|  |
| --- |
| Họ và tên: Nguyễn Trần Bảo Anh  Mã số sinh viên: 22520066  Lớp: IT007.O21.CNVN.1 |

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:** 9

*\*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]

**Trình bày cách làm**

**A screen shot of a computer program

Description automatically generated**

**Chụp hình minh chứng**

**A screenshot of a computer

Description automatically generated**

Hình 1: Process thực hiện việc cộng Products được chạy trước.

A screenshot of a computer

Description automatically generated

Hình 2: Sau khi đạt giới hạn, Products bị buộc dừng bởi sem\_wait và Sells được chạy.

**Giải thích kết quả**

sem1 đóng vai trò là điều kiện (sells<=products), sem2 đóng vai trò là điều kiện (products <= sells + 66). Khi chương trình được nạp, giả sử ProcessA được nạp trước. Khi đó A sẽ bị khóa lại với bởi biến sem1 (khởi tạo = 0). Vì vậy ProcessB sẽ được chạy. Products và sem1 (bởi hàm sem\_post) sẽ được tăng lên cho đến khi sem2=0 (được giảm bởi hàm sem\_wait). Khi đó ProcessA sẽ được chạy. Sells và sem2 (bởi hàm sem\_post) sẽ tăng cho đến khi sem1=0 (được giảm bởi hàm sem\_wait). Hai tiến trình này sẽ được lặp đi lặp lại trong 1 vòng lặp vô hạn.

# 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. Chưa đồng bộ**

**Trình bày cách làm**

**A screen shot of a computer program

Description automatically generated**

**Chụp hình minh chứng**

A screenshot of a computer

Description automatically generated

Hình 3: Tiến trình sinh ra phần tử và bỏ vào a được chạy trước.

A black screen with white text

Description automatically generated

Hình 4: Lỗi xuất hiện khi chưa đồng bộ.

**b. Đã đồng bộ**

**Trình bày cách làm**

**A screen shot of a computer program

Description automatically generated**

**Chụp hình minh chứng**

**A screenshot of a computer

Description automatically generated**

Hình 5: Số phần tử trong a được thêm vào và lấy ra luôn có sự đồng nhất về dữ liệu.

**A screenshot of a computer program

Description automatically generated**

Hình 6: Tương tự khi 2 bên đổi vị trí.

**Giải thích kết quả**

Hàm arrange đóng vai trò sắp xếp lại hàm khi lấy ngẫu nhiên một phần tử trong a ra. Biến sem1 đóng vai trò điều kiện đảm bảo rằng sẽ không thể lấy phần tử trong A khi không có phần tử nào. Biến sem2 đóng vai trò điều kiện đảm bảo rằng sẽ không thể thêm phần tử quá số lượng phần tử được cho phép. Khi chương trình được nạp, giả sử ProcessB được nạp trước sẽ bị khóa bởi biến sem1 (khởi tạo = 0). Khi đó ProcessA được chạy, tiếp tục thêm vào phần tử có giá trị ngẫu nhiên vào a và tăng sem1 (bởi hàm sem\_post) cho đến khi sem2=0 (được giảm bởi hàm sem\_wait). Khi đó ProcessB được chạy, lấy một phần tử ngẫu nhiên trong a ra và tăng sem2 sem1 (bởi hàm sem\_post) cho đến khi sem1 = 0 (được giảm bởi hàm sem\_wait). Hai tiến trình này sẽ được lặp đi lặp lại trong 1 vòng lặp vô hạn.

# 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.

**Trình bày cách làm**

**A screen shot of a computer program

Description automatically generated**

**Chụp hình minh chứng**

**A screenshot of a computer

Description automatically generated**

**Giải thích kết quả**

* Lỗi xuất hiện khi chưa được đồng bộ. Process B đã lấy giá trị của biến x ở thời điểm Process A đang tính toán x=12, do đó sau khi Process A kết thúc quá trình chạy thì Process B lập tức in ra kết quả đã tính trước đó.

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

**Trình bày cách làm**

**A screen shot of a computer program

Description automatically generated**

**Chụp hình minh chứng**

**A screenshot of a computer

Description automatically generated**

Hình 7: Process A kết thúc và lập tức Process B tiếp tục tính toán theo kết quả cuối cùng của Process A

**A screenshot of a computer

Description automatically generated**

Hình 8: Tương tự khi Process B kết thúc.

**Giải thích kết quả**

biến mutex đóng vai trò là chìa khóa đóng mở vùng tranh chấp, tránh tình trạng process lấy giá trị của biến trong vùng tranh chấp khi 1 process đang chạy làm dữ liệu trở nên không đồng nhất.

**5.6. BÀI TẬP ÔN TẬP**

# Biến ans được tính từ các biến x1, x2, x3, x4, x5, x6 như sau:

* w = x1 \* x2; (a)
* v = x3 \* x4; (b)
* y = v \* x5; (c)
* z = v \* x6; (d)
* y = w \* y; (e)
* z = w \* z; (f)
* ans = y + z; (g)
* Giả sử các lệnh từ (a) 🡪 (g) nằm trên các thread chạy song song với nhau. Hãy lập trình mô phỏng và đồng bộ trên C trong hệ điều hành Linux theo thứ tự sau:
* (c), (d) chỉ được thực hiện sau khi v được tính
* (e) chỉ được thực hiện sau khi w và y được tính
* (g) chỉ được thực hiện sau khi y và z được tính

**Trình bày cách làm**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**Chụp hình minh chứng**

**A screenshot of a computer

Description automatically generated**

* Process a) và b) được chạy trước.
* Lần lượt các process c) d) e) g) được chạy.

**Giải thích kết quả**

* Các biến sem\_AB, sem\_CD, sem\_EF và sem\_G là các biến semaphore đóng vai trò điều kiện giúp các process khác được chạy. Các biến sem\_CD\_extra và sem\_EF\_extra đóng vai trò điều kiện lần lượt cho các process CD và EF được chạy. Biến busy đóng vai trò là mutex, khóa và mở vùng tranh chấp. Khi chương trình được nạp, Giả sử các Process khác AB được chạy trước:
* **+** Process G sẽ bị khóa bởi biến sem\_EF (khởi tạo=0).
* + Process EF sẽ bị khóa bởi biến sem\_EF\_extra (khởi tạo=0).
* + Process CD sẽ bị khóa bởi biến sem\_CD\_extra (khởi tạo=0).
* + Process AB sẽ được chạy do biến semG được khởi tạo =1 và khóa vùng tranh chấp.
* + Sau khi AB chạy xong, các điều kiện sẽ được kích hoạt để CD có thể chạy (Các điều kiện này không xung đột với điều kiện ở các process còn lại), đồng thời mở vùng tranh chấp.
* + Sau khi CD chạy xong, tiếp tục tương tự với AB khi cho EF chạy tiếp.
* + EF chạy xong kích hoạt điều kiện cho G được chạy.
* + G sau khi chạy xong, kích hoạt điều kiện để cho A được chạy tiếp, tiếp tục chương trình như 1 vòng lặp.