấp phát cho thread.

```
The pthread_attr_setstacksize() function sets the stack size attribute of the thread attributes object referred to by attr to the value specified in stacksize.

The stack size attribute determines the minimum size (in bytes) that will be allocated for threads created using the thread attributes object attr.

The pthread_attr_getstacksize() function returns the stack size attribute of the thread attributes object referred to by attr in the buffer pointed to by stacksize.
```

Chú thích:

Hàm set stacksize dùng để đặt lại kích thước ngăn xếp.

Hàm get stacksize trả về giá trị trong biến được trỏ tới bởi stacksize.

Nếu thành công, trả về 0. Ngược lại sẽ trả về nột giá trị khác 0.

Chương trình:

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <limits.h>
int main(){
        pthread_attr_t tattr;
        int poll;
        size_t get;
        size_t size = 350123;

        if (pthread_attr_init(&tattr) == 0){
                printf("Init successed\n");
                pthread_attr_getstacksize (&tattr, &get);
                printf("current stacksize: %ld\n", get);
                pthread_attr_setstacksize (&tattr, size);
                pthread_attr_getstacksize (&tattr, &get);
                printf("Changed stacksize: %ld\n", get);
        } else {
                printf("Init failed\n");
                return 1;
        }
        if (pthread_attr_destroy(&tattr) == 0){
                printf ("Destroy successed\n");
        }
        else {
                printf ("Destroy failed\n");
        }
        return 0;
}
```

Logic chương trình:

Khởi tạo các thuộc tính cho 1 thread bằng hàm init. Lấy thuộc tính stacksize bằng hàm get. Sau đó thay đổi thuộc tính này bằng hàm set với tham số size; rồi lấy giá trị thay đổi ra. Sau cùng là thu hồi vùng nhớ của thread attributes.

Chạy chương trình:

```
pngha2@pngha3:~$ ./bt3_stacksize
Init successed
current stacksize: 8388608
Changed stacksize: 350123
Destroy successed
```

4. Viết chương trình làm các công việc sau theo thứ tự:

a. In ra dòng chữ: "Welcome to IT007, I am !"

b. Mở tệp abcd.txt bằng vim editor

c. Tắt vim editor khi người dùng nhấn CRTL+C

d. Khi người dùng nhấn CTRL+C thì in ra dòng chữ: "You are pressed CTRL+C! Goodbye!"

giải thích:

- Ctr+C: tương ứng với tín hiệu SIGINT, làm ngắt tiến trình (interupt)
- Hàm on_signint: in ra dòng chữ yêu cầu của câu d khi nhấn Ctrl + C lần thứ 2.
- Hàm on_signint2: kill vim khi nhấn Ctrl + C lần thứ 1:
    + Lệnh killall: kill tất cả các tiến trình được kể tên
- Hàm OpenVim: tạo tiến trình mới thực thi lệnh vim abcd.txt
    + Hàm fork(): Tạo tiến trình mới
    + Hàm execl: Thực thi 1 lệnh dựa vào lệnh nằm trong đường dẫn cho trước, dùng để thực thi lệnh "vim abcd.text"
    + Hàm signal: khi nhập Ctrl + C (tương ứng tín hiệu SIGINT), tiến trình sẽ thực thi việc thoát Vim (lần 1)
- Trong main:
    - Hàm pthread_create: tạo tiến trình
    - Hàm pthread_join: kết hợp tiến trình
    - Hàm signal: gọi bộ xử lý với signum, dùng cho lần Ctrl+C thứ 2 để xuất theo yêu cầu câu d.

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <limits.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
long id_thread;
int loop = 2;
void on_signint(){
        if (loop == 1){
                printf("\nYou are pressing CTRL+C! Good bye!\n");
        }
        loop--;
}
void on_signint2(){
        if (loop == 2){
                system("killall vim");
        }
        loop=1;
}
void *openVim(void *args){
        pid_t pid;
        pid = fork();
        if (pid == 0){
                execl("/usr/bin/vim", "vim", "abcd.txt", NULL);
        }
        signal(SIGINT, on_signint2);
        while(loop!=1){}
}
int main(){
        printf("\nWelcome to IT007, I am 21520586!\n");
        pthread_create(&id_thread, NULL, &openVim, (void*)id_thread);
        pthread_join(id_thread, NULL);
        signal (SIGINT, on_signint);
        while(loop){}
        return 0;
}
```

Chạy chương trình:

```
pngha2@pngha3:~$ ./bt5

Welcome to IT007, I am 21520586!
^C^C
You are pressing CTRL+C! Good bye!
```