FACULTY OF INFORMATION TECHNOLOGY

>>> COURSE MATERIAL <<<

INTRODUCTION TO OPERATING SYSTEM

Course ID 502047

Lab Part 3 – Process management

Cập nhật lần cuối: 22/6/202

Mục tiêu	**Lý thuyết liên quan**	**Tài nguyên**
Tạo tiến trình	Ch3: Process Creation	Sử dụng image Ubuntu 14 / 16
Huỷ tiến trình	Ch3: Process Creation	
Lệnh exec	Ch2: Compile and Linking	
Lệnh system	Ch2: Compile and Linking	
Programming Problems	Ch2: OS structure	Github
Programming Problems	Ch3: Process	

Yêu cầu nộp bài: các tập tin mã nguồn .c và tập tin khả thực thi .out của các "ví dụ" và bài tập cuối hướng dẫn.

Preferences

[1] Abraham Silberschatz, Peter B. Galvin, Greg Gagne, [2018], Operating System Concepts, 10th edition, John Wiley & Sons, New Jersey.

Programming Problems of Chapter 3.

[2] Nguyễn Hồng Vũ, [2016] Hướng dẫn LAB Hệ điều hành, khoa Công nghệ thông tin, ĐH Tôn Đức Thắng.

[3] Greg Gagne , [2019], GitHub OS-BOOK OSC10e, Westminster College, United States

Access https://github.com/greggagne/osc10e in September 2019.

[4] Zombie and Orphan Processes in C, *geeksforgeeks.org*

Tìm hiểu quản lý tiến trình trên Linux

Trên hệ điều hành Linux, tiến trình được nhận biết thông qua số hiệu (định danh) tiến trình **pid.** Một tiến trình có thể nằm trong một nhóm nào đó, có thể nhận biết thông qua số hiệu nhóm **pgrp.** Một số hàm của C cho phép lấy các thông số này:

int getpid() : trả về giá trị int là pid của tiến trình hiện tại

int getppid() : trả về giá trị int là pid của tiến trình cha của tiến trình hiện tại

int getpgrp() : trả về giá trị int là số hiệu của nhóm tiến trình

int setpgrp() : trả về giá trị int là số hiệu nhóm tiến trình mới tạo ra

Hiển thị thông tin tiến trình – lệnh ps, pstree

Để biết thông tin các tiến trình hiện hành ta sử dụng lệnh: **ps [option]**

-e: hiển thị thông tin về mỗi tiến trình.
-l: hiển thị thông tin đầy đủ tiến trình.
-f: hiển thị thông tin về tiến trình cha.
-a: hiển thị tất cả các tiến trình.

1. Tạo tiến trình

Thư viện "unistd.h" chứa các lời gọi hệ thống tạo tiến trình fork() và xem ID của chúng getpid() và xem ID của tiến trình cha getppid().

1.1. Xem thông tin số hiệu tiến trình.

Ví dụ 1.1 : Tạo tập tin P1.c chứa hàm main như sau, sau đó biên dịch và cho biết ID của tiến trình này cùng với ID của tiến trình cha.

```
1      // P1.c
2      #include <stdio.h>
3      #include <unistd.h>
4      int main() {
5          printf("Current process ID: %d\n", getpid());
6          printf("Parent process ID: %d\n", getppid())
```

```
7           return 0;
8       }
```

Trong tiến trình này, sử dụng các hàm **getpid()**, **getppid()** để lấy định danh của tiến trình đang chạy (chứa lời gọi hàm) và định danh của tiến trình cha của nó.

Các định danh tiến trình là tham số quan trọng để cho phép các tiến trình giao tiếp với nhau qua cơ chế signal.

Ví dụ 1.2: Tạo tập tin P2.c chứa hàm main như sau: sau đó biên dịch và chạy để xem kết quả. Có thể thêm vài lệnh fork() và xem kết quả thay đổi ra sao.

```
        // P2.c
1       #include <stdio.h>
2       #include <sys/types.h>
3       #include <unistd.h>
4       int main()
5       {   // make two process which run same
6           // program after this instruction
7         // Thêm một vài lệnh printf và theo dõi.
8           fork();
9           // thêm một vài lệnh fork vào đây và xem thay đổi.
10          printf("Hello world! from Process ID: %d\n", getpid());
11          return 0;
        }
```

Hàm tạo tiến trình pid_t fork(void) được sử dụng với lời gọi pid_t pid=fork():

Nếu thành công và pid =0: đang trong thân process con.

Nếu thành công và pid>0: xử lý trong thân process cha.

Nếu thất bại: pid=-1 kèm lý do, ví dụ như ENOMEM: không đủ bộ nhớ, hay là EAGAIN: số tiến trình vượt quá giới hạn cho phép.

Cấu trúc của một chương trình thông thường như sau:

```
1     // P_template.c
2     #include <stdio.h>
3     #include <unistd.h>
4     int main()
5     {   pid_t pid;
6         pid = fork();
7         if(pid < 0) printf("fork error! No child created\n");
8         else if(pid == 0)  {//code of parent process  }
9         else { // code of the new child process }
10        return 0;
11    }
```

Ví dụ 1.3: Tạo tập tin P3.c chứa hàm main như sau, sau đó biên dịch chạy. Lưu ý thứ tự các câu in ra màn hình có thể thay đổi khác nhau ở các lần chạy, hãy giải thích tại sao?

```
1     // P3.c
2     #include <stdio.h>
3     #include <unistd.h>
4     int main()
5     {   pid_t pid;
6         pid = fork();
7         if(pid < 0) printf("fork error! No child created\n");
8         else if(pid == 0)  {
9             printf("Hello from Child!\n");
10            printf("Child ID is %d!\n", getpid());
11.       }
12        else {
13           printf("Hello from Parent!\n");
14            printf("Parent ID is %d!\n", getpid());
15        }
16        return 0;
```

```
17      }
```

2. Kết thúc tiến trình

- Dùng hàm exit()

- Orphaned process: process cha kết thúc trước; process con sau đó sẽ có cha là init (PID=1)

- Zombie process

  - Tiến trình con kết thúc nhưng chưa báo trạng thái cho tiến trình cha.
  - Dùng hàm wait() hoặc waitpid() ở tiến trình cha để lấy trạng thái trả về từ tiến trình con.

Ví dụ 2.1 Hãy biên dịch và thực thi các tập tin zombie.c và orphan.c được cung cấp và dùng lời gọi ps rồi tìm chúng trong bảng tiến trình.

Lưu ý 1: Chúng ta có thể chạy một chương trình ở chế độ "background" bằng cách thêm "&" vào sau lời gọi.

```
> ./main.out &
```

Khi đó main.out thực thi bên dưới và chúng ta có thể gõ lệnh hay chạy một tiến trình khác ở bên trên terminal.

Lưu ý 2: Khi một tiến trình đang thực thi, chúng ta có thể đưa tiến trình này vào "background" thông qua việc gửi tín hiệu SIGTSTP bằng cách nhấn Ctrl + Z, khi đó chúng ta có thể gõ lệnh hoặc chạy một tiến trình khác.

- Để xem danh sách tiến trình kèm thông tin, sử dụng

```
> ps
> ps -l -y  //long format and not show flags
```

- Sử dụng thư viện và các hàm

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```

Tất cả các lời gọi hệ thống này được sử dụng để chờ sự thay đổi trạng thái trong một tiến trình con (của tiến trình đang thực hiện lời gọi) và lấy thông tin về tiến trình con đó. Sự thay đổi trạng thái có thể là: tiến trình con chấm dứt thực thi; tiến trình con bị chặn lại bởi một tín hiệu; hoặc tiến trình con được phục hồi bởi một tín hiệu. Trong trường hợp một tiến trình bị chấm dứt thực thi, việc gọi lệnh wait() cho phép hệ thống giải phóng các tài nguyên liên quan đến tiến trình con; nếu lệnh wait() không được thực hiện, thì tiến trình con bị chấm dứt vẫn ở trạng thái "zombie".

Nếu một tiến trình con đã thay đổi trạng thái, những lời gọi hệ thống này trả về kết quả ngay lập tức. Mặt khác, chúng chặn cho đến khi một tiến trình con thay đổi trạng thái hoặc bộ xử lý tín hiệu làm gián đoạn lời gọi.

- wait() và waitpid() khác nhau thế nào?

Cuộc gọi hệ thống wait () tạm dừng thực thi tiến trình gọi cho đến khi một trong các con của nó kết thúc. Lời gọi wait(&status) tương đương với:

```
wait(&status); //the same with
waitpid(-1, &status, 0);
```

Lời gọi hệ thống waitpid() tạm dừng thực thi tiến trình gọi cho đến khi một tiến trình con - được chỉ định bởi đối số pid - đã thay đổi trạng thái. Mặc định, waitpid() chỉ đợi tiến trình con bị chấm dứt, tuy nhiên có thể dùng đối số tùy chọn để thay đổi, như được mô tả dưới đây.

- Trả về ID của tiến trình con thay đổi trạng thái. Trả về -1 nếu thất bại.

  Đối số pid của hàm:

  - <-1: đợi bất kỳ tiến trình nào có groupID bằng với group của |pid|.
  - -1: đợi bất kỳ tiến trình nào.
  - 0: đợi bất kỳ tiến trình con nào có GroupID bằng với Group ID của tiến trình gọi.
  - >0: Đợi tiến trình có ID bằng với pid.

Ví dụ 2.2 sau đây tạo ra một số lượng tiến trình con (được truyền vào qua đối số hàm main), in ra ID của các tiến trình con và lời gọi chờ của tiến trình cha.

```
1  #include <sys/wait.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <stdio.h>
5
6  int main(int argc, char *argv[])
7  {
8      int pnum, count, retval, child_no;
9      pnum = atoi(argv[1]);
```

```
10      if(pnum <= 0) {
11          printf("So tien trinh con phai lon hon 0");
12          return -1;
13      }
14      else {
15          retval=1;
16          for(count=0, count<pnum; count++) {
17              if(retval!=0) retval=fork();
18              else break;
19          }
20          if(retval == 0) {
21              child_no = count;
22              printf("Tien  trinh  %d,  PID  %d\n",child_no,
23  getpid());
24          } else {
25          for(count=0; count<pnum; count++) wait(NULL);
26          printf("Tien trinh cha PID %d", getpid());
27          }
28      }
29      wait();  // chờ một trong những đứa con.
30      return 0;
    }
```

Ngoài cách sử dụng lời gọi wait(), chúng ta có thể bỏ qua tín hiệu SIGCHLD.

Khi một tiến trình con bị chấm dứt, tín hiệu SIGCHLD tương ứng sẽ được gửi đến tiến trình cha, nếu chúng ta thực hiện lời gọi "(SIGCHLD, SIG_IGN)", thì sau đó tín hiệu SIGCHLD sẽ bị hệ thống bỏ qua và chỉ mục của tiến trình con bị xóa khỏi bảng tiến trình . Do đó, không có zombie nào được tạo ra. Tuy nhiên, trong trường hợp này, tiến trình cha không thể biết về tình trạng thoát ra của tiến trình con.

Ví dụ 2.3:

```
1   // A C program to demonstrate ignoring
2   // SIGCHLD signal to prevent Zombie processes
3   #include<stdio.h>
4   #include<unistd.h>
5   #include<sys/wait.h>
6   #include<sys/types.h>
7
8   int main()
```

```
9   {
10      int i;
11      int pid = fork();
12      if (pid == 0)
13           for (i=0; i<20; i++)
14                printf("I am Child\n");
15      else
16      {
17           signal(SIGCHLD,SIG_IGN);
18           printf("I am Parent\n");
19           while(1);
20      }
21  }
```

3. Các lệnh liên quan tiến trình

- Liệt kê tiến trình:

```
>  ps [options]
```

- Các option:

- -a: hiển thị các tiến trình của user được liên kết
- -e: hiển thị tất cả tiến trình
- -f: hiển thị PID của tiến trình cha và thời điểm bắt đầu
- -l: tương tự -f

- Hủy tiến trình:

```
> kill [-signal | -s signal] pid
```

Thường kết hợp với lệnh ps để lấy ID tiến trình

- Các signal:

- 2: tương đương CTRL + C
- 9: buộc kết thúc
- 15: mặc định – kết thúc êm ái
- 19: tạm dừng
- Kill – l: liệt kê tất cả các signal

- Ví dụ:

kill -9 11234 (kill tiến trình có pid = 11234)

```
kill -s 9 11234
```

4. Các hàm gọi thực thi chương trình

*int system(const char *string);*

*int execl(const char *path, const char *arg, ...)*

*int execv(const char *path, const char *argv[ ])*

*int execlp(const char *lename, const char *arg, ...)*

*int execvp(const char *lename, const char *argv[ ])*

*int execle(const char *path, const char *arg, ...,const char **env)*

*int execve(const char *path, const char *argv[ ], const char **env)*

5. Hàm system

- Cú pháp: *int system(const char *string);*

- Thực thi lệnh trong đối số string và trả về kết quả khi thực hiện lệnh xong. Khi gọi hàm system(string), hệ thống sẽ thực hiện lệnh sh -c string.

- Giá trị trả về:

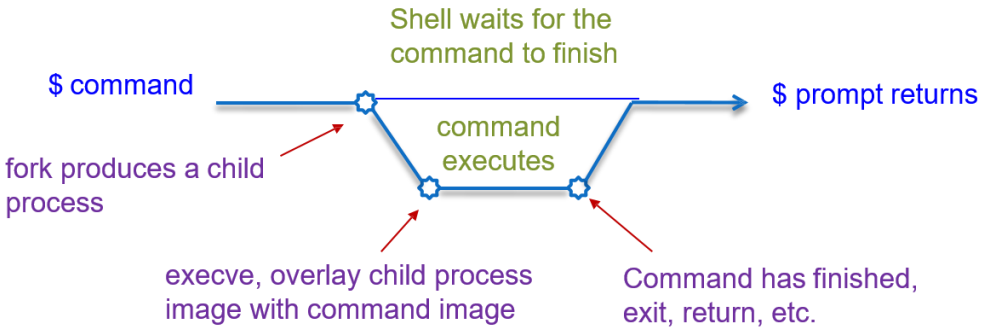
- 0: thành công
- -127: Không khởi động shell để thực hiện lệnh
- -1: lỗi khác
- -1: mã trả về khi thực hiện lệnh string.

Ví dụ:

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[])
4  {
5      int re;
6      printf("Call system to execute ls -a\n";
7      re=system("ls -a");
8      if(re != -1) printf("System call ls is done!\n");
```

```
9
10      printf("Call system to execute ps -a\n";
11      re=system("ps -a");
12      if(re != -1) printf("System call ps is done!\n");
13
14      return 0;
15  }
```

- Các hàm exec... Thực hiện theo cơ chế sau:



- Các hàm exec… sẽ thay thế tiến trình gọi hàm bằng chương trình tương ứng trong tham số nhập của hàm. Vùng text, data, stack bị thay thế.

- Chương trình được gọi bắt đầu thực thi ở hàm main, có thể nhận tham số nhập thông qua các tham số truyền.

Vấn đề lập trình

Xin xem file "Exercise 03A.pdf" và trả lời các câu hỏi trong file đó.

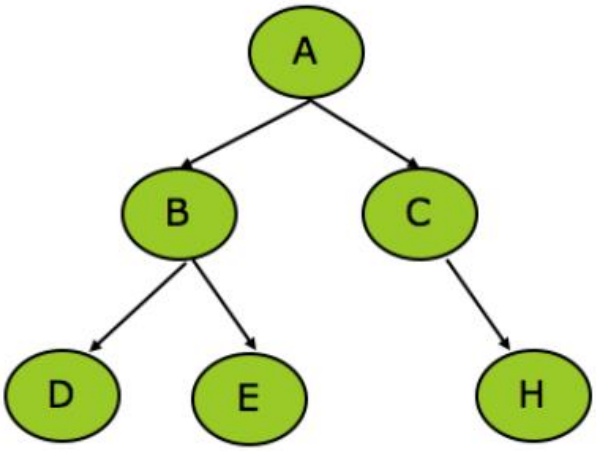
Bài tập lập trình

1. Viết chương trình để truyền đối vào số nguyên dương n vào và
   - Tiến trình cha tiếp tục tính rồi xuất ra tổng S = 1 + 2 + …. + n
   - Đồng thời tạo một tiến trình con tính tổng các ước số của n và in ra màn hình.

   Hãy sử dụng lời gọi wait() để chắc chắn rằng tiến trình con không trở thành một tiến trình mồ côi. Nếu không sử dụng lời gọi wait() thì rủi ro nào có thể xảy ra cho chương trình này?

2. Sử dụng lời gọi system() để viết chương trình thực thi các lệnh Linux đã học tại LAB01.

3. Tạo ra cây tiến trình như sau, với mỗi tiến trình con, in ra ID của nó và ID của tiến trình cha.



4. Viết một chương trình lặp vô tận với lời gọi while(1); và thực thi nó. Đưa tiến trình này vào "background" thông qua việc gửi tín hiệu SIGTSTP bằng cách nhấn Ctrl + Z. Sử dụng lệnh ps để xác định PID của nó và sử dụng kill để kết thúc nó.

5. Programming Problems 3.18 in book [1]
   Viết chương trình mà khi chạy nó sinh ra tiến trình con, để tiến trình con trở thành zombie. Tiến trình zombie này cần tồn tại trong hệ thống tối thiểu 10 giây (bằng cách dùng lời gọi sleep(10)). Sau đó dùng lệnh ps –l để xem trạng thái các tiến trình. Kết liễu zombie này bằng cách xác định PID của tiến trình cha nó và sử dụng lệnh kill.

6. Programming Problems 3.19 in book [1]

   Viết chương trình xác định thời gian cần thiết để thực thi một lệnh, lệnh này truyền vào qua phần đối số như minh hoạ dưới đây. Hàm gettimeofday() có thể dùng để tính thời gian. Thân hàm chính nên được thiết kế như sau:

   - Lấy thời gian hiện tại (bắt đầu)
   - fork() để tạo tiến trình con và tiến trình con dùng system() để thực thi lệnh truyền vào.
   - Khi tiến trình con kết thúc, tiến trình cha đợi bằng lời gọi wait().
   - Lấy thời gian hiện tại (kết thúc) và tính ra thời gian đã trôi qua (kết thúc – bắt đầu).

Ví dụ để đo thời gian lệnh ls thực thi

```
>./time ls
 time.c
 time
```

```
Elapsed time: 0.25422
```

7. Programming Problems 3.21 in book [1]

Phỏng đoán Collatz[1] tin rằng dãy số sinh ra sẽ luôn tiến về 1 với bất kỳ số nguyên dương nào được tạo ra ở bước đầu. Dãy số được tạo ra theo giải thuật sau:

$$n = \begin{cases} \frac{n}{2} & nếu\ n\ là\ số\ chẵn \\ 3 * n + 1 & nếu\ \ là\ số\ lẻ \end{cases}$$

Ví dụ với n=35, dãy số sinh ra là 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

Viết một chương trình nhận số nguyên dương n vào thông qua đối số, kiểm tra tính đúng của giá trị này. Tạo ra một tiến trình con để tính và in ra dãy số, tronng lúc tiến trình cha cần chờ tiến trình con hoàn thành thông qua lời gọi wait().

```
>./collazt 35
 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1
 Ket thuc tien trinh con
```

[1] https://vi.wikipedia.org/wiki/Ph%E1%BB%8Fng_%C4%91o%C3%A1n_Collatz

CHECKLIST LAB

- ❑ Khởi động Ubuntu.
- ❑ Chạy Terminal.
- ❑ Biên tập một tập tin văn bản .txt, .c
- ❑ Biên dịch và Liên kết một mã nguồn .c
- ❑ Tạo thư viện liên kết tĩnh.
- ❑ Tạo thư viện liên kết động.
- ❑ Chạy một chương trình đã biên dịch.
- ❑ Truyền đối số qua dòng lệnh.
- ❑ Tạo tiến trình con.
- ❑ Cấu trúc mã nguồn cho tiến trình cha và tiến trình con.
- ❑ Tạo cây tiến trình
- ❑ Đồng bộ kết thúc tiến trình cha/con.
- ❑ Gọi chạy một lệnh từ trong chương trình.