







>>> COURSE MATERIAL <<<

INTRODUCTION TO OPERATING SYSTEM / Course ID 502047

BÀI TẬP CHƯƠNG 3 – TIỀN TRÌNH

Hướng dẫn tự học, chỉ sử dụng để tham khảo và có thể thay đổi mà không báo trước.

	Programming Exercises	12-----3-6-8901
	Exam multichoice questions	---45--89-12----7----
	Discussion questions	--3-----0-----
	Just for reference	-----67-----45-----

3.1 Khi chạy đoạn mã sau, nội dung gì được in ra màn hình bởi dòng lệnh A?

01	#include <sys/types.h>
02	#include <stdio.h>
03	#include <unistd.h>
04	int value = 5;
05	int main()
06	{
07	pid_t pid;
08	pid = fork();
09	if (pid == 0) { /* child process */
10	value += 15;
11	return 0;

12	}
13	else if (pid > 0) { /* parent process */
14	wait(NULL);
15	printf("PARENT: value = %d",value); /* LINE A */
16	return 0;
17	}
18	}

3.2 Có bao nhiêu tiến trình được tạo ra sau khi chạy đoạn chương trình dưới đây, bao gồm cả tiến trình cha ban đầu?

01	#include <stdio.h>
02	#include <unistd.h>
03	int main()
04	{
05	/* fork a child process */
06	fork();
07	/* fork another child process */
08	fork();
09	/* and fork another */
10	fork();
11	return 0;
12	}

3.3 Phiên bản gốc của iOS (Apple) không có khái niệm xử lý đồng thời. Hãy thảo luận 3 khó khăn khi đưa yêu cầu xử lý đồng thời vào một hệ điều hành.

3.4 Một số hệ thống máy tính cung cấp nhiều bộ thanh ghi. Hãy mô tả lại quá trình chuyển ngữ cảnh với một hệ thống như vậy. Hãy thảo luận về các vấn đề:

- Nó hoạt động ra sao, cần thêm những thông tin điều khiển gì?
- Điều gì sẽ xảy ra nếu chuyển ngữ cảnh được tiến hành mà ngữ cảnh mới đã nằm trong một tập thanh ghi nào đó.
- Điều gì sẽ xảy ra nếu ngữ cảnh đang lưu ở bộ nhớ mới hơn ngữ cảnh ở tập thanh ghi và tất cả tập thanh ghi đã được sử dụng?

3.5 Khi một tiến trình tạo ra một tiến trình mới bằng cách sử dụng lệnh fork(), những trạng thái nào sau đây được chia sẻ giữa tiến trình cha và tiến trình con?

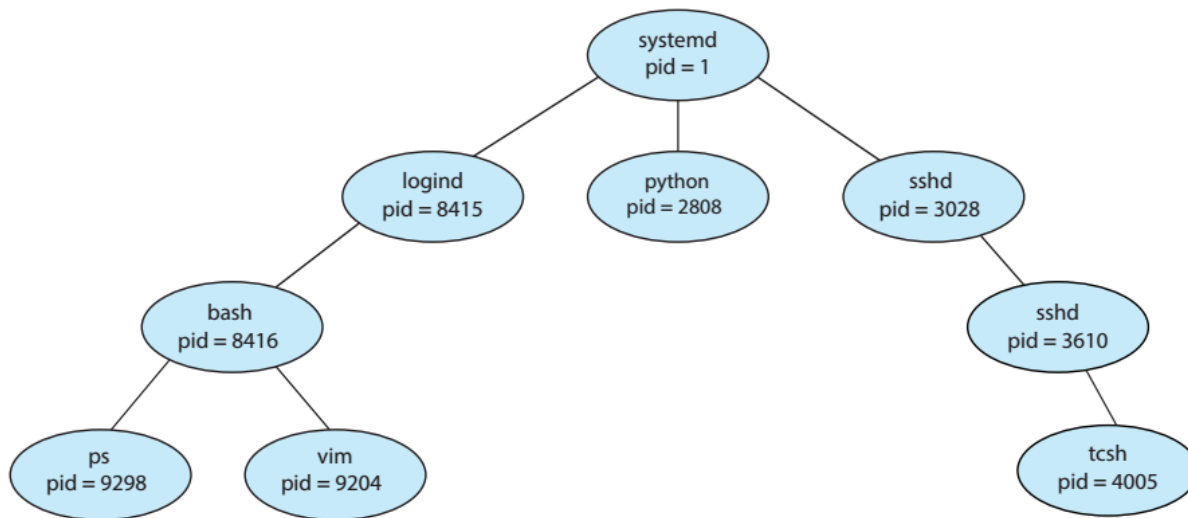
- a. Stack
- b. Heap
- c. Shared memory segments (phần đoạn bộ nhớ chia sẻ)

3.6 Hãy xem xét các ngữ nghĩa “Chính xác một lần” đối với cơ chế RPC. Liệu thuật toán để thực hiện ngữ nghĩa này có thực hiện đúng không ngay cả khi thông điệp ACK gửi lại cho máy khách bị mất do sự cố mạng? Mô tả trình tự các thông điệp, và thảo luận ngữ nghĩa “Chính xác một lần” vẫn còn được bảo tồn hay không?

3.7 Giả sử rằng một hệ thống phân tán dễ bị lỗi máy chủ. Cơ chế gì cần thiết để bảo vệ ngữ nghĩa “Chính xác một lần” để thực hiện các RPC?

3.8 Nhân hệ điều hành cần phải làm những thao tác gì khi chuyển ngữ cảnh giữa các tiến trình.

3.9 Tạo ra cây tiến trình như hình sau



Để có được thông tin tiến trình trong hệ thống UNIX hoặc Linux, sử dụng lệnh `ps -aef`. (Sử dụng lệnh man `ps` để có thêm thông tin về lệnh `ps`). Trong Windows, ID của tiến trình cha không thể hiện, tuy nhiên công cụ theo dõi tiến trình¹ có cung cấp công cụ cây tiến trình.

3.10 Giải thích vai trò của tiến trình `init` (hoặc `systemd`) trên UNIX và các hệ thống Linux liên quan đến việc chấm dứt một tiến trình.

3.11 Có bao nhiêu tiến trình được tạo ra khi thực thi đoạn mã dưới đây, bao gồm cả tiến trình ban đầu?

```
01 #include <stdio.h>
02 #include <unistd.h>
03 int main()
04 {
05     int i;
06     for (i = 0; i < 4; i++)
07         fork();
08     return 0;
09 }
```

3.12 Trong các trường hợp nào, dòng mã J trong đoạn chương trình dưới đây sẽ được thực thi? Giải thích.

```
01 #include <sys/types.h>
02 #include <stdio.h>
03 #include <unistd.h>
04 int main()
05 {
06     pid_t pid;
07     /* fork a child process */
08     pid = fork();
09     if (pid < 0) { /* error occurred */
10         fprintf(stderr, "Fork Failed");
11         return 1;
12     }
13     else if (pid == 0) { /* child process */
14         execlp("/bin/ls", "ls", NULL);
```

¹ <https://docs.microsoft.com/en-us/sysinternals/downloads/procmon>

15	printf("LINE J");
16	}
17	else { /* parent process */
18	/* parent will wait for the child to complete */
19	wait(NULL);
20	printf("Child Complete");
21	}
22	return 0;
23	}

3.13 Cho biết giá trị của PID tại các dòng A, B, C, D là bao nhiêu khi chạy đoạn chương trình dưới đây. Giả sử rằng giá trị PID thực tế của tiến trình cha và con lần lượt là 2600 và 2603.

01	#include <sys/types.h>
02	#include <stdio.h>
03	#include <unistd.h>
04	int main()
05	{
06	pid_t pid, pid1;
07	/* fork a child process */
08	pid = fork();
09	if (pid < 0) { /* error occurred */
10	fprintf(stderr, "Fork Failed");
11	return 1;
12	}
13	else if (pid == 0) { /* child process */
14	pid1 = getpid();
15	printf("child: pid = %d",pid); /* A */
16	printf("child: pid1 = %d",pid1); /* B */
17	}
18	else { /* parent process */
19	pid1 = getpid();
20	printf("parent: pid = %d",pid); /* C */
21	printf("parent: pid1 = %d",pid1); /* D */
22	wait(NULL);
23	}
24	return 0;
25	}

3.14 Cho một ví dụ về tình huống trong đó các đường ống thông thường [ordinary pipe] phù hợp hơn các ống được đặt tên [named pipe] và một ví dụ về tình huống trong đó các ống được đặt tên phù hợp hơn các ống thông thường.

3.15 Xem xét cơ chế RPC. Mô tả những hậu quả không mong muốn có thể phát sinh từ việc không thực thi cả hai ngữ nghĩa “tối thiểu một lần” và cả “chính xác một lần. Mô tả các cách sử dụng có thể cho một cơ chế không có những đảm bảo này.

3.16 Sử dụng chương trình sau đây, cái gì sẽ in ra sẽ ở dòng X và Y.

01	#include <sys/types.h>
02	#include <stdio.h>
03	#include <unistd.h>
04	#define SIZE 5
05	int nums[SIZE] = {0,1,2,3,4};
06	int main()
07	{

```

08  int i;
09  pid_t pid;
10  pid = fork();
11  if (pid == 0) {
12      for (i = 0; i < SIZE; i++) {
13          nums[i] *= -i;
14          printf("CHILD: %d ", nums[i]); /* LINE X */
15      }
16  }
17  else if (pid > 0) {
18      wait(NULL);
19      for (i = 0; i < SIZE; i++)
20          printf("PARENT: %d ", nums[i]); /* LINE Y */
21  }
22  return 0;
23  }

```

3.17 Những lợi ích và nhược điểm của mỗi điều sau đây là gì? Xem xét ở cả cấp độ hệ thống và cấp độ lập trình viên.

- Giao tiếp đồng bộ và không đồng bộ
- Bộ đệm tự động (Automatic buffering) và rõ ràng (explicit buffering).
- Send by copy và send by reference
- Tin nhắn có kích thước cố định và kích thước tùy biến.

3.18 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`.

3.19 Viết chương trình xác định thời gian cần thiết để thực thi một lệnh, lệnh này truyền vào qua phần đối số như minh họa 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

```

3.20 Bộ quản lý tiến trình của một hệ điều hành có trách nhiệm định danh. Khi một tiến trình được tạo lần đầu tiên, nó được gán một pid duy nhất bởi bộ quản lý pid. Pid được trả lại cho bộ quản lý pid khi tiến trình khi nó hoàn tất thực thi và bộ quản lý sau đó có thể dùng lại giá trị pid này. Các con số định danh tiến trình được thảo luận đầy đủ hơn trong Phần 3.3.1. Điều gì quan trọng nhất ở đây là nhận ra rằng các định danh quy trình phải là độc nhất; không có hai quá trình hoạt động có thể có cùng pid.

Sử dụng các hằng số sau để xác định phạm vi của pid có thể các giá trị:

```
#define MIN_PID 300
#define MAX_PID 5000
```

Bạn có thể sử dụng bất kỳ cấu trúc dữ liệu nào để thể hiện khả năng sẵn có của các mã định danh tiến trình. Một chiến lược Linux đang thực hiện là sử dụng một bitmap trong đó giá trị 0 tại vị trí i xác định rằng một id tiến trình có giá trị i sẵn sàng để cấp và giá trị 1 chỉ ra rằng nó đang được một tiến trình sử dụng.

Thực hiện API sau đây để cấp phát và trả lại pid:

- `int allocate_map (void)` Tạo ra và khởi tạo cấu trúc dữ liệu để biểu diễn cho pids; trả về -1 nếu không thành công, 1 nếu thành công
- `int allocate_pid (void)` – Cấp phát và trả về một pid; trả về -1 nếu không thể phân bổ pid (tất cả các pids đang sử dụng)
- `void release_pid (int pid)` – Trả lại một pid

3.21 Phỏng đoán Collatz² 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:

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

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, trong lúc tiến trình cha chờ tiến trình con hoàn thành thông qua lời gọi `wait()`.

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

² https://vi.wikipedia.org/wiki/Ph%E1%BB%8Fng_%C4%91%C3%A1n_Collatz