**MÔN HỌC: HỆ ĐIỀU HÀNH**

**CÂU HỎI VÀ BÀI TẬP CHƯƠNG 3**

BIÊN SOẠN: PHAN ĐÌNH DUY

1. Một tiến trình chứa những thành phần gì?

Một chương trình (program) sẽ trở thành tiến trình (process) chỉ khi nó được nạp lên bộ nhớ (memory) của máy tính (hay nói cách khác – được thực thi).

Một tiến trình bao gồm :

Trong bộ nhớ (memory) :

* **Text Section (Program code) :** Chứa những đoạn mã chương trình đã được biên dịch bởi compiler.
* **Data Section (khu vực dữ liệu) :** Chứa các biến toàn cục (global variables) và các biến tĩnh (static variables) được khởi tạo trước khi hàm main được gọi.
* **Heap :** Dùng để lưu trữ các bộ nhớ được cấp phát động (như việc gọi new, delete, malloc, calloc, free,…).
* **Stack :** Dùng để lưu trữ các biến cục bộ (local variables). VD như lúc bạn khai báo trong C : int i = 0. Thì biến i sẽ được lưu trong bộ nhớ Stack của Process.

Ngoài ra, trong process còn có :

* Program Counter (PC).
* Process status word (PSW).
* Stack pointer (SP).
* Memory management registers.

1. Tiến trình có những trạng thái nào? Cách tiến trình chuyển trạng thái?

Tiến trình có các trạng thái sau :

* **new :** Tiến trình vừa được tạo.
* **ready :**Tiến trình đã có đủ tài nguyên, đang chờ được cấp CPU để chạy.
* **running :**Các lệnh của tiến trình đang được thực thi.
* **waiting (hay blocked) :** Tiến trình đợi I/O hoàn tất.
* **terminated :** Tiến trình đã kết thúc (đã thực thi xong).

Cách tiến trình chuyển trạng thái :

Đầu tiên, khi vừa khởi tạo, tiến trình sẽ ở trạng thái là **new.**

Thông qua bộ định thời dài hạn **Long-term scheduling** (hay còn gọi là bộ định thời công việc – **Job Scheduler)**, tiến trình của chúng ta từ **new** sẽ được sắp xếp vị trí để “chui” vào trong hàng đợi **ready.**

Ở một số hệ điều hành có thêm bộ định thời **Medium-term Scheduler.** Thông qua bộ định thời này, các tiến trình sẽ được swap-out (chuyển tiến trình từ bộ nhớ chính sang bộ nhớ phụ) và swap-in (chuyển tiến trình từ bộ nhớ phụ vào bộ nhớ). VD : Paging trong HĐH Windows, hoặc Swap trong Linux.

Thông qua bộ định thời **Short-term Scheduling** (hay còn được gọi là **Dispatcher**), từ trạng thái **ready** tiến trình sẽ được sắp xếp để chuyển qua trạng thái **running**(là trạng thái chạy – hay trạng thái sử dụng CPU – của tiến trình).

Trong khi đang ở trạng thái **running**, có 3 trạng thái tiếp theo mà tiến trình có thể đạt được tiếp theo :

* **waiting :** Khi tiến trình đang chờ I/O (VD : Khi gọi hàm print(), scanf() trong C).
* **ready :**Khi tiến trình bị interrupt (bị ngắt, không cho chạy nữa) bởi **Short-term Scheduler.** Các lý do ngắt có thể là : Ngắt thời gian (Clock Interrupt), Ngắt ngoại vi (I/O Interrupt), Lời gọi hệ thống (Operating System Call), Signal.
* **terminated :** Khi ứng dụng thực thi xong : Khi gặp lệnh exit, khi thực thi lệnh cuối.

Trong khi ở trạng thái **waiting,** tiến trình sẽ chuyển sang trạng thái **ready**(vào hàng đợi **ready**) sau khi đã thực thi xong I/O.

(Nói thêm : Có thể có **>= 1** tiến trình ở trạng thái ready và waiting, tuy nhiên chỉ có **duy nhất 1** tiến trình ở trạng thái runningtại một **thời điểm nhất định** mà ta đang xét).

1. Tại sao phải cộng tác giữa các tiến trình?

Trong tiến trình thực thi, các tiến trình có thể cộng tác (cooperate) để hoàn thành các công việc.

Các tiến trình cộng tác với nhau để :

* Chia sẻ dữ liệu (information sharing).
* Tăng tốc độ tính toán (computational speedup).
  + Các mạng lưới máy tính sẽ hợp với nhau để tạo thành các cluster.
  + Nếu hệ thống có nhiều CPU, chia công việc tính toán thành nhiều công việc tính toán nhỏ chạy song song.
* Thực hiện một công việc chung.
  + Xây dựng một phần mềm phức tạp bằng cách chia thành các module/process hợp tác nhau.

1. PCB là gì? Dùng để làm gì?

Mỗi tiến trình trong hệ thống đều được cấp phát một **Process Control Block** (PCB). Là một trong các cấu trúc dữ liệu quan trọng nhất của hệ điều hành.

PCB chứa các thông tin liên quan đến process như :

* Trạng thái tiến trình (Process State) : new, ready, running,…
* Bộ đếm chương trình (Program Counter) : Chỉ đến địa chỉ của lệnh tiếp theo sẽ được thực thi cho tiến trình này.
* Các thanh ghi CPU (CPU Registers) : Phụ thuộc vào kiến trúc máy tính. Có thể kể đến vài loại như accumulators, index registers, stack pointers, general-purpose registers, condition-code information.
* Thông tin lập thời biểu CPU (CPU Scheduling Information) : Độ ưu tiên, con trỏ đến các hàng đợi, và các tham số của việc lập thời biểu.
* Thông tin quản lý bộ nhớ (Memory-Management Information) : Chứa page tables, segment tables, memory limits (giới hạn bộ nhớ).
* Thông tin trạng thái I/O (I/O status information) : Chứa danh sách các thiết bị I/O đã được cấp phát cho tiến trình, danh sách các file tiến trình đang mở,…
* Các thông tin quan trọng khác như : Lượng CPU, thời gian sử dụng,PID,…

Vậy, các PCB đơn giản là các kho dữ liệu, trong đó chứa các dữ liệu khác nhau của nhiều process khác nhau.

1. Tiểu trình là gì?

Tiểu trình : Là một đơn vị cơ bản sử dụng CPU, gồm :

* Thread ID.
* PC (Program Counter).
* Registers.
* Stack.
* **Chia sẻ chung code, data, resources (file).**

1. Trình tự thực thi của tiến trình cha và tiến trình con?

Trình tự thực thi của tiến trình cha và tiến trình con ?

Có 2 khả năng có thể xảy ra :

* Tiến trình cha tiếp tục thực thi song song với các tiến trình con của nó.
* Tiến trình cha đợi đến khi một vài hoặc tất cả các tiến trình con của nó đã hoàn tất.

Trong khi giải bài tập thì chúng ta sẽ giải theo trường hợp 2 (Tiến trình cha đợi đến khi tiến trình con vừa gọi kết thúc mới thực thi tiếp).

1. (Bài tập mẫu) Cho đoạn chương trình sau:

|  |
| --- |
| int main (int argc, char\*\* argv)  {  int i = 2;  while (i < =5)  {  i++;  if (i % 2 == 0)  {  printf (“Hello”);  printf (“Hi”);  }  else  {  printf (“Bye”);  }  }  exit (0);  } |

Hỏi trong quá trình thực thi thì tiến trình khi chạy từ chương trình trên đã trải qua những trạng thái nào? Vẽ sơ đồ chuyển trạng thái trong quá trình thực thi?

Trả lời:

Trong quá trình thực thi thì tiến trình khi chạy từ chương trình trên đã trải qua những trạng thái như sau: new – ready – running – waiting – ready – running – waiting – ready – running – waiting – ready – running – waiting – ready – running – terminated

1. Cho đoạn chương trình sau:

|  |
| --- |
| /\* test.c \*/  int main(int argc, char\*\* argv)  {  int a;  for (int i = 1; i < 5; i++)  {  if (i % 2 == 0)  printf(“Hello world\n");  else a = 5\*9;  }  exit(0);  } |

Hỏi trong quá trình thực thi thì tiến trình khi chạy từ chương trình trên đã trải qua những trạng thái nào? Vẽ sơ đồ chuyển trạng thái trong quá trình thực thi?

New – ready – running – waiting – ready – running – waiting – ready – running – terminated

1. Cho đoạn chương trình sau:

|  |
| --- |
| int main (int argc, char\*\* argv)  {  int i = 2;  while (i < =5)  {  i++;  if (i % 2 == 0)  {  printf (“Hello”);  printf (“Hi”);  }  else  {  printf (“Bye”);  }  }  exit (0);  } |

Hỏi trong quá trình thực thi thì tiến trình khi chạy từ chương trình trên đã trải qua những trạng thái nào? Vẽ sơ đồ chuyển trạng thái trong quá trình thực thi?

new - ready - running - waiting - ready - running - waiting - ready - running - waiting - ready - running - waiting - ready - running - waiting - ready - running - waiting - ready - running - terminated.

1. Cho đoạn chương trình sau:

|  |
| --- |
| int main (int argc, char\*\* argv)  {  int a, b, i;  for (i = 16, i >=6; i --)  {  if (i % 3 == 0)  {  printf (“Số %d chia hết cho 3”, i);  }  else  {  a = b + i;  }  }  exit (0);  } |

Hỏi trong quá trình thực thi thì tiến trình khi chạy từ chương trình trên đã trải qua những trạng thái nào? Vẽ sơ đồ chuyển trạng thái trong quá trình thực thi?

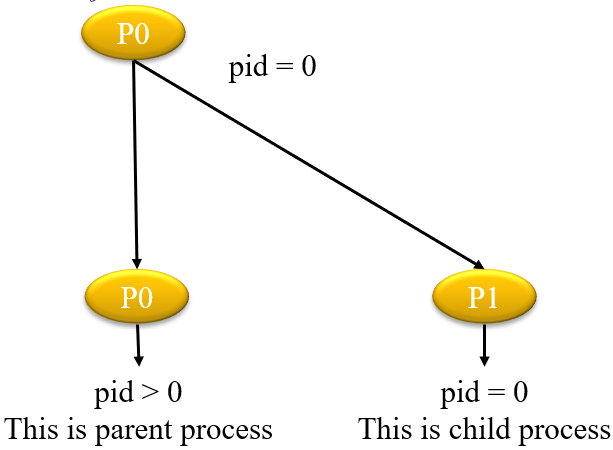
new - ready - running - waiting - ready - running - waiting - ready - running - waiting - ready - running - waiting - terminated.

1. (Bài tập mẫu) Cho đoạn code sau, hỏi khi chạy, bao nhiêu process được sinh ra và chương trình sẽ in ra những gì? Vẽ cây tiến trình khi thực thi đoạn chương trình sau

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  int main (int argc, char \*argv[]){  int pid;  /\* create a new process \*/  pid = fork();  if (pid > 0){  printf(“This is parent process”);  wait(NULL);  exit(0);}  else if (pid == 0) {  printf(“This is child process”);  execlp(“/bin/ls”, “ls”, NULL);  exit(0);}  else { // pid < 0  printf(“Fork error\n”);  exit(-1);  }  } |

Trả lời:

Khi chạy đoạn chương trình trên, khi chạy hết sẽ có 2 process được sinh ra bao gồm 1 tiến trình cha và 1 tiền trình con. Theo chương trình trên thì tiến trình cha sẽ in ra dòng chữ “This is parent process”; và tiến trình con sẽ in ra dòng chữ “This is child process”. Cây tiến trình khi thực thi đoạn chương trình trên như sau:



1. Cho đoạn code sau, hỏi khi chạy, bao nhiêu process (kể cả cha) được sinh ra? Vẽ cây tiến trình khi thực thi đoạn chương trình sau

|  |
| --- |
| int main()  {  fork();  fork();  fork();  fork();  return 0;  } |

Khi chạy đoạn chương trình trên, khi chạy hết sẽ có 16 process được sinh ra bao gồm 1 tiến trình cha và 15 tiến trình con.

1. Cho đoạn code sau, hỏi khi chạy thì tiến trình được tạo ra từ chương trình trên sẽ in ra màn hình những gì? Vẽ cây tiến trình và những từ được in ra khi thực thi đoạn chương trình sau?

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  int main()  {  int i;  for (i = 0; i < 4; i++)  {  fork();  printf("hello\n");  }  return 0;  } |

1. Cho đoạn code sau, hỏi khi chạy thì tiến trình được tạo ra từ chương trình trên sẽ in ra màn hình những gì? Vẽ cây tiến trình và những từ được in ra khi thực thi đoạn chương trình sau?

|  |
| --- |
| int main (int argc, char \*\*argv)  {  int pid;  printf(“Tiến trình cha \n”);  pid = fork();  if (pid > 0)  {  fork();  printf(“Tiến trình cha \n”);  }  else  {  printf(“Tiến trình con \n”);  if(fork() > 0 )  printf("Tiến trình cha \n");  else  printf("Tiến trình con \n");  }  } |

1. Cho đoạn code chương trình sau:

|  |
| --- |
| if (fork() == 0)  {     a = a + 5;     printf("%d,%d\n", a, &a);  }  else  {      a = a –5;      printf("%d, %d\n", a, &a);  } |

Giả sử u, v là các giá trị được in ra bởi process cha, và x, y là các giá trị được in ra bởi process con. Tìm mối quan hệ giữa u, v và x, y?