|  |
| --- |
| Họ và tên: Đoàn Phương Nam  Mã số sinh viên: 22520908  Lớp: IT007.O11.2 |

HỆ ĐIỀU HÀNH  
BÁO CÁO LAB 5

**CHECKLIST**

**5.5. BÀI TẬP THỰC HÀNH**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **BT 1** | **BT 2** | **BT 3** | **BT 4** |
| **Trình bày cách làm** |  |  |  |  |
| **Chụp hình minh chứng** |  |  |  |  |
| **Giải thích kết quả** |  |  |  |  |

**5.6. BÀI TẬP ÔN TẬP**

|  |  |
| --- | --- |
|  | **BT 1** |
| **Trình bày cách làm** |  |
| **Chụp hình minh chứng** |  |
| **Giải thích kết quả** |  |

**Tự chấm điểm:** 10/10

*\*Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp:* ***<Tên nhóm>\_LAB5.pdf***

**5.5. BÀI TẬP THỰC HÀNH**

# Hiện thực hóa mô hình trong ví dụ 5.3.1.2, tuy nhiên thay bằng điều kiện sau: sells <= products <= sells + [4 số cuối của MSSV]

A screen shot of a computer program

Description automatically generated

**Trình bày cách làm**

Khởi tạo semaphore:

sem\_init(&sem1, 0, 0): Khởi tạo semaphore sem1 với giá trị ban đầu là 0.

sem\_init(&sem2, 0, sells + 908): Khởi tạo semaphore sem2 với giá trị ban đầu là sells cộng với 908.

Tạo luồng:

pthread\_create(&pA, NULL, &ProcessA, NULL): Tạo luồng A để thực thi hàm

ProcessA.

pthread\_create(&pB, NULL, &ProcessB, NULL): Tạo luồng B để thực thi hàm ProcessB.

Vòng lặp của mỗi luồng:

Process A:

sem\_wait(&sem1): Đợi semaphore sem1 được unlock.

sells++: Tăng giá trị biến sells lên 1.

printf("Sells = %d\n", sells): In ra giá trị hiện tại của sells.

sem\_post(&sem2): Unlock semaphore sem2.

Process B:

sem\_wait(&sem2): Đợi semaphore sem2 được unlock.

products++: Tăng giá trị biến products lên 1.

printf("Products = %d\n", products): In ra giá trị hiện tại của products.

sem\_post(&sem1): Unlock semaphore sem1.

Đồng bộ hóa:

Semaphore sem1 kiểm soát việc truy cập biến sells. Chỉ một luồng có thể tăng sells tại một thời điểm.

Semaphore sem2 kiểm soát việc truy cập biến products. Chỉ một luồng có thể tăng products tại một thời điểm.

**Giải thích**

Các luồng A và B sẽ thay phiên nhau in ra giá trị tăng dần của sells và products.

Luồng B sẽ không bắt đầu tăng products cho đến khi bằng sells + 0908.

Đảm bảo rằng các luồng truy cập và sửa đổi biến một cách an toàn, tránh các race condition và dữ liệu không nhất quán.

A screen shot of a computer

Description automatically generated

# Cho một mảng a được khai báo như một mảng số nguyên có thể chứa n phần tử, a được khai báo như một biến toàn cục. Viết chương trình bao gồm 2 thread chạy song song:

* Một thread làm nhiệm vụ sinh ra một số nguyên ngẫu nhiên sau đó bỏ vào a. Sau đó đếm và xuất ra số phần tử của a có được ngay sau khi thêm vào.
* Thread còn lại lấy ra một phần tử trong a (phần tử bất kỳ, phụ thuộc vào người lập trình). Sau đó đếm và xuất ra số phần tử của a có được ngay sau khi lấy ra, nếu không có phần tử nào trong a thì xuất ra màn hình “Nothing in array a”.

Chạy thử và tìm ra lỗi khi chạy chương trình trên khi chưa được đồng bộ. Thực hiện đồng bộ hóa với semaphore.

A screen shot of a computer program

Description automatically generatedA computer screen shot of text

Description automatically generated

A screen shot of a computer program

Description automatically generated

Sau đồng bộ

A screen shot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

**Trình bày cách làm:**

Khởi tạo biến:

a: con trỏ đến mảng lưu trữ các số nguyên

n: kích thước của mảng

i: biến đếm số phần tử hiện có trong mảng

sem1, sem2, busy: ba semaphore dùng để đồng bộ hóa các luồng

Hàm Arrange(a, x):

Hàm này thực hiện việc sắp xếp lại các phần tử trong mảng khi một phần tử bị xóa.

Nếu x bằng i, tức là phần tử cuối cùng bị xóa, thì i giảm 1.

Ngược lại, tất cả các phần tử từ vị trí x trở đi sẽ được dịch lên 1 vị trí.

Luồng ProcessA:

Luồng này liên tục thực hiện các bước sau:

Đợi semaphore sem2 để đảm bảo có chỗ trống trong mảng.

Tăng semaphore busy để ngăn luồng ProcessB truy cập mảng.

Thêm một số ngẫu nhiên vào mảng.

Tăng biến đếm i.

In ra số phần tử hiện có trong mảng.

Đợi semaphore sem1 để báo hiệu luồng ProcessB có thể thực hiện.

Giảm semaphore busy để cho phép luồng ProcessB tiếp tục.

Luồng ProcessB:

Luồng này liên tục thực hiện các bước sau:

Đợi semaphore sem1 để đảm bảo có phần tử trong mảng.

Tăng semaphore busy để ngăn luồng ProcessA truy cập mảng.

Kiểm tra xem mảng có trống hay không.

Nếu mảng trống, in thông báo "Nothing in the array".

Ngược lại, chọn ngẫu nhiên một phần tử và xóa nó khỏi mảng.

Gọi hàm Arrange(a, r) để sắp xếp lại các phần tử.

In ra số phần tử còn lại trong mảng.

Đợi semaphore sem2 để báo hiệu luồng ProcessA có thể tiếp tục.

Giảm semaphore busy để cho phép luồng ProcessA tiếp tục.

**Giải thích kết quả:**

Hai luồng ProcessA và ProcessB hoạt động đồng thời và truy cập vào mảng a thông qua các semaphore.

Luồng ProcessA liên tục thêm các phần tử vào mảng, làm tăng số phần tử i.

Luồng ProcessB liên tục xóa các phần tử khỏi mảng, làm giảm số phần tử i.

Semaphore sem1 đảm bảo rằng luồng ProcessB chỉ có thể xóa phần tử sau khi luồng ProcessA đã thêm phần tử mới.

Semaphore sem2 đảm bảo rằng luồng ProcessA chỉ có thể thêm phần tử mới sau khi luồng ProcessB đã xóa phần tử cũ.

Semaphore busy đảm bảo rằng chỉ có một luồng có thể truy cập vào mảng a tại một thời điểm, để tránh lỗi race condition.

# Cho 2 process A và B chạy song song như sau:

|  |  |
| --- | --- |
| int x = 0; | |
| **PROCESS A** | **PROCESS B** |
| processA()  {  while(1){  x = x + 1;  if (x == 20)  x = 0;  print(x);  }  } | processB()  {  while(1){  x = x + 1;  if (x == 20)  x = 0;  print(x);  }  } |

Hiện thực mô hình trên C trong hệ điều hành Linux và nhận xét kết quả.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

Trình bày cách làm:

Đoạn code trên tạo ra hai luồng, Process A và Process B, cả hai đều truy cập và sửa đổi biến chia sẻ x cùng một lúc. Cả hai luồng đều tăng x và đặt lại nó về 0 khi nó đạt đến 20. Tuy nhiên, do thiếu đồng bộ nên dẫn đến hành vi không xác định do các điều kiện thi đấu.

Dưới đây là phân tích các lỗi tiềm ẩn:

Race condition : Cả hai luồng đều có thể truy cập và sửa đổi x cùng một lúc. Điều này có thể dẫn đến tình huống một luồng đọc giá trị của x, tăng nó, nhưng trước khi có thể ghi giá trị mới trở lại bộ nhớ, luồng kia cũng đọc giá trị cũ, tăng nó và ghi giá trị của riêng mình. Điều này dẫn đến giá trị cuối cùng không chính xác cho x.

Cập nhật bị mất: Nếu một luồng tăng x và ghi nó trở lại bộ nhớ, nhưng sau đó luồng kia chiếm ưu thế và đọc giá trị cũ trước khi cập nhật, cập nhật của luồng đầu tiên bị mất, dẫn đến sự không nhất quán trong giá trị cuối cùng của x.

Đầu ra không chính xác: Do các điều kiện thi đấu và cập nhật bị mất, đầu ra của chương trình sẽ không thể đoán trước và có thể không chính xác. Thay vì chuỗi số dự kiến từ 1 đến 20, đầu ra lộn xộn và không nhất quán.

# Đồng bộ với mutex để sửa lỗi bất hợp lý trong kết quả của mô hình Bài 3.

A screen shot of a computer program

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer

Description automatically generated

**Trình bày cách làm:**

Tạo ra hai luồng, Process A và Process B, thực hiện các hành động sau:

Khởi tạo mutex: Trước khi tạo luồng, chương trình khởi tạo một mutex có tên mutex bằng pthread\_mutex\_init(&mutex, NULL). Mutex này sẽ được sử dụng để đảm bảo truy cập độc quyền vào biến x.

Tạo luồng: Chương trình tạo hai luồng sử dụng pthread\_create:

Luồng A chạy hàm ProcessA, luồng B chạy hàm ProcessB.

Luồng thực thi:

Cả hai luồng đều chạy các vòng lặp vô hạn, lặp đi lặp lại các bước sau:

Khóa mutex: Luồng gọi pthread\_mutex\_lock(&mutex) để lấy khóa mutex, ngăn chặn luồng khác truy cập biến x.

Truy cập và sửa đổi x: Luồng tăng giá trị của x lên 1. Nếu x đạt 20, nó sẽ được đặt lại thành 0.

In giá trị x: Luồng in giá trị hiện tại của x và tên luồng ra màn hình.

Mở khóa mutex: Luồng gọi pthread\_mutex\_unlock(&mutex) để giải phóng khóa, cho phép luồng khác truy cập x.

Đồng bộ hóa: Mutex đảm bảo rằng chỉ một luồng có thể truy cập và sửa đổi x tại một thời điểm. Điều này ngăn chặn các điều kiện tranh chấp và đảm bảo rằng giá trị của x được cập nhật một cách nhất quán.

Thứ tự thực hiện: Các luồng có thể chạy luân phiên hoặc xen kẽ, tùy thuộc vào cách lập lịch trình của hệ điều hành. Tuy nhiên, mutex đảm bảo rằng dù thứ tự thực thi như thế nào, các giá trị của x vẫn được cập nhật chính xác và nhất quán.

**Giải thích:**

Luồng A và luồng B sẽ thay phiên nhau in các giá trị tăng dần từ 1 đến 20.

Mỗi luồng sẽ in giá trị x sau khi nó đã tăng giá trị và giữ khóa mutex.

Khi x đạt đến 20, nó sẽ được đặt lại thành 0 và quá trình sẽ tiếp tục.

Chuỗi số sẽ tiếp tục lặp lại từ 1 đến 20 do vòng lặp vô hạn trong các luồng.