Name: NGUYỄN ANH TÀI

ID:20520924

Class: IT007.M22.1

OPERATING SYSTEM
LAB 03'S REPORT

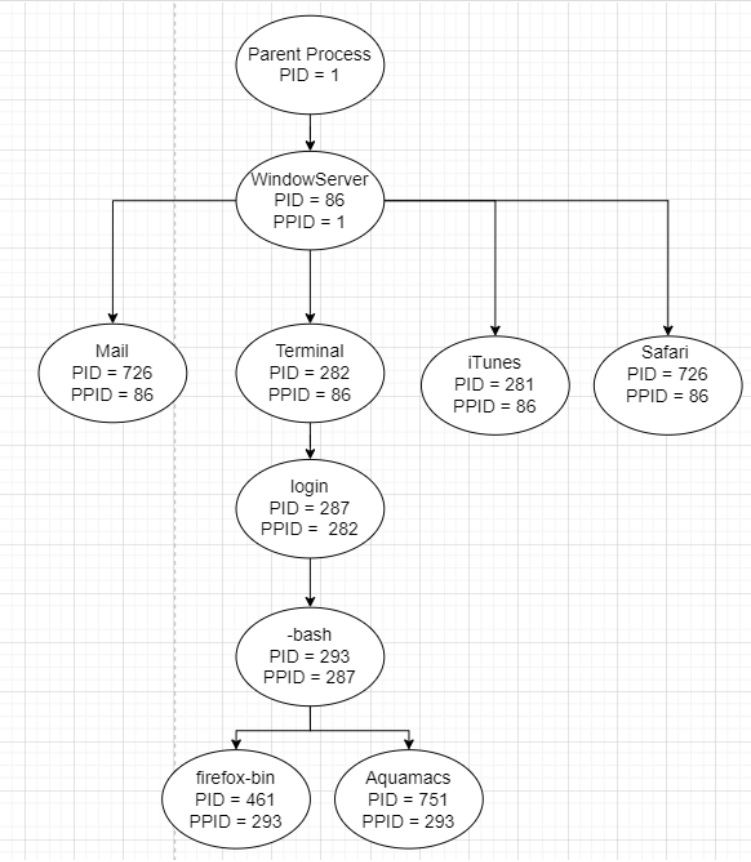
SUMMARY

Self-scrores: 9

TASK 01: 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



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

Bước 1: sử dụng man ps để tìm hiểu các thông tin về lệnh ps trong linux

Từ các kết quả của man ps ta có được cách dùng lệnh ps:

```
To see every process on the system using standard syntax:
    ps -e
    ps -ef
    ps -eF
    ps -ely
```

```
p pidlist
       Select by process ID.  Identical to -p and --pid.

-p pidlist
       Select by PID.  This selects the processes whose process ID numbers appear in pidlist.  Identical to p and --pid.

--pid pidlist
       Select by process ID.  Identical to -p and p.

--ppid pidlist
       Select by parent process ID.  This selects the processes with a parent process ID in pidlist.  That is, it selects processes that are children of those listed in pidlist.
```

```
 -f     Do full-format listing.  This option can be combined with many other UNIX-style options to add additional columns.  It also causes the command arguments to be printed.  When
        used with -L, the NLWP (number of threads) and LWP (thread ID) columns will be added.  See the c option, the format keyword args, and the format keyword comm.
```

Hình 1: các pattern để lệnh ps hiện thị được thông tin cần thiết

Bước 2: Tiến Hành Sử dụng lệnh ps để đưa ra kq như đề bài yêu cầu

2.1. Dùng ps -e -f để hiện thị đầy đủ thông tin tất cả tiến trình đang chạy trong hệ thống

```
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 15:14 ?        00:00:00 /init
root          17     1  0 15:14 ?        00:00:00 /init
root          18    17  0 15:14 ?        00:00:00 /init
root          19    18  0 15:14 pts/0    00:00:00 sh -c "$VSCODE_W
root          20    19  0 15:14 pts/0    00:00:00 sh /mnt/c/Users/
root          25    20  0 15:14 pts/0    00:00:00 sh /root/.vscode
root          29    25  0 15:14 pts/0    00:00:13 /root/.vscode-se
root          40    29  0 15:14 pts/0    00:00:11 /root/.vscode-se
root          77     1  0 15:14 ?        00:00:00 /init
root          78    77  0 15:14 ?        00:00:01 /init
root          79    78  0 15:14 pts/1    00:00:02 /root/.vscode-se
root          87     1  0 15:14 ?        00:00:00 /init
root          88    87  0 15:14 ?        00:00:00 /init
root          89    88  0 15:14 pts/2    00:00:01 /root/.vscode-se
root          98    29  0 15:14 pts/0    00:00:00 /root/.vscode-se
root         203    29  0 15:14 pts/0    00:00:13 /root/.vscode-se
root         213    18  0 15:14 ?        00:00:01 /usr/bin/tmux
root         214   213  0 15:14 pts/5    00:00:00 -bash
root         285   203  0 15:14 pts/0    00:00:02 /root/.vscode-se
root        1359   213  0 15:27 pts/6    00:00:00 -bash
root        3995   285  0 16:04 pts/0    00:00:00 /root/.vscode-se
root        4335   213  0 16:07 pts/8    00:00:00 -bash
root        4475   213  0 16:08 pts/10   00:00:00 -bash
root        4586    40  0 16:09 pts/4    00:00:00 /usr/bin/bash
root        6590  4586  0 16:32 pts/4    00:00:00 ps -e -f
```

Hình 2 Kết quả sau khi thực hiện lệnh ps -e -f

2.2. Dùng ps --pid [pid của tiến trinh] -f để xem thông tin đầy đủ của tiến trình cụ thể.

-Chọn pid của tiến trình = 3995

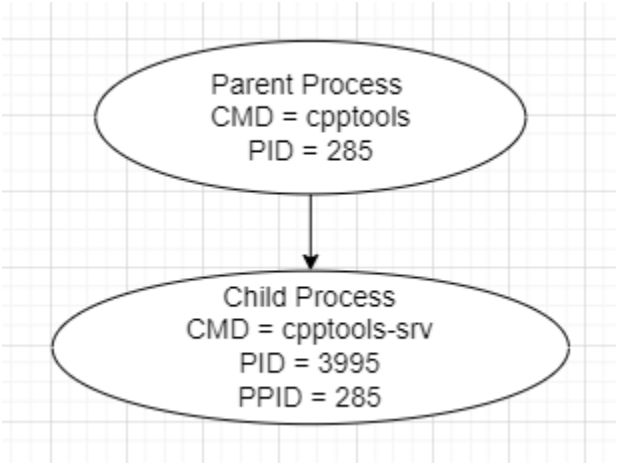
```
root@MSI:/home/Tai-20520924# ps --pid 3995 -f
UID          PID  PPID  C STIME TTY          TIME CMD
root        3995   285  0 16:04 pts/0    00:00:00 /root/.vscode-server/extensions/ms-vscode.cpptools-1.9.8-linux-x64/bin/cpptools-srv 285
```

Hình 3 Kết quả sau khi thực hiện lệnh ps --pid 3995

Giải thích: Ta có thể thấy được PPID = 285 chính là PID của tiến trình sinh ra tiến trình 3995

CMD = /root/.vscode-server/extensions/ms-vscode.cpptools-1.9.8-linux-x64/bin/cpptools-srv trong đó cpptools-srv là tên của tiến trình

Từ đó ta có thể có suy ra được quan hệ tiến trình như sau:



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.

Kiểm tra pstree đã được cài đặt chưa bằng lệnh whereis [tên của chương trình]

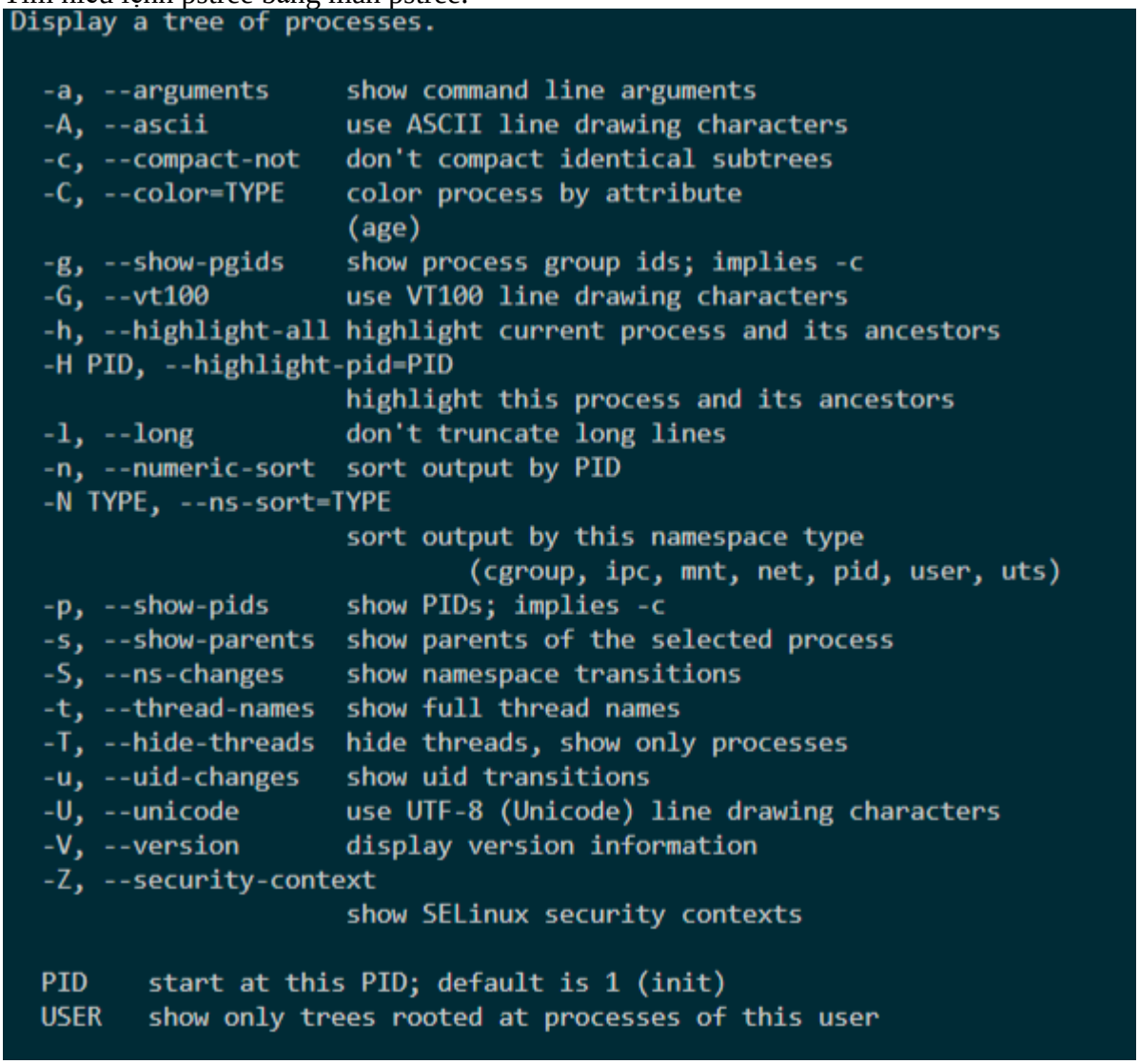
```
root@MSI:/home/Tai-20520924# whereis pstree
pstree: /usr/bin/pstree.x11 /usr/bin/pstree /usr/share/man/man1/pstree.1.gz
```

Hình 4 kết quả sau khi chạy lệnh whereis pstree

kết quả trả về đường dẫn nơi pstree được cài đặt

=> lệnh pstree đã được cài đặt trong HDH

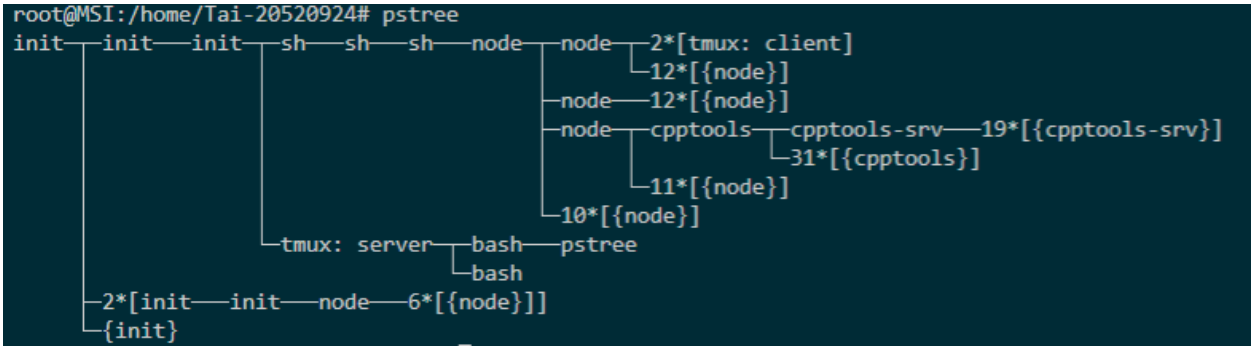
Tìm hiểu lệnh pstree bằng man pstree.



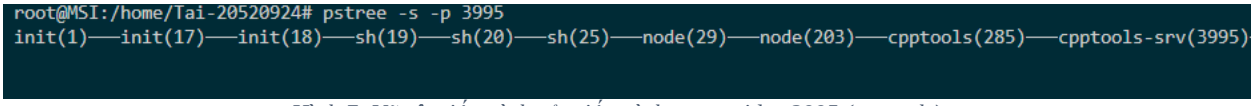
```
Display a tree of processes.

  -a, --arguments     show command line arguments
  -A, --ascii         use ASCII line drawing characters
  -c, --compact-not   don't compact identical subtrees
  -C, --color=TYPE    color process by attribute
                      (age)
  -g, --show-pgids    show process group ids; implies -c
  -G, --vt100         use VT100 line drawing characters
  -h, --highlight-all highlight current process and its ancestors
  -H PID, --highlight-pid=PID
                      highlight this process and its ancestors
  -l, --long          don't truncate long lines
  -n, --numeric-sort  sort output by PID
  -N TYPE, --ns-sort=TYPE
                      sort output by this namespace type
                              (cgroup, ipc, mnt, net, pid, user, uts)
  -p, --show-pids     show PIDs; implies -c
  -s, --show-parents  show parents of the selected process
  -S, --ns-changes    show namespace transitions
  -t, --thread-names  show full thread names
  -T, --hide-threads  hide threads, show only processes
  -u, --uid-changes   show uid transitions
  -U, --unicode       use UTF-8 (Unicode) line drawing characters
  -V, --version       display version information
  -Z, --security-context
                      show SELinux security contexts

  PID    start at this PID; default is 1 (init)
  USER   show only trees rooted at processes of this user
```

Hình 5 Kết quả sau khi sử dụng man pstree để tìm hiểu cách sử dụng pstree

Tiến hành sử dụng pstree



Hình 6 Hình vẽ cây quan quan hệ của toàn bộ tiến trình có trong hệ thống bằng lệnh pstree



Hình 7: Vẽ cây tiến trình của tiến trình mang pid = 3995 (cpptools)

Giải thích:
-lệnh pstree ở hình 6 sẽ vẽ ra cây tiến trình của toàn bộ tiến trình đang chạy trong hệ thống
-lệnh pstree -s -p 3995 ở hình 7 sẽ vẽ cây tiến trình của tiến trình mang pid 3995 và ta có thể thấy được quan hệ cha con của các tiến trình liên quan tới tiến trình pid 3995

TASK 02. Chương trình bên dưới in ra kết quả gì? Giải thích tại sao?

```c
/*##########################################
# University of Information Technology     #
# IT007 Operating System                   #
# <Your name>, <your Student ID>           #
# File: exercise_2.c                       #
##########################################*/

#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);
}
```

Chương trình vẫn chưa chạy được vì thiếu thư viện

```c
/*#####################################
# University of Information Technology #
# IT007 Operating System    #
# <Nguyen Anh Tai>, <20520924> #
# File: Bai2.c #
#####################################*/
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{
    pid_t pid;
    int num_coconuts = 17;
    pid = fork();
    if (pid == 0)
    {
        num_coconuts = 42;
        exit(0); // tiến trình con kết thúc rồi, k in gì nữa đâ
    }
    else
    {
        wait(NULL); /*wait until the child terminates */
    }

    printf("I see %d coconuts!\n", num_coconuts);
    exit(0);
}
```

Hình 7: Chương trình sau khi thêm thư viện <stdlib.h> <sys/wait.h> <unistd.h>

Tiến hành thực thi chương trình

```
root@MSI:/home/Tai-20520924/Lab03/Bao Cao# gcc Bai2.c -o Bai2
root@MSI:/home/Tai-20520924/Lab03/Bao Cao#
```

Trong đó gcc chính là lệnh gọi trình biên dịch C/C++
Bai2.c chính là đường dẫn tới chương trình cần thực thi
-o Bai2 là xuất file thực thi thành file có tên Bai2

```
root@MSI:/home/Tai-20520924/Lab03/Bao Cao# ./Bai2
I see 17 coconuts!
```

Sau khi thực thi thì ta có thể thấy kết quả in ra trên màn hình

Giải thích kết quả:
- Vì tiến trình con có dòng lệnh exit(0) nên sau khi biến num_coconuts ở tiến trình con được gán thì sẽ kết thuc luôn mà không in ra màn hình gì cả. Kết quả chúng ta thấy ở màn hình chính là biến num_coconuts ở tiến trình cha là 17.
- Vậy làm sao để in ra màn hình được biến num_coconuts ở tiến trình con?

Ta chỉ cần xoá dòng lệnh 19 exit (0)

```c
/*########################################
# University of Information Technology #
# IT007 Operating System    #
# <Nguyen Anh Tai>, <20520924> #
# File: Bai2.c #
########################################*/
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{
    pid_t pid;
    int num_coconuts = 17;
    pid = fork();
    if (pid == 0)
    {
        num_coconuts = 42;
    }
    else
    {
        wait(NULL); /*wait until the child terminates */
    }

    printf("I see %d coconuts!\n", num_coconuts);
    exit(0);
}
```

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

root@MSI:/home/Tai-20520924/Lab03/Bao Cao# ./Bai2
I see 42 coconuts!
I see 17 coconuts!
```

Hình 8 Chương trình sau khi xoá dòng 19 và kết quả thực thi

Sau khi thực thi thì chương trình in ra 42 trước thay vì 17, điều đó xảy ra là vì ở tiến trình cha sẽ đi vào dòng lệnh else, ở đây tiến trình cha gặp lệnh wait (NULL) tức là đợi tới khi chương trình con terminate rồi mới thực thi tiếp.

TASK 03. 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_*)

POSIX thread là gì?
POSIX thread là một Thư viện chuẩn cho lập trình thread trong C/C++. Nó cho phép tạo ra các ứng dụng chạy song song theo luồng, rất hiệu quả trên hệ thống nhiều bộ vi xử lý hoặc bộ vi xử lý nhiều nhân ở đó các luồng xử lý có thể được lập lịch chạy trên các bộ xử lý khác nhau do đó tăng được tốc độ xử lý song song hoặc xử lý phân tán.

Các hàm được sử dụng để làm thay đổi thuộc tính của pthread.

pthread_attr_init	Được sử dụng để khởi tạo đối tượng thuộc tính với giá trị mặc định.
pthread_attr_destroy	Được sử dụng để hủy một đối tượng thuộc tính luồng.
pthread_attr_getdetachstate **pthread_attr_setdetachstate**	Get/set trạng thái khởi tạo của thread, có thể thể là riêng lẻ hoặc kết hợp.
pthread_attr_getinheritsched **pthread_attr_setinheritsched**	Get/set thuộc tính kế thừa có trong cấu trúc thuộc tính.
pthread_attr_getschedparam **pthread_attr_setschedparam**	Get/Set các thuộc tính tham số lập lịch của đối số
pthread_attr_getschedpolicy **pthread_attr_setschedpolicy**	Get/set chính sách lịch trình (scheduling policy) của thread
pthread_attr_getguardsize **pthread_attr_setguardsize**	Set/get kích thước của khu vực an toàn của thread
pthread_attr_getscope **pthread_attr_setscope**	Get/set phạm vi của thread
pthread_attr_getstack **pthread_attr_setstack**	Get/set địa chỉ stack của thread
pthread_attr_getstacksize **pthread_attr_setstacksize**	Get/set kích thước stack của thread

Các chương trình minh họa tác động của các thuộc tính này.

1. Initializing Attributes

```c
/*####################################
# University of Information Technology #
# IT007 Operating System    #
# <Nguyen Anh Tai>, <20520924> #
# File: pthread_attr_init_example.c #
####################################*/
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
int main()
{
    pthread_attr_t tattr;
    int ret = pthread_attr_init(&tattr);
    /* câu lệnh này sẽ khởi tạo biến tattr thành một biến pthread_attr_t có các giá trị default */
    if (ret == 0)
    {
        printf("initialized an attribute (tattr) to the default value\n");
    }

    exit(0);
}
```

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

root@MSI:/home/Tai-20520924/Lab03/Bao Cao# ./init
initialized an attribute (tattr) to the default value
root@MSI:/home/Tai-20520924/Lab03/Bao Cao#
```

Hình 9 Chương trình khởi tạo pthread_attr_init mẫu

Sau khi thực thi xong dòng 14 thì biến tattr sẽ được khởi tạo thành một biến pthread_attr_t có các giá trị default. Các giá trị mặc định của biến tattr được xác định theo bảng sau

Attribute	Value	Result
scope	PTHREAD_SCOPE_PROCESS	New thread contends with other threads the process.
detachstate	PTHREAD_CREATE_JOINABLE	Completion status and thread *ID* are preserved after the thread exits.
stackaddr	NULL	New thread has system-allocated stack address.
stacksize	0	New thread has system-defined stack siz
priority	0	New thread has priority 0.
inheritsched	PTHREAD_EXPLICIT_SCHED	New thread does not inherit parent thre scheduling priority.
schedpolicy	SCHED_OTHER	New thread uses the traditionalSolaris time-sharing (TS) scheduling class.
guardsize	PAGESIZE	Stack overflow protection.

2. Destroying Attributes

Lab03 > Bao Cao > C example_pthread_Destroying_Attributes.c > main()

```c
/*#####################################
# University of Information Technology #
# IT007 Operating System    #
# <Nguyen Anh Tai>, <20520924> #
# File: example_pthread_Destroying_Attributes.c #
#####################################*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
void *TestThread(void *arg)
{
    printf("hello from the thread\n");
    pthread_exit(NULL);
}
int main()
{
    pthread_attr_t attr;
    pthread_t thid;
    if (pthread_attr_init(&attr) == -1)
    {
        perror("error in pthread_attr_init");
        exit(1);
    }
    if (pthread_create(&thid, &attr, TestThread, NULL) == -1)
    {
        perror("error in pthread_create");
        exit(2);
    }
    if (pthread_attr_destroy(&attr) == -1)
    {
        perror("error in pthread_attr_destroy");
        exit(5);
    }
    exit(0);
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

root@MSI:/home/Tai-20520924/Lab03/Bao Cao# gcc example_pthread_Destroying_Attributes.c -o destroy -pthread
root@MSI:/home/Tai-20520924/Lab03/Bao Cao# ./destroy
root@MSI:/home/Tai-20520924/Lab03/Bao Cao#
```

Hình 10: Chương trình mô tả quá trình huỷ một thuộc tính được tạo của thread

Tại câu lệnh 17 sẽ trả về 1 và in ra màn hình “error in pthread_attr_init” nếu khởi tạo thuộc tính mặc định thất bại.
Tại câu lệnh 22 sẽ trả về 2 và in ra màn hình “error in pthread_create" nếu tạo thread thất bại.
Tại câu lệnh 27 sẽ trả về 5 và in ra màn hình “error in pthread_attr_destroy" nếu huỷ bỏ thuộc tính thất bại.

3. Setting/Getting the Detach State

Hàm pthread_attr_setdetachstate() được dùng để set thuộc tính detach state của đối tượng thuộc tính luồng attr thành giá trị được chỉ định như sau:

PTHREAD_CREATE_DETACHED

Threads that are created using *attr* will be created in a detached state.

PTHREAD_CREATE_JOINABLE

Threads that are created using *attr* will be created in a joinable state.

Thuộc tính trạng thái tách ra xác định xem một luồng được tạo bằng đối tượng thuộc tính luồng sẽ được tạo ở trạng thái có thể kết hợp hay tách rời.

Lab03 > Bao Cao > C example_Setting_Getting_the_Detach_State.c > ...

```c
/*
# I160/M21.1 Operating System    #
# <Nguyen Anh Tai>, <20520924> #
# File: example_Setting_Getting_the_Detach_State.c #
#####################################*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void Get_Detach_State(pthread_attr_t attr)
{
    int foundstate;
    // Sử dụng hàm pthread_attr_getdetachstate để get trạng thái khởi tạo của tiến trình và gán vào foundstate
    pthread_attr_getdetachstate(&attr, &foundstate);
    // in ra màn hình các trạng thái khởi tạo mà hàm pthread_attr_getdetachstate trả về vào biến foundstate
    switch (foundstate)
    {
    case PTHREAD_CREATE_JOINABLE:
        printf("%d joinable\n", foundstate);
        break;
    case PTHREAD_CREATE_DETACHED:
        printf("%d detached\n", foundstate);
        break;
    default:
        exit(2);
    }
}
int main(void)
{
    pthread_attr_t attr;
    int rc, newstate;

    if (pthread_attr_init(&attr) == -1)
    // khởi tạo thuộc tính mặc định, nếu không thành công trả về 1
    {
        exit(1);
    }
    // Đặt newstate = PTHREAD_CREATE_DETACHED ở trạng thái DETACHED
    newstate = PTHREAD_CREATE_DETACHED;
    // Sử dụng hàm pthread_attr_setdetachstate để set trạng thái khởi tạo của tiến trình thành DETACHED
    pthread_attr_setdetachstate(&attr, newstate);
    Get_Detach_State(attr);

    // Đặt newstate = PTHREAD_CREATE_DETACHED ở trạng thái JOINABLE
    newstate = PTHREAD_CREATE_JOINABLE;
    // Sử dụng hàm pthread_attr_setdetachstate để set trạng thái khởi tạo của tiến trình thành JOINABLE
    pthread_attr_setdetachstate(&attr, newstate);
    Get_Detach_State(attr);

    exit(0);
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
root@MSI:/home/Tai-20520924/Lab03/Bao Cao# ./DetachState
1 detached
0 joinable
```

Hình 11: Chương trình minh hoạ Detach State và kết quả thực thi.

Giải thích kết quả:
2 kết quả của chương trình là kết quả của việc thay đổi thuộc tính Detach State ở dòng 40 và 46

các thay đổi được gán vào đối tượng thuộc tính attr bằng hàm pthread_attr_setdetachstate () và biến foundstate nhận được kết quả thông qua hàm pthread_attr_getdetachstate ()

4. Setting /Getting the Stack Guard Size

pthread_attr_setguardsize ()/pthread_attr_getguardsize () được dùng set/get thuộc tính guardsize của đối tượng thuộc tính attr.

-Thuộc tính guardsize được cung cấp vì những lí do sau:

+ tránh việc tràn có thể dẫn đến lãng phí tài nguyên hệ thống.
+ để dễ dàng phát hiện việc tràn dữ liệu ở các thread được cấp phát nhiều tài nguyên

```c
/*####################################
# University of Information Technology #
# IT007.M21.1 Operating System    #
# <Nguyen Anh Tai>, <20520924> #
# File: example_Destroying_Attributes.c #
####################################*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int main(void)
{
    int PAGESIZE = 1024;
    pthread_attr_t attr;
    int rc;
    size_t guardsize;

    if (pthread_attr_init(&attr) == -1)
    // nếu khởi tạo thuộc tính không thành công -> thoát
        exit(1);
    printf("Set guardsize to value of PAGESIZE.\n");
    if (pthread_attr_setguardsize(&attr, PAGESIZE) != 0)
        // nếu hàm pthread_attr_setguardsize trả về giá trị khác 0 -> set Stack Guard thất bại -> thoát
        exit(2);
    else
        printf("Set guardsize is %d\n", PAGESIZE);

    if (pthread_attr_getguardsize(&attr, &guardsize) != 0)
        // nếu hàm pthread_attr_getguardsize trả về giá trị khác 0 -> get Stack Guard thất bại -> thoát
        exit(3);
    else
        printf("Retrieved guardsize is %ld\n", guardsize);
    exit(0);
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

root@MSI:/home/Tai-20520924/Lab03/Bao Cao# gcc example_Setting_Getting_Stack_Guard_Size.c -o StackGuard -pthread
root@MSI:/home/Tai-20520924/Lab03/Bao Cao# ./StackGuard
Set guardsize to value of PAGESIZE.
Set guardsize is 1024
Retrieved guardsize is 1024
```

Hình 12 Chương trình minh hoạ việc get/set stackguardsize và kết quả thực thi

Giải thích kết quả:
“Set guardsize to value of PAGESIZE” thông báo rằng việc set thuộc tính guardsize vào đối tượng thuốc tính attr thành công

Retrieved guardsize is 1024 thông báo rằng việc get thuộc tính guardsize của đối tượng thuốc tính attr thành công và giá trị của guardsize là 1024

5. Setting /Getting the Scope

Hàm pthread_attr_setscope và pthread_attr_getscope được sử dụng để set/get thiết lập phạm vi tranh chấp của một thread, các phạm vì mà các hàm này trả về bao gồm

PTHREAD_SCOPE_SYSTEM hoặc PTHREAD_SCOPE_PROCESS

Với PTHREAD_SCOPE_SYSTEM có nghĩa là luồng này tranh chấp với tất cả các thread trong hệ thống

Với PTHREAD_SCOPE_PROCESS, có nghĩa là luồng này tranh chấp với các luồng khác có trong tiến trình.

```c
/*#######################################
# University of Information Technology #
# IT007.M21.1 Operating System    #
# <Nguyen Anh Tai>, <20520924> #
# File: example_Destroying_Attributes.c #
#######################################*/
#include <pthread.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
    int i, scope;
    pthread_attr_t attr;
    /* get the default attributes */
    pthread_attr_init(&attr);
    if (pthread_attr_setscope(&attr, PTHREAD_SCOPE_PROCESS) == 0)
        printf("success to set scope to PTHREAD_SCOPE_PROCESS \n");

    if (pthread_attr_getscope(&attr, &scope) == 0)
        if (scope == PTHREAD_SCOPE_PROCESS)
            printf("PTHREAD_SCOPE_PROCESS\n");
        else if (scope == PTHREAD_SCOPE_SYSTEM)
            printf("PTHREAD_SCOPE_SYSTEM\n");
        else
            printf("Illegal scope value.\n");
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

root@MSI:/home/Tai-20520924/Lab03/Bao Cao/pthread_attr# gcc example_Setting_Getting_the_Scope.c  -o Scope
root@MSI:/home/Tai-20520924/Lab03/Bao Cao/pthread_attr# ./Scope
success to set scope to PTHREAD_SCOPE_PROCESS
PTHREAD_SCOPE_PROCESS
```

Hình 13 Chương trình minh hoạ việc get/set phạm vi tranh chấp của thread và kết quả thực thi.

Giải thích kết quả:

“success to set scope to PTHREAD_SCOPE_PROCESS” thông báo rằng việc set phạm vi tranh chấp của thread thành PTHREAD_SCOPE_PROCESS thành công.
“PTHREAD_SCOPE_PROCESS” là kết quả của hàm getscope.

6. Setting Getting the Scheduling Policy

```c
/*#######################################
# University of Information Technology #
# IT007.M21.1 Operating System    #
# <Nguyen Anh Tai>, <20520924> #
# File: example_Destroying_Attributes.c #
#######################################*/
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
// Thư Viện POSIX chuẩn cup cấp các giá trị của thuộc tính scheduling policy bao gồm
// SCHED_FIFO(1) (first-in-first-out), SCHED_RR(2) (round-robin),
// SCHED_OTHER(0) (an implementation-defined method)
void *TestThread(void *arg)
{
    pthread_exit(NULL);
}
void *TestSchedulePolicy(pthread_attr_t attr)
{
    int foundschd;
    pthread_attr_getschedpolicy(&attr, &foundschd);

    switch (foundschd)
    // in ra màn hình các SCHED mà hàm pthread_attr_getschedpolicy trả về vào biến
foundschd
    {
    case SCHED_FIFO: // nếu Sheduling Policy là  FIFO(1) thì in ra màn hình FIFO
        printf("%d FIFO\n", foundschd);
        break;
    case SCHED_RR: // nếu Sheduling Policy là  RR(2) thì in ra màn hình RR
        printf("%d RR\n", foundschd);
        break;
    case SCHED_OTHER: // nếu Sheduling Policy là  OTHER(0) thì in ra màn hình OTHER
        printf("%d OTHER\n", foundschd);
        break;
    default:
```

```
        exit(2);
    }
}
int main(void)
{
    pthread_attr_t attr;
    pthread_t thid;

    int newshd;
    if (pthread_attr_init(&attr) == -1)
    // khởi tạo thuộc tính mặc định, nếu không thành công trả về 1
    {
        exit(1);
    }
    // Tạo một thread với thuộc tính mặc định
    pthread_create(&thid, &attr, TestThread, NULL);
    /*LƯU Ý: pthread_attr_setschedpolicy chỉ có tác dụng khi thuộc tính pthread_attr_t
được gán cho một thread*/

    // Sử dụng hàm pthread_attr_setschedpolicy để set thuộc tính schedpolicy thành
SCHED_FIFO
    if (pthread_attr_setschedpolicy(&attr, SCHED_FIFO) == 0)
        // Sử dụng hàm pthread_attr_getschedpolicy để get trạng thái khởi tạo của tiến
trình và gán vào foundschd
        TestSchedulePolicy(attr);

    // Sử dụng hàm pthread_attr_setschedpolicy để set thuộc tính schedpolicy thành
SCHED_RR
    if (pthread_attr_setschedpolicy(&attr, SCHED_RR) == 0)
        // Sử dụng hàm pthread_attr_getschedpolicy để get trạng thái khởi tạo của tiến
trình và gán vào foundschd
        TestSchedulePolicy(attr);

    // Sử dụng hàm pthread_attr_setschedpolicy để set thuộc tính schedpolicy thành
SCHED_OTHER
    if (pthread_attr_setschedpolicy(&attr, SCHED_OTHER) == 0)
        // Sử dụng hàm pthread_attr_getschedpolicy để get trạng thái khởi tạo của tiến
trình và gán vào foundschd
        TestSchedulePolicy(attr);
    exit(0);
}
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

root@MSI:/home/Tai-20520924/Lab03/Bao Cao/pthread_attr# gcc example_Setting_Getting_the_Scheduling_Policy.c  -o SchedulingPolicy -pthread
root@MSI:/home/Tai-20520924/Lab03/Bao Cao/pthread_attr# ./SchedulingPolicy
1 FIFO
2 RR
0 OTHER
```

Hình 14: kết quả thực thi việc get/set chính sách định thời của thuộc tính luồng attr

Giải thích kết quả:
"1 FIFO","2 RR","0 OTHER" là kết quả của chuỗi lệnh set thuộc tính shedpolicy thành các thuộc tính lần lượt là SCHED_FIFO(1) (định thời first-in-first-out), SCHED_RR(2) (định thời round-robin), SCHED_OTHER(0) (an implementation-defined method)

7. Setting Getting the Inherited Scheduling Policy

Các hàm pthread_attr_getinheritsched () và pthread_attr_setinheritsched () lấy và đặt thuộc tính kế thừa(inheritsched) cho đối số attr.

Khi các đối tượng thuộc tính được sử dụng bởi pthread_create (), thuộc tính kế thừa sẽ xác định cách đặt các thuộc tính lập lịch khác của luồng đã tạo.

Các giá trị của thuộc tính kế thừa(inheritsched) được xác định theo bảng sau :

PTHREAD_INHERIT_SCHED	Chỉ định rằng các thuộc tính lập lịch luồng được kế thừa từ luồng và các thuộc tính lập lịch trong đối số attr này bị bỏ qua.
PTHREAD_EXPLICIT_SCHED	Chỉ định rằng các thuộc tính lập lịch luồng được đặt thành các giá trị tương ứng từ đối tượng thuộc tính này.

```c
/*########################################
# University of Information Technology #
# IT007.M21.1 Operating System    #
# <Nguyen Anh Tai>, <20520924> #
# File: example_Setting_Getting _the_Inherited_Scheduling_Policy.c #
########################################*/
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

void *TestThread(void *arg)
{
    printf("hello from the thread\n");
    pthread_exit(NULL);
}
void *Test_Getinheritsched(pthread_attr_t attr)
{
    int foundschd;
```

```
     // Sử dụng hàm pthread_attr_getinheritsched để get trạng thái kế thừa của tiến trình và gán vào foundschd
    pthread_attr_getinheritsched(&attr, &foundschd);

    switch (foundschd)
    // in ra màn hình thuộc tính inheritsched mà hàm pthread_attr_getschedpolicy trả về vào biến foundschd
    {
    case PTHREAD_INHERIT_SCHED: // nếu thuộc tính inheritsched là  PTHREAD_INHERIT_SCHED thì in ra màn hình PTHREAD_INHERIT_SCHED
        printf("%d PTHREAD_INHERIT_SCHED\n", foundschd);
        break;
    case PTHREAD_EXPLICIT_SCHED:// nếu thuộc tính inheritsched là  PTHREAD_EXPLICIT_SCHED thì in ra màn hình PTHREAD_EXPLICIT_SCHED
        printf("%d PTHREAD_EXPLICIT_SCHED\n", foundschd);
        break;
    default:
        exit(2);
    }
}
int main(void)
{
    pthread_attr_t attr;
    pthread_t thid;

    int newshd;
    if (pthread_attr_init(&attr) == -1)
    // khởi tạo thuộc tính mặc định, nếu không thành công trả về 1
    {
        exit(1);
    }
    // Tạo một thread với thuộc tính mặc định
    pthread_create(&thid, &attr, TestThread, NULL);
    /*LƯU Ý: pthread_attr_setinheritsched chỉ có tác dụng khi thuộc tính pthread_attr_t được gán cho một thread*/

    // Sử dụng hàm pthread_attr_setinheritsched để set thuộc tính thành SCHED_FIFO
    pthread_attr_setinheritsched (&attr, PTHREAD_INHERIT_SCHED);
    Test_Getinheritsched(attr);

    // Sử dụng hàm pthread_attr_setinheritsched để set thuộc tính thành PTHREAD_EXPLICIT_SCHED
    pthread_attr_setinheritsched (&attr, PTHREAD_EXPLICIT_SCHED);
    Test_Getinheritsched(attr);
```

```
    exit(0);
}
```

```
root@MSI:/home/Tai-20520924/Lab03/Bao Cao/pthread_attr# gcc example_Setting_Getting_the_Inherited_Scheduling_Policy.c -o Inherited -pth
read
root@MSI:/home/Tai-20520924/Lab03/Bao Cao/pthread_attr# ./Inherited
0 PTHREAD_INHERIT_SCHED
1 PTHREAD_EXPLICIT_SCHED
```

Hình 15: kết quả sau khi thực thi chươgn trình minh hoạ việc get/set thuộc tính kế thừa của thread

Giải thích kết quả:
"0 PTHREAD_INHERIT_SCHED"," 1 PTHREAD_EXPLICIT_SCHED" là kết quả của chuỗi lệnh set/get thuộc tính kế thừa của đối tượng thuộc tính thành các giá trị lần lượt là PTHREAD_INHERIT_SCHED sau đó là PTHREAD_INHERIT_SCHED

8. Setting Getting the Scheduling Parameters

Hàm pthread_attr_getschedparam () và hàm pthread_attr_setschedparam () là hàm dùng get và set thuộc tính scheduling priority từ thuộc tính *attr* và lưu nó vào một *param*.

Lab03 > Bao Cao > pthread_attr > example_Setting_Getting_the_Scheduling_Parameters.c > main()

```c
/*#####################################
# University of Information Technology #
# IT007.M21.1 Operating System    #
# <Nguyen Anh Tai>, <20520924> #
# File: example_Setting_Getting_the_Scheduling_Parameters.c #
#####################################*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sched.h>
void *TestThread(void *arg)
{
    printf("hello from the thread\n");
    pthread_exit(NULL);
}
int main()
{
    pthread_attr_t attr;
    int rc;
    pthread_t thid;
    struct sched_param param;
    if (pthread_attr_init(&attr) == 0)
    {
        pthread_create(&thid, &attr, TestThread, NULL);
    }
    if (pthread_attr_getschedparam(&attr, &param) == 0)
   {
      printf("get schedpriority: %d\n", param.sched_priority);
   }
    int newprior = 1;
    param.sched_priority = newprior; // gán priority của biến param thành 0
   if (pthread_attr_setschedparam(&attr, &param) == 0 )
   {
      printf("set priority param to %d\n",param.sched_priority);
   }
   else printf("fail to set schedparam to %d\n",param.sched_priority);
    exit(0);
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
root@MSI:/home/Tai-20520924/Lab03/Bao Cao/pthread_attr# gcc example_Setting_Getting_the_Scheduling_Parameters.c -o SchedulingParameter -pthread
root@MSI:/home/Tai-20520924/Lab03/Bao Cao/pthread_attr# ./SchedulingParameter
get schedpriority: 0
fail to set schedparam to 1
```

Hình 16: Chương trình minh hoạ và kết quả thực thi việc set/get thuộc tính schedparam của đối tượng thuộc tính attr

Giải thích kết quả:
“get schedpriority: 0” thông báo rằng việc get độ thuộc tính độ ưu tiên của thread thành công và kết quả là 1

“fail to set schedparam to 1” thông báo rằng việc set độ ưu tiên cho thread thành 1 thất bại.

9. Setting Getting the Stack Size

```c
/*#######################################
# University of Information Technology #
# IT007.M21.1 Operating System    #
# <Nguyen Anh Tai>, <20520924> #
# File: example_Destroying_Attributes.c #
#######################################*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int main(void)
{
    pthread_attr_t attr;
    int rc;
    size_t size = 4096; // khởi tạo size_t = 1024

    if (pthread_attr_init(&attr) == -1)
    // nếu khởi tạo thuộc tính không thành công -> thoát
        exit(1);

    if (pthread_attr_setstacksize(&attr, size) == 0)
    // in ra màn hình kết quả nếu set stacksize thành công
        printf("Set stacksize to value%d: \n",(int) size);

    if (pthread_attr_getstacksize(&attr, &size) == 0)
      // in ra màn hình kết quả nếu get stacksize thành công
        printf("get stacksize: %d\n",(int)size);

    exit(0);
}
```

```
root@MSI:/home/Tai-20520924/Lab03/Bao Cao/pthread_attr# gcc example_Setting_Getting_the_Stack_Size.c -o StackSize -pthread
root@MSI:/home/Tai-20520924/Lab03/Bao Cao/pthread_attr# ./StackSize
get stacksize: 8388608
```

10. Setting Getting the Stack Address And Stack Size

Hàm pthread_attr_setstack () được sử dụng để đặt thuộc tính stack address và stack size của đối tượng thuộc tính luồng được tham chiếu bởi attr thành giá trị được chỉ định ở các đối số truyền vào.

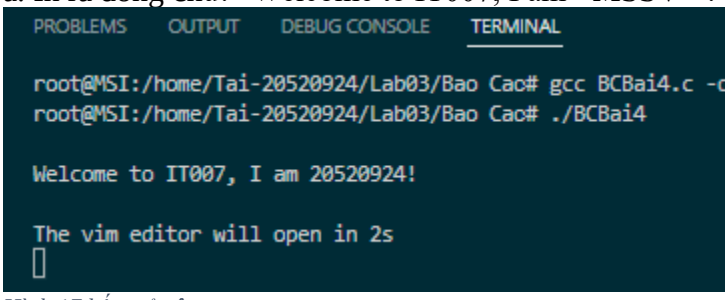
Hàm pthread_attr_getstackaddr () được sử dụng để lấy các thuộc tính stack address và stack size của đối tượng thuộc tính luồng và lưu vào các đối số truyền vào.

```c
/*#######################################
# University of Information Technology #
# IT007.M21.1 Operating System    #
# <Nguyen Anh Tai>, <20520924> #
# File: example_Setting_Getting_the_Stack_Address_And_Stack_Size.c #
#######################################*/

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

int main(void)
{
    pthread_attr_t attr;
    size_t stack_size = 4096;
    void *stack_addr = malloc(stack_size);
      // cấp phát địa cho biến stack_addr
    if (pthread_attr_init(&attr) == -1)
    {// khởi tạo giá trị mặc định thuộc tính attr
        exit(1);
    }
     if (pthread_attr_setstack(&attr, stack_addr, stack_size) == 0)
      // sử dụng hàm setstack để set các thuộc tính của đối tượng thuộc tính
      // nếu hàm trả về kết quả 0 thì có nghĩa là các giá trị của các đối số
      // stack_addr và stack_size truyền vào đã được đặt vào thuộc tính tương    // ứng
của thuộc tính đối tượng thuộc tính đối tượng attr.

    {
        printf("stackaddr set to: %p\n", stack_addr);
        printf("stacksize set to: %p\n", stack_size);
    }
    if (pthread_attr_getstack(&attr, &stack_addr, &stack_size) == 0)
      // sử dụng hàm setstack để set các thuộc tính của đối tượng thuộc tính
      // nếu hàm trả về kết quả 0 có nghĩa các đối số stack_addr và stack_size
      // truyền vào đã nhận được các giá trị tương ứng in ra màn hình thuộc tính //
stack size của đối tượng thuộc tính attr lúc này chỉ cần hiển thị kết      // quả ra màn
hình
      {
      printf("Retrieved stackaddr is %p\n", stack_addr);
      printf("Retrieved stacksize is %p\n", stack_size);
    }
```

```
    pthread_attr_destroy(&attr);
    exit(0);
}
```

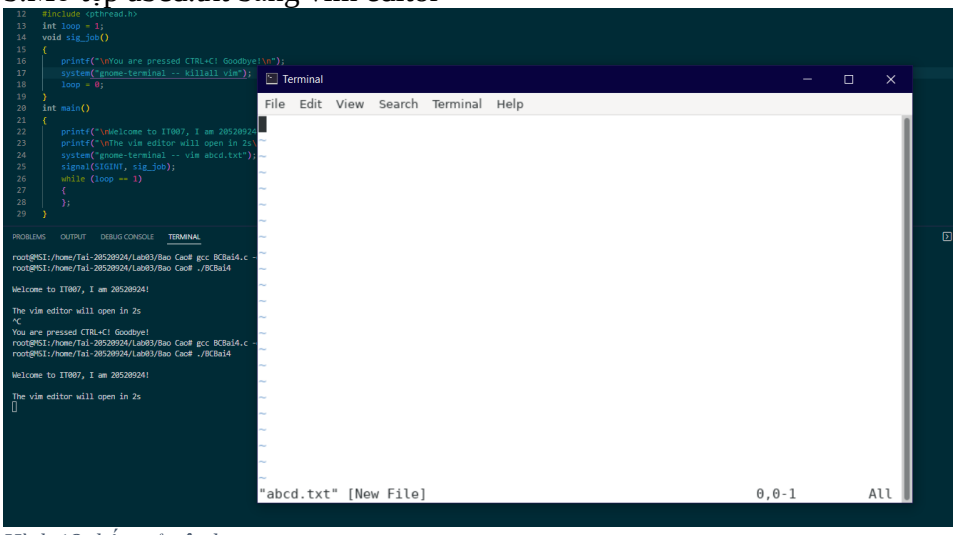
TASK 04: Viết chương trình làm các công việc sau theo thứ tự:

a. In ra dòng chữ: “Welcome to IT007, I am <MSSV> !”



Hình 17 kết quả câu a.

b.Mở tệp abcd.txt bằng vim editor



Hình 18: kết quả câu b

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!”

```
root@MSI:/home/Tai-20520924/Lab03/Bao Cao# ./BCBai4

Welcome to IT007, I am 20520924!

The vim editor will open in 2s
^C
You are pressed CTRL+C! Goodbye!
root@MSI:/home/Tai-20520924/Lab03/Bao Cao#
```

Hình 19: kết quả câu c và d

Chương trình

```c
/*#######################################
# University of Information Technology #
# IT007.M21.1 Operating System    #
# <Nguyen Anh Tai>, <20520924> #
# File: Task04.c #
#######################################*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
int loop = 1;
void sig_job()
{
    printf("\nYou are pressed CTRL+C! Goodbye!\n");
    system("gnome-terminal -- killall vim");
    loop = 0;
}
int main()
{
    printf("\nWelcome to IT007, I am 20520924!\n");
    printf("\nThe vim editor will open in 2s\n");
    system("gnome-terminal -- vim abcd.txt");
    signal(SIGINT, sig_job);
    while (loop == 1)
    {
    };
}
```

Giải thích: chương trình khi bắt đầu sẽ in ra 2 dòng

```
Welcome to IT007, I am 20520924!

The vim editor will open in 2s
```

Khi gặp

```
    system("gnome-terminal -- vim abcd.txt");
```

sẽ tạo mới một tiến trình và thực thi dòng lệnh bên trong “” kết quả của câu lệnh này sẽ mở tệp abcd.txt bằng trình soạn thảo vim

```
    signal(SIGINT, sig_job);
```

câu lệnh này sẽ tạo một handler để nhận tín hiệu Crt+C nhập vào từ người dùng
Khi người gửi tín hiệu Crt+C từ bàn phím handler sig_job sẽ được gọi hàm sig_job sẽ in ra màn hình “You are pressed CTRL+C! Goodbye!” và kết thúc tiến trình vim bằng lệnh killall vim