|  |
| --- |
| Name: NGUYỄN ANH TÀI  ID:20520924  Class: IT007.M22.1 |

OPERATING SYSTEM  
LAB 03’S REPORT

**SUMMARY**

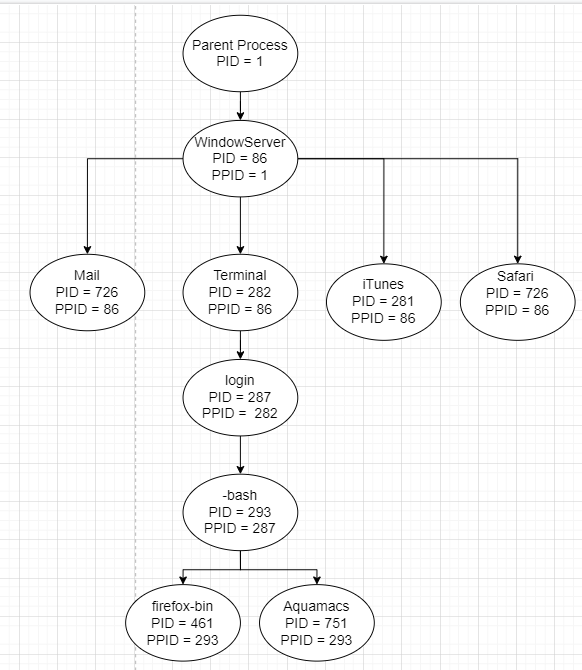
|  |  |  |
| --- | --- | --- |
| **Task** | **Status** | **Page** |
| 1. Mối quan hệ cha-con giữa các tiến trình | DONE | 1 |
| 2. Chương trình bên dưới in ra kết quả gì? Giải thích tại sao? | DONE | 6 |
| 3. Trong phần thực hành, các ví dụ chỉ sử dụng thuộc tính mặc định của pthread… | DONE | 10 |
| 4. Viết chương trình làm các công việc sau theo thứ tự… | DONE | 23 |

**Self-scrores: 9**

TASK 01: Mối quan hệ cha-con giữa các tiến trình

1. 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 |



1. 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:

Ảnh có chứa văn bản

Mô tả được tạo tự động

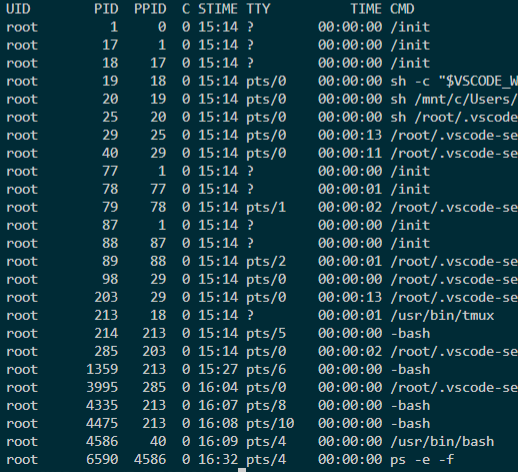
Ảnh có chứa văn bản, màn hình, ảnh chụp màn hình

Mô tả được tạo tự động



Hình : 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  
  


Hình 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

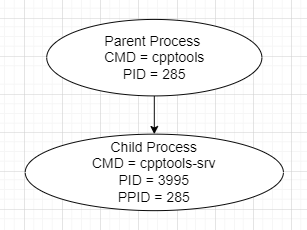


Hình Kết quả sau khi thực hiện lệnh ps --pid 3995

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:



1. 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]  


Hình kết quả sau khi chạy lệnh whereis pstree

kết quả trả về đường dẫn nơi pstree được cài đặt

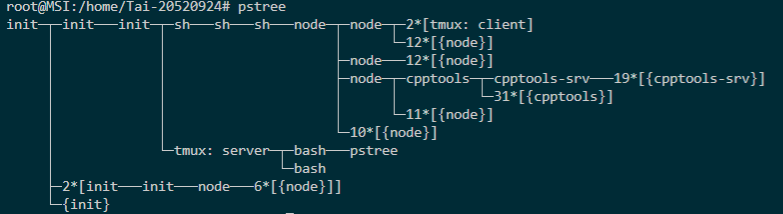
=> lệnh pstree đã được cài đặt trong HDH

Tìm hiểu lệnh pstree bằng man pstree. Ảnh có chứa văn bản

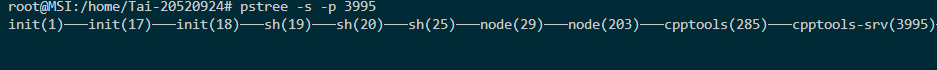
Mô tả được tạo tự động

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

Tiến hành sử dụng pstree



Hình 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?**

Ảnh có chứa văn bản

Mô tả được tạo tự động

Chương trình vẫn chưa chạy được vì thiếu thư viện Ảnh có chứa văn bản

Mô tả được tạo tự động

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

Tiến hành thực thi chương trình  
Ảnh có chứa văn bản

Mô tả được tạo tự động

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  
Ảnh có chứa văn bản

Mô tả được tạo tự động

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)  
Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình 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.**

## Initializing Attributes

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình 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 in 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 size. |
| *priority* | 0 | New thread has priority 0. |
| *inheritsched* | PTHREAD\_EXPLICIT\_SCHED | New thread does not inherit parent thread scheduling priority. |
| *schedpolicy* | SCHED\_OTHER | New thread uses the traditionalSolaris time-sharing (TS) scheduling class. |
| *guardsize* | PAGESIZE | Stack overflow protection. |

## Destroying Attributes

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : 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.

## 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.

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : 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 ()

## 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

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình 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

## 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.

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình 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.

## Setting Getting the Scheduling Policy

*/\*######################################*

*# 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 Viên 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*

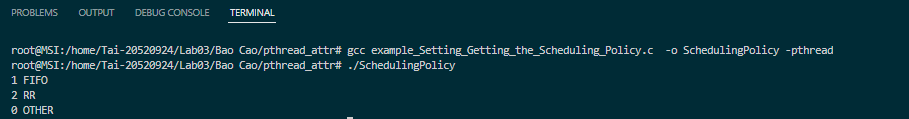
    if (pthread\_attr\_setschedpolicy(&attr, SCHED\_OTHER) == 0)

*// Sử dụng hàm pthread\_attr\_getschedpolicy để get trạng thái khởi tạo của tiến trình và gán vào foundschd*

        TestSchedulePolicy(attr);

    exit(0);

}



Hình : 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)

## 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. |

*/\*######################################*

*# 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);

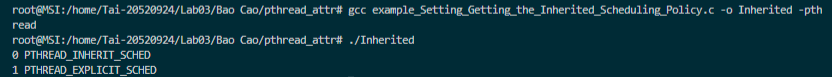
*// Sử dụng hàm pthread\_attr\_setinheritsched để set thuộc tính thành PTHREAD\_EXPLICIT\_SCHED*

    pthread\_attr\_setinheritsched (&attr, PTHREAD\_EXPLICIT\_SCHED);

    Test\_Getinheritsched(attr);

    exit(0);

}



Hình : 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

## 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*.

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : 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.

## Setting Getting the Stack Size

*/\*######################################*

*# 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);

}

Ảnh có chứa văn bản

Mô tả được tạo tự động

## 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.

*/\*######################################*

*# 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> !”

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình 17 kết quả câu a.

b.Mở tệp abcd.txt bằng vim editor

Ảnh có chứa văn bản

Mô tả được tạo tự động

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!”

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình 19: kết quả câu c và d

Chương trình

*/\*######################################*

*# 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

Ảnh có chứa văn bản

Mô tả được tạo tự động

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