Name: Huỳnh Viết Tuấn Kiệt

ID: 20521494

Class: IT007.M13.2

OPERATING SYSTEM
LAB 3'S REPORT

SUMMARY

Task		**Status**	**Page**
Question	Ex 1	Hoàn thành	2
	Ex 2	Hoàn thành	4
	Ex 3	Hoàn thành	5
	Ex 4	Hoàn thành	8

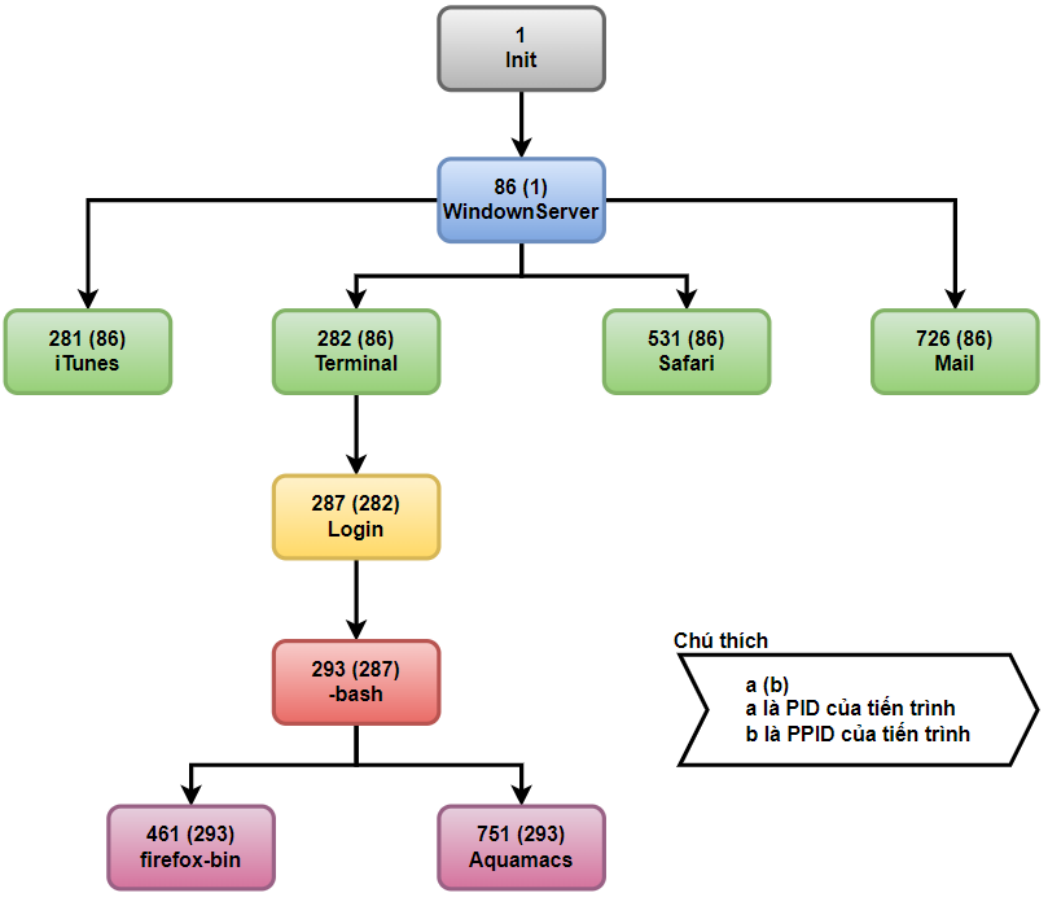
Self-scrores: 7.5

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

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

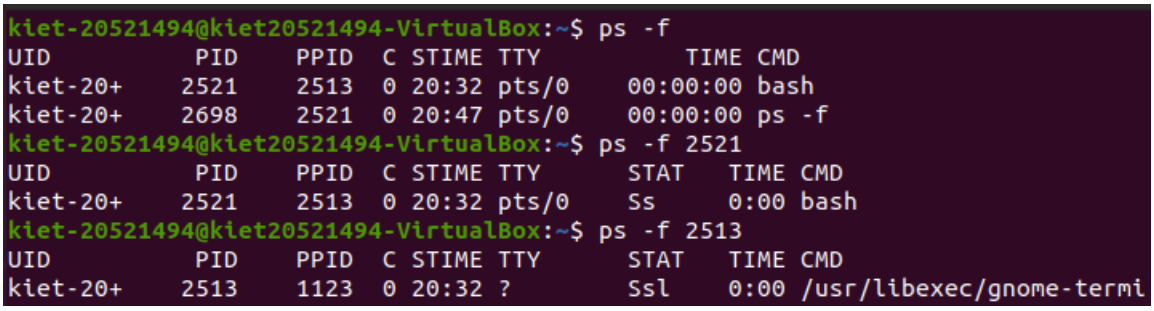
UID	PID	PPID	COMMAND
88	86	1	WindowServer
501	281	86	iTunes
501	282	86	Terminal
0	287	282	login
501	461	293	firefox-bin
501	531	86	Safari
501	726	86	Mail
501	751	293	Aquamacs
501	293	287	-bash

Hình 1.1: Các tiến trình



Hình 1.2: Cây quan hệ parent – child của các tiến trình

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

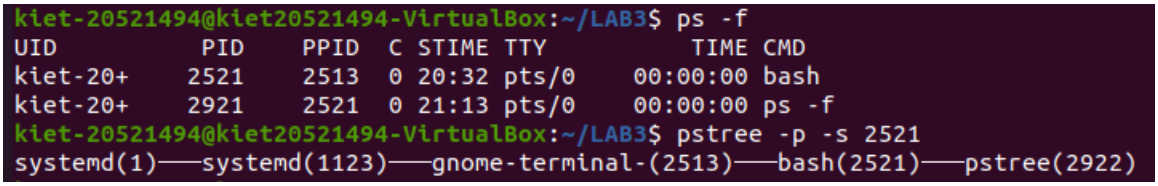


Hình 1.3: Tìm kiếm tiến trình cha của 1 tiến trình sử dụng lệnh ps

Lệnh `ps -f` để hiển thị thông tin các tiến trình tại thời điểm khởi chạy lệnh

- Lúc này có 2 tiến trình đang thực thi tại thời điểm khởi chạy lệnh ps gồm tiến trính `bash` và chính tiến trình `ps -f`
- Cột **PPID** là **PID** của tiến trình cha của tiến trình được chỉ định
- Để tìm tiến trình cha của 1 tiến trình, sử dụng lệnh `ps -f [PID]`, chẳng hạn như trên *hình 1.2*, muốn biết tiến trình cha của tiến trình có *PID 2521*, chúng ta sử dụng lệnh `ps -f 2521`

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



Hình 1.4: Tìm kiếm tiến trình cha của 1 tiến trình sử dụng lệnh pstree

Lệnh `ps -f` để hiển thị thông tin các tiến trình tại thời điểm khởi chạy lệnh

Sử dụng lệnh `pstree -p -s [PID]` để xây dựng cây tiến trình

- `-p` để hiện **PID** của tiến trình đó
- `-s` để hiển thị các tiến trình cha của tiến trình được gọi.

Chẳng hạn như trên *hình 1.4*, sử dụng lệnh `pstree -p -s 2521` để vẽ cây tiến trình của tiến trình có *PID 2521*

2. Chương trình bên dưới in ra kết quả gì? Giải thích tại sao (Sources code đính kèm trong LAB3)

```c
/*########################################
# University of Information Technology
# IT007 Operating System
# Huynh Viet Tuan Kiet, ID 20521494
# File ex2.c
########################################*/

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/types.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);
}
```

Hình 2.1: Hiện thực chương trình

```
kiet-20521494@kiet20521494-VirtualBox:~/LAB3$ ./ex2
I see 17 coconuts!
```

Hình 2.2: Kết quả khi thực thi chương trình

Giải thích:

Chương trình sẽ in ra kết quả `I see 17 coconuts` thay vì kết quả khác vì tiến trình con sinh ra được thực thi và kết thúc bằng lệnh `exit(0)` trước khi thực hiện câu lệnh `printf`, nên câu lệnh `printf` của tiến trình cha sẽ được thực thi. Mà tiến trình cha và tiến trình con có bộ nhớ riêng nên biến `num_coconuts` vẫn có giá trị là `17`.

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_*)

`pthread_attr_init();`

Hàm `pthread_attr_init()` khởi tạo đối tượng thuộc tính luồng được trỏ tới bởi **attr** với các giá trị thuộc tính mặc định.

`pthread_attr_destroy();`

Hàm `pthread_attr_destroy()` sẽ hủy một đối tượng thuộc tính luồng. Một lỗi xảy ra nếu một đối tượng thuộc tính luồng được sử dụng sau khi nó đã bị phá hủy. **attr** là một con trỏ đến một đối tượng thuộc tính luồng được khởi tạo bởi `pthread_attr_init()`.

`pthread_attr_getdetachstate();`

`pthread_attr_setdetachstate();`

Hàm `pthread_attr_getdetachstate()` trả về giá trị hiện tại của thuộc tính **detachstate** cho đối tượng thuộc tính luồng, **attr** được tạo bởi `pthread_attr_init()`. Các giá trị thuộc tính **detachstate** là: **Undetached** (Một luồng chưa được xóa sẽ giữ tài nguyên của nó sau khi kết thúc) hoặc **Detached** (Một luồng tách rời sẽ có tài nguyên của nó được hệ thống tự động giải phóng khi kết thúc).

Hàm `pthread_attr_setdetachstate()` đặt thuộc tính trạng thái tách của đối tượng thuộc tính luồng được tham chiếu bởi attr thành giá trị được chỉ định trong **detachstate**. Thuộc tính trạng thái tách rời xác định xem một tiểu trình được tạo bằng đối tượng thuộc tính tiểu trình sẽ được tạo ở trạng thái có thể kết hợp hay tách rời.

pthread_attr_getguardsize()

pthread_attr_setguardsize()

Hàm `pthread_attr_getguardsize()` lấy thuộc tính Guardize từ attr và lưu trữ nó vào Guardize.

Hàm `pthread_attr_setguardsize()` đặt thuộc tính Guardize trong **attr** bằng cách sử dụng giá trị của Guardize.

pthread_attr_getinheritsched()

pthread_attr_setinheritsched()

Hàm `pthread_attr_getinheritsched()` sẽ lấy thuộc tính kế thừa trong đối số attr.

Hàm `pthread_attr_setinheritsched()` đặt thuộc tính kế thừa lập lịch của đối tượng thuộc tính luồng được tham chiếu bởi **attr** đến giá trị được chỉ định trong kế thừa.

pthread_attr_getshedparam()

pthread_attr_setshedparam()

Hàm `pthread_attr_getschedparam()` nhận thuộc tính ưu tiên lập lịch từ **attr** và lưu trữ nó vào **param**. **Param** trỏ đến đối tượng tham số lập lịch do người dùng xác định mà `pthread_attr_getschedparam()` sao chép thuộc tính ưu tiên lập lịch luồng.

Hàm `pthread_attr_setschedparam()` đặt các thuộc tính tham số lập lịch của đối tượng thuộc tính luồng được tham chiếu bởi **attr** đến các giá trị được chỉ định trong bộ đệm được trỏ tới bởi tham số.

pthread_attr_getshedpolicy()

pthread_attr_setshedpolicy()

Hàm `pthread_attr_getschedpolicy()` trả về thuộc tính **sheduling policy** của đối tượng thuộc tính luồng trong bộ đệm được trỏ tới bởi **policy**.

Hàm `pthread_attr_setschedpolicy()` đặt thuộc tính **sheduling policy** của đối tượng thuộc tính luồng được tham chiếu bởi **attr** đến giá trị được chỉ định trong **policy**. Thuộc tính này xác định chính sách lập lịch của một luồng được tạo bằng cách sử dụng đối tượng thuộc tính luồng.

pthread_attr_getscope()

pthread_attr_setscope()

Hàm `pthread_attr_getscope()` sẽ nhận thuộc tính **contentionscope** trong đối tượng **attr.**

Hàm `pthread_attr_setscope()` đặ thuộc tính **contentionscope** của đối tượng thuộc tính luồng được tham chiếu bởi **attr** đến giá trị được chỉ định trong phạm vi.

pthread_attr_getstack()

pthread_attr_setstack()

Hàm `pthread_attr_getstack()` trả về địa chỉ ngăn xếp và các thuộc tính kích thước ngăn xếp của đối tượng thuộc tính luồng được tham chiếu bởi **attr** trong bộ đệm được trỏ tới bởi **stackaddr** và **stacksize** tương ứng.

Hàm `pthread_attr_setstack()` đặt các thuộc tính địa chỉ ngăn xếp và kích thước ngăn xếp của đối tượng thuộc tính luồng được tham chiếu bởi **attr** tới các giá trị được chỉ định trong **stackaddr** và **stacksize** tương ứng

...

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

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

b. Mở tệp abcd.txt bằng vim editor

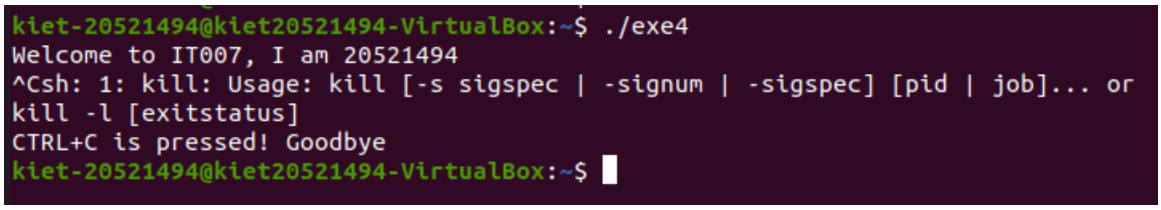
c. Tắt vim editor khi người dùng nhấn CRTL+C

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

```c
/*########################################
# University of Information Technology
# IT007 Operating System
# Huynh Viet Tuan Kiet, ID 20521494
# File exe4.c
########################################*/

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <pthread.h>

int close_vim = 0;
void open_Vim()
{
        system("vim ~/abcd.txt");
}

void on_sigint()
{
        system("kill -9 `pidof vim`");
        close_vim = 1;
}
int main()
{
        signal(SIGINT, on_sigint);
        printf("Welcome to IT007, I am 20521494\n");
        open_Vim();
        while (close_vim != 1){}
        printf("CTRL+C is pressed! Goodbye\n");
        return 0;
}
```

Hình 4.1: Hiện thực chương trình

- Lệnh `signal(SIGINT, on_sigint);` dùng để thông báo nếu người dùng gửi tín hiệu ngắt (Ctrl + C)
- Hàm `on_sigint()` thực hiện 2 nhiệm vụ: `kill -9 `pidof vim`` để thoát tiến trình ngay lập tức, và gán `close_vim = 1` để ngừng vòng lặp while dẫn đến dừng chương trình
- Lệnh `open_Vim();` sẽ gọi tới hàm `open_Vim()` và hệ thống sẽ mở tệp *abcd.txt*
- Biến toàn cục `close_vim` được gán ban đầu bằng `0`, và khi không nhận được *signal* ***SIGINT***, nó sẽ lặp vô hạn cho đến khi người dùng nhấn CTRL+C



Hình 4.2: Kết quả thực thi chương trình

**Lưu ý: Chương trình được viết dựa trên hướng dẫn của thầy Lộc trong buổi thực hành thứ 3*