BÀI TẬP CHƯƠNG 4 – TIỂU TRÌNH VÀ ĐỒNG THỜI

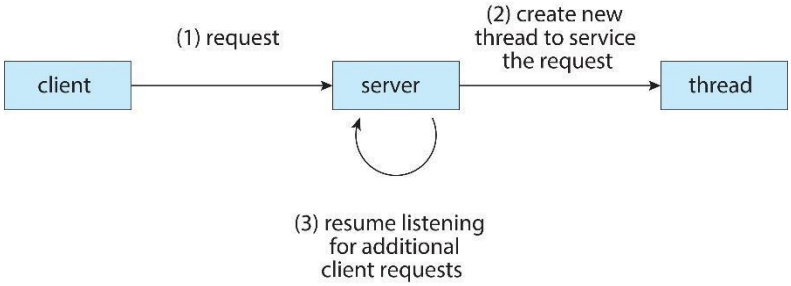
4.1 Cho 3 bài toán ví dụ mà giải pháp lập trình đa luồng sẽ chạy hiệu quả hơn giải pháp đơn luồng?

4.2 Sử dụng luật Amdahl, tính hệ số tăng tốc của một ứng dụng có 60% thành phần được song song hóa chạy trên

a. CPU 2 nhân.

b. CPU 4 nhân.

4.3 Máy chủ web đa luồng được mô tả dưới đây thực hiện song song tác vụ hay song song dữ liệu?



4.4 Chỉ ra 2 khác biệt chính giữa tiểu trình mức người dùng và tiểu trình mức nhân? Trong điều kiện nào thì tiểu trình mức người dùng tốt hơn? Và trong điều kiện nào thì tiểu trình mức nhân tốt hơn?

4.5 Mô tả các hành động của nhân để chuyển ngữ cảnh giữa các luồng mức nhân.

4.6 Những tài nguyên nào được sử dụng khi tạo ra một tiểu trình? Khi đó sự khác biệt với tạo ra một tiến trình mới là gì?

4.7 Giả sử rằng hệ điều hành ánh xạ tiểu trình mức người dùng xuống nhân bằng mô hình nhiều-nhiều và quá trình ánh xạ được hoàn tất bởi các LWP. Hơn nữa, hệ thống cho phép các nhà phát triển tạo ra những tiểu trình thời gian thực để sử dụng trong các hệ thống thời gian thực. Có cần phải trói buộc một tiểu trình thời gian thực vào một LWP? Giải thích.

4.8 Cho 2 bài toán ví dụ mà giải pháp lập trình đa luồng không giúp cho việc thực thi hiệu quả hơn giải pháp đơn luồng.

4.9 Trong điều kiện nào thì giải pháp lập trình đa luồng giúp cho việc thực thi hiệu quả hơn giải pháp đơn luồng chạy trên bộ xử lý đơn nhân.

4.10 Trong trạng thái của một chương trình, thành phần nào sau đây của được chia sẻ giữa các tiểu trình trong một tiến trình đa luồng? Giải thích.

a. Giá trị các thanh ghi.

b. Bộ nhớ đống (heap).

c. Giá trị các biến toàn cục.

d. Bộ nhớ ngăn xếp.

4.11 Một giải pháp đa luồng sử dụng nhiều tiểu trình cấp người dùng chạy trên một hệ thống đa vi xử lý có thể đạt được hiệu năng tốt hơn khi chạy trên hệ thống đơn nhân hay không? Giải thích.

4.12 Tại chương 3, chúng ta đã thảo luận về trình duyệt Google Chrome về lợi ích của việc sinh ra tiến trình mới đáp ứng cho từng thẻ (tab) riêng biệt. Giả sử rằng, Chrome được thiết kế lại và thay vì mỗi tiến trình, được thay bằng mỗi tiểu trình đáp ứng cho từng thẻ riêng biệt, thì những lợi ích tương tự có đạt được hay không? Giải thích.

4.13 Có thể hiện thực đồng thời mà không phải là song song được không? Giải thích.

4.14 Sử dụng luật Amdahl, tính hệ số tăng tốc của các ứng dụng sau:

Có 40% thành phần được song song hóa chạy trên CPU (a) 8 nhân và (b) 16 nhân.

Có 67% thành phần được song song hóa chạy trên CPU (a) 2 nhân và (b) 4 nhân.

Có 90% thành phần được song song hóa chạy trên CPU (a) 4 nhân và (b) 8 nhân.

4.17 Cho đoạn mã sau:

```
pidt pid;
pid = fork();
if (pid == 0){ /* child process */
fork();
threadcreate( . . .);
}
fork();
```

a. Có bao nhiêu tiến trình riêng biệt được tạo ra?

b. Có bao nhiêu tiểu trình riêng biệt được tạo ra?

4.18* Như được mô tả trong Phần 4.7.2, Linux không phân biệt giữa các tiến trình và tiểu trình. Thay vào đó, Linux đối xử với cả hai theo cùng một cách, cho phép một tác vụ gần giống với một tiến trình hoặc một tiểu trình tùy thuộc vào tập hợp các cờ được chuyển đến thông qua lời gọi hệ thống clone(). Tuy nhiên, các hệ điều hành khác, chẳng hạn như Windows, xử lý các tiến trình và tiểu trình khác nhau. Thông thường, như vậy các hệ thống sử dụng ký hiệu trong đó cấu trúc dữ liệu cho một tiến trình chứa con trỏ đến các tiểu trình riêng biệt thuộc tiến trình đó. Tương phản hai cách tiếp cận để mô hình hóa các tiến trình và tiểu trình trong nhân.

4.19 Chương trình sau đây sử dụng Pthreads API. Cái gì sẽ được in ra ở dòng C và P?

```
#include <pthread.h>
#include <stdio.h>
int value = 0;
void *runner(void *param); /* the thread */
int main(int argc, char *argv[])
{
pid t pid;
pthread t tid;
pthread attr t attr;
pid = fork();
if (pid == 0) { /* child process */
pthread attr init(&attr);
pthread create(&tid,&attr,runner,NULL);
pthread join(tid,NULL);
printf("CHILD: value = %d",value); /* LINE C */
}
else if (pid > 0) { /* parent process */
wait(NULL);
printf("PARENT: value = %d",value); /* LINE P */
}
}
void *runner(void *param) {
value = 5;
pthread exit(0);
}
```

4.20 Hãy xem xét một hệ thống có vi xử lý đa nhân và một chương trình đa luồng được viết sử dụng mô hình ánh xạ nhiều-nhiều. Giả định rằng số lượng tiểu trình mức người dùng trong chương trình lớn hơn số nhân xử lý trong hệ thống. Thảo luận về ý nghĩa hiệu suất của các kịch bản sau đây.

a. Số lượng tiểu trình mức nhân được phân bổ cho chương trình ít hơn số lượng nhân xử lý.

b. Số lượng tiểu trình mức nhân được phân bổ cho chương trình bằng với số lượng nhân xử lý.

c. Số lượng tiểu trình mức nhân được phân bổ cho chương trình lớn hơn số lượng nhân xử lý nhưng nhỏ hơn số lương tiểu trình mức người dùng.

4.21* Pthreads cung cấp API để quản lý hủy bỏ tiểu trình. Các Hàm pthread setcancelstate() được sử dụng để đặt trạng thái hủy. Nguyên mẫu của nó xuất hiện như sau:

pthread setcancelstate (int state, int * oldstate)

Hai giá trị có thể có cho trạng thái là PTHREAD CANCEL ENABLE và PTHREAD CANCEL DISABLE.

Sử dụng đoạn mã dưới đây, cung cấp các ví dụ về hai thao tác phù hợp để thực hiện giữa các lời gọi để vô hiệu hóa hủy bỏ tiểu trình hoặc cho phép hủy bỏ tiểu trình.

```
int oldstate;
pthread setcancelstate(PTHREAD CANCEL DISABLE, &oldstate);
/* What operations would be performed here? */
pthread setcancelstate(PTHREAD CANCEL ENABLE, &oldstate);
```

4.22 Viết chương trình đa luồng tính toán các giá trị thống kê khác nhau từ một danh sách các số được truyền vào thông qua đối số của dòng lệnh. Chương trình sau đó sẽ tạo ba tiểu trình tính toán riêng biệt. Một tiểu trình sẽ xác định trung bình cộng của các số, tiểu trình thứ hai sẽ xác định giá trị lớn nhất và tiểu trình thứ ba sẽ xác định giá trị nhỏ nhất. Ví dụ:

```
>./bai22.out 90 81 78 95 79 72 85
Gia tri trung binh: 82
Gia tri lon nhat: 95
Gia tri nho nhat: 72
```

Các biến số đại diện cho các giá trị trung bình, nhỏ nhất và lớn nhất sẽ được lưu trữ trên toàn cục. Các tiểu trình sẽ tính toán các giá trị này và tiến trình cha sẽ xuất ra các giá trị kết quả khi tiểu trình kết thúc.

Mở rộng: có thể tạo thêm các tiểu trình để tính giá trị trung vị; độ lệch chuẩn; sắp xếp dãy số, ...

4.23 Viết chương trình đa luồng để xuất ra số nguyên tố. Người dùng chạy chương trình và nhập vào một số nguyên thông qua đối số tại dòng lệnh. Chương trình sau đó sẽ tạo ra một tiến trình riêng biệt xuất ra tất cả các số nguyên tố nhỏ hơn hoặc bằng số được nhập bởi người dùng.

4.24 Một cách giá trị π khá thú vị là sử dụng kỹ thuật Monte Carlo, liên quan đến ngẫu nhiên. Kỹ thuật này hoạt động như sau:

- Giả sử bạn có một vòng tròn bán kính là 1 nội tiếp trong một hình vuông cạnh là 2, như thể hiện trong hình sau:
- Đầu tiên, tạo một chuỗi các điểm ngẫu nhiên dưới dạng tọa độ (x, y) đơn giản. Những điểm này phải nằm trong tọa độ Descartes bị ràng buộc hình vuông. Trong tổng số điểm ngẫu nhiên được tạo, một số sẽ xảy ra trong vòng tròn.
- Tiếp theo, ước tính π bằng cách thực hiện phép tính sau: π = 4 × (số điểm trong vòng tròn) / (tổng số điểm)

Hãy viết một phiên bản đa luồng của thuật toán này để tạo ra một tiểu trình riêng biệt sinh ra một số lượng điểm ngẫu nhiên; sau đó tính số lượng điểm nằm trong hình và lưu trữ kết quả đó trong một biến toàn cục. Khi tiểu trình này kết thúc, tiến trình cha sẽ tính toán và xuất giá trị ước tính của π. Hãy đánh giá độ chính xác của số π với số lượng điểm ngẫu nhiên được tạo ra. Theo nguyên tắc, số lượng điểm càng lớn, giá trị tính càng tiến gần π.

*Mã nguồn được cung cấp, xin tìm trong LAB.

4.25* Thực hiện lại bài 4.24, nhưng thay vì sử dụng tiểu trình sinh các điểm ngẫu nhiên, hãy dùng OpenMP[1] để song song hóa quá trình sinh các điểm đó. Cẩn thận tránh đặt đoạn tính giá trị π vào vùng song song, vì π chỉ cần tính một lần.

4.27 Chuỗi Fibonacci là chuỗi các số 0, 1, 1, 2, 3, 5, 8, ... có thể được thể hiện bằng công thức như sau:

$$fb0 = 0$$

[1] https://www.openmp.org/

$fb1 = 1$

$fbn = fbn\text{-}1 + fbn\text{-}2$

Viết chương trình đa luồng tạo ra chuỗi Fibonacci. Thông qua đối số dòng lệnh, người dùng sẽ nhập số lượng số Fibonacci mà cần sinh ra. Chương trình sau đó sẽ tạo một tiểu trình sinh số Fibonacci rồi lưu trữ chuỗi này vào nơi có thể chia sẻ được giữa các tiểu trình (mảng là lựa chọn tốt). Khi hoàn tất thực thi, tiến trình cha sẽ xuất ra dãy số đã sinh bởi tiểu trình con. Bởi vì tiến trình cha không thể bắt đầu xuất chuỗi Fibonacci trước khi tiểu trình con hoàn tất, tiến trình cha sẽ phải chờ. Sử dụng các kỹ thuật được mô tả trong Mục 4.4 để đáp ứng yêu cầu này.

4.28 Sửa đổi vấn đề lập trình Bài tập 3.20 từ Chương 3 (yêu cầu thiết kế quản lý pid). Sửa đổi này sẽ bao gồm viết một chương trình đa luồng kiểm tra giải pháp của bạn cho Bài tập 3.20. Bạn sẽ tạo ra một số tiểu trình, ví dụ 100, và mỗi tiểu trình sẽ yêu cầu một pid, tạm dừng trong một khoảng thời gian ngẫu nhiên, và sau đó trả lại pid. Trên các hệ thống UNIX và Linux, việc tạm dừng được thực hiện thông qua lời gọi sleep(), được truyền một giá trị nguyên biểu thị cho số giây để tạm dừng. Vấn đề này sẽ được đề cập lại trong Chương 7.

4.29 Bài tập 3.25 trong Chương 3 liên quan đến việc thiết kế một máy chủ echo bằng cách sử dụng Java threading API.. Máy chủ này là đơn luồng, có nghĩa là máy chủ không thể đáp ứng với máy khách echo đồng thời cho đến máy khách hiện tại thoát. Sửa đổi giải pháp cho Bài tập 3.25 để máy chủ echo phụ vụ mỗi máy khách trong một yêu cầu riêng biệt.