



**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA HỆ THỐNG THÔNG TIN**

**🙢 🕮 🙠**

**-Logo

Description automatically generated**

**THỰC HÀNH HỆ ĐIỀU HÀNH**

**Bài thực hành số 5**

**Họ và tên: Lê Anh Thư**

**MSSV: 20521985**

**Giảng viên: Nguyễn Thanh Nam**

**Câu 1: Hiện thực hóa mô hình trong ví dụ 5.3.1.2, tuy nhiên thay bằng điều kiện sau: sells <= products <= sells + [2 số cuối của MSSV + 10]**

Hiện thực hóa mô hình trong ví dụ 5.3.1.2 và thay bằng điều kiện:

sells <= products <= sells + [2 số cuối của MSSV + 10]

Mã số sinh viên: 20521985 => 2 số cuối của MSSV: 85

Vậy thay điều kiện trên bằng 85 + 10 = 95

* sem\_init(&sem2,0,95);

**Source code:**

#include <semaphore.h>

#include <pthread.h>

#include <stdio.h>

int sells = 0, products = 0;

sem\_t sem1;

sem\_t sem2;

void \*processA(void \*mess) {

while(1) {

        sem\_wait(&sem1);

        sells++;

        printf("SELLS = %d\n",sells);

        sem\_post(&sem2);

}

}

void \*processB(void \*mess) {

while(1) {

sem\_wait(&sem2);

    products++;

    printf("PRODUCTS = %d\n",products);

    sem\_post(&sem1);

}

}

void main() {

    sem\_init(&sem1,0,0);

    sem\_init(&sem2,0,95);

    pthread\_t pA, pB;

    pthread\_create(&pA,NULL,&processA,NULL);

    pthread\_create(&pB,NULL,&processB,NULL);

    while(1){}

}

**Hình ảnh:**

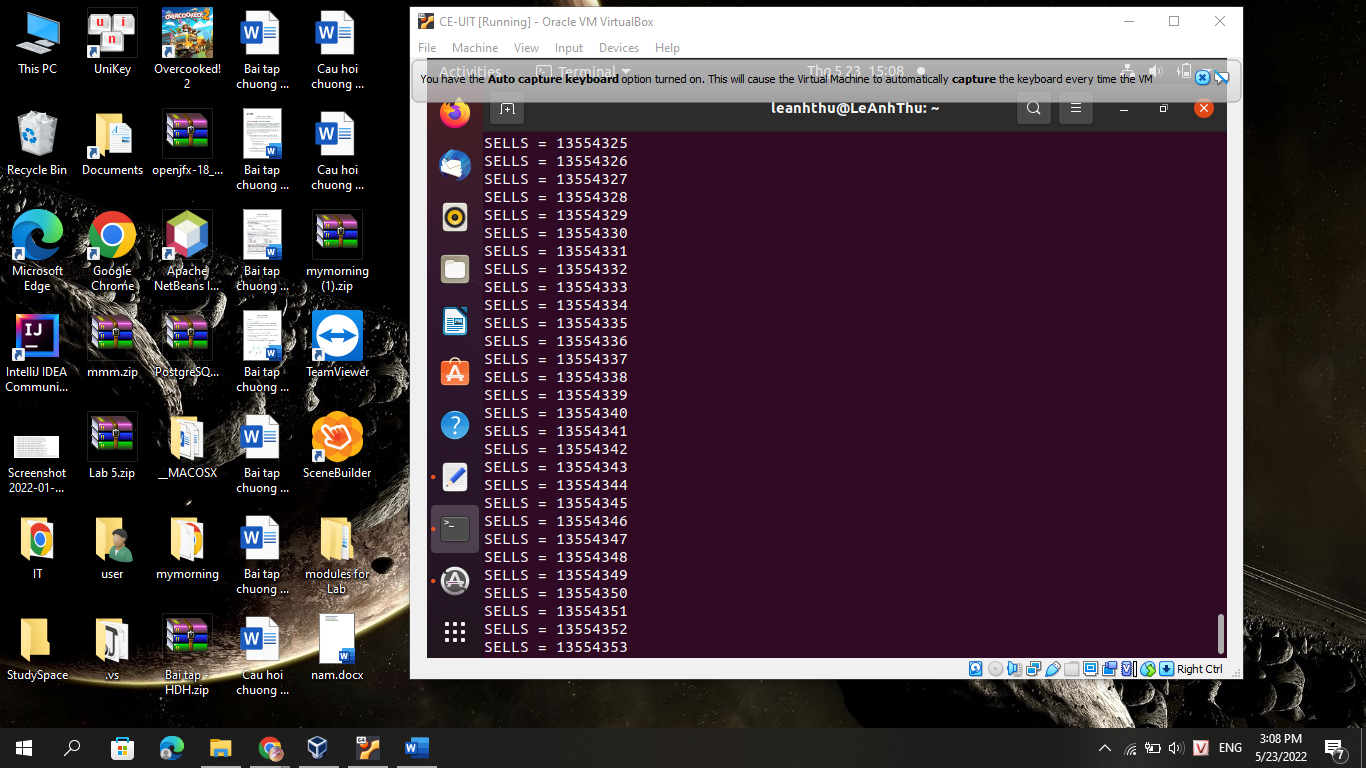
Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

**Kết quả:**



**Câu 2: 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.**

1. Chương trình khi chưa đồng bộ hóa bằng semaphore:

**Source code:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <time.h>

#include <pthread.h>

#include <semaphore.h>

int n;

int i = 0;

int dem = 0;

int a[100000];

void\* processA()

{

    while (1) {

        if (dem < n) {

            a[i++] = rand() % (n - 1);

            dem++;

            printf("\n[PUSH] Length of array a: %d", dem);

        }

        int time\_sleep = rand() % 2 + 1;

        sleep(time\_sleep);

    }

}

void\* processB()

{

    int j;

    while (1) {

        dem--;

        if (dem == 0) {

            printf("\n[POP] Nothing in array a");

        }

        else {

            for (j = 0; j < dem; j++) {

                a[j] = a[j + 1];

            }

            printf("\n[POP] Length of array a: %d", dem);

        }

    int time\_sleep = rand() % 2 + 1;

    sleep(time\_sleep);

    }

}

void main()

{

    printf("\nEnter n: ");

    scanf("%d",&n);

    pthread\_t pA, pB;

    pthread\_create(&pA, NULL, processA, NULL);

    pthread\_create(&pB, NULL, processB, NULL);

    while(1) { }

}

**Hình ảnh:**

Text

Description automatically generated

Text

Description automatically generated

**Kết quả:**

A screenshot of a computer

Description automatically generated with medium confidence

**Nhận xét:**

Chương trình có lỗi vì chưa được đồng bộ bằng semaphore. Chính vì thế một trong hai process sẽ chạy không ngừng, dẫn tới vượt quá kích thước mảng a, kết quả là chương trình sẽ break.

1. Chương trình khi đồng bộ hóa bằng semaphore:

**Source code:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <time.h>

#include <pthread.h>

#include <semaphore.h>

sem\_t sem1, sem2;

int n;

int i = 0;

int dem = 0;

int a[100000];

void\* processA()

{

    while (1) {

        if (dem < n) {

            a[i++] = rand() % (n - 1);

            dem++;

            printf("\n[PUSH] Length of array a: %d", dem);

        }

        int time\_sleep = rand() % 2 + 1;

        sleep(time\_sleep);

        sem\_post(&sem1);

    }

}

void\* processB()

{

    int j;

    while (1) {

        sem\_wait(&sem1);

        dem--;

        if (dem == 0) {

            printf("\n[POP] Nothing in array a");

        }

        else {

            for (j = 0; j < dem; j++) {

                a[j] = a[j + 1];

            }

            printf("\n[POP] Length of array a: %d", dem);

        }

    int time\_sleep = rand() % 2 + 1;

    sleep(time\_sleep);

    }

}

void main()

{

    sem\_init(&sem1, 1, 0);

    sem\_init(&sem2, 0, 0);

    printf("\nEnter n: ");

    scanf("%d",&n);

    pthread\_t pA, pB;

    pthread\_create(&pA, NULL, processA, NULL);

    pthread\_create(&pB, NULL, processB, NULL);

    while(1) { }

}

**Hình ảnh:**

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

**Kết quả:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Nhận xét:**

Sau khi chương trình được đồng bộ với semaphore. Chương trình không bị chạy liên tục. Process A và Process B chạy song song nhau, cho ra kết quả tính đúng như thực tế.

**Câu 3: 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.**

* **Hiện thực mô hình trên C trong hệ điều hành Linux**

**Source code:**

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

int x=0;

void processA(void mess){

while(1){

    x = x+1;

    if(x == 20)

        x=0;

    printf("PA: \nx= %d",x);

}

}

void processB(void mess){

while(1){

    x = x+1;

    if(x == 20)

        x=0;

    printf("PB: \nx= %d",x);

}

}

void main(){

    pthread\_t pA,pB;

    pthread\_create(&pA,NULL,&processA,NULL);

    pthread\_create(&pB,NULL,&processB,NULL);

    while(1){};

}

**Hình ảnh:**

Graphical user interface, text, application

Description automatically generated

**Kết quả:**

A screenshot of a computer

Description automatically generated

Text

Description automatically generated

**Nhận xét:** Chương trình lỗi vì process A không đồng bộ biến x với process B.

- Dữ liệu của biến x không được nhất quán. Process A và Process B chạy song song làm cho biến x có giá trị bị lộn xộn và kết quả là không theo thứ tự tăng dần.

- Lỗi xảy ra khi có xuất hiện chuyển ngữ cảnh của CPU, khi process A đang chạy, biến x sẽ được đọc vào thanh ghi nhưng chưa kịp xuất ra kết quả màn hình thì hết time slice, sau đó process B làm thay đổi dữ liệu ở biến count, và khi process A được tiếp tục thực thi thì dữ liệu khi nãy được load vào thanh ghi đã lỗi thời nên gây ra hiện tượng này.

**Câu 4: Đồng bộ với mutex để sửa lỗi bất hợp lý trong kết quả của mô hình Bài 3.**

**Source code:**

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

sem\_t sem\_1, sem\_2;

int x=0;

pthread\_mutex\_t mutex;

void processA(void mess){

while(1){

pthread\_mutex\_lock(&mutex);

    x = x+1;

    if(x == 20)

        x=0;

    printf("PA: \nx= %d",x);

pthread\_mutex\_unlock(&mutex);

}

}

void processB(void mess){

while(1){

pthread\_mutex\_lock(&mutex);

    x = x+1;

    if(x == 20)

        x=0;

    printf("PB: \nx= %d",x);

pthread\_mutex\_unlock(&mutex);

}

}

void main(){

pthread\_mutex\_init(&mutex, NULL);

    pthread\_t pA,pB;

    pthread\_create(&pA,NULL,&processA,NULL);

    pthread\_create(&pB,NULL,&processB,NULL);

    while(1){};

}

**Hình ảnh:**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Kết quả:**

Graphical user interface, text

Description automatically generated

**Nhận xét:** Biến x đã được đồng bộ giữa hai process. Process A kết thúc và lập tức process B tiếp tục tính toán theo kết quả cuối cùng của process A và tương tự khi process B kết thúc.

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.

**Câu 5: 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**

Cách thực hiện:

**Vì (c), (d) chỉ được thực hiện sau khi v được tính :**

* Process C chỉ được thực thi sau khi process B hoàn thành
* Process D chỉ được thực thi sau khi process B hoàn thành

**Vì (e) chỉ được thực hiện sau khi w và y được tính :**

* Process E chỉ được thực thi sau khi process A và C hoàn thành

**Vì z = w \* z; (f)**

**Mà w = x1 \* x2; (a) và z = v \* x6; (d)**

* Process F chỉ được thực thi sau khi process A và D hoàn thành

**Vì (g) chỉ được thực hiện sau khi y và z được tính**

* Process G chỉ được thực thi sau khi process E và F hoàn thành

- Nếu 1 process cần đợi tới 2 process khác thực thi xong thì ta phải đặt sem\_wait() 2 lần, và mỗi process cần thực thi trước nó đều gọi sem\_post() để unlock cho process đó.

- Nếu 2 process cùng đợi 1 process thì 2 process đó đều đặt sem\_wait(), và process cần chạy trước đặt sem\_post() 2 lần.

**Source code:**

#include <stdio.h>

#include <semaphore.h>

#include <unistd.h>

#include <pthread.h>

#include <stdlib.h>

int x1 = 1, x2 = 2, x3 = 3, x4 = 4, x5 = 5, x6 = 6;

int w, v, y, z, ans;

sem\_t sem1, sem2, sem3, sem4;

void \*ProcessA(void\* mess)

{

    w = x1 \* x2;

    printf("\npA: w = x1 \* x2 = %d + %d = %d\n", x1, x2, w);

    sem\_post(&sem2);

sem\_post(&sem3);

}

void \*ProcessB(void\* mess)

{

    v = x3 \* x4;

    printf("\npB: v = x3 \* x4 = %d + %d = %d\n", x3, x4, w);

    sem\_post(&sem1);

sem\_post(&sem1);

}

void \*ProcessC(void\* mess)

{

    sem\_wait(&sem1);

    y = v \* x5;

    printf("\npC: y = v \* x5 = %d + %d = %d\n", v, x5, y);

    sem\_post(&sem2);

}

void \*ProcessD(void\* mess)

{

sem\_wait(&sem1);

    z = v \* x6;

    printf("\npD: z = v \* x6 = %d + %d = %d\n", v, x6, z);

    sem\_post(&sem3);

}

void \*ProcessE()

{

    sem\_wait(&sem2);

sem\_wait(&sem2);

int temp = y;

    y = w \* y;

    printf("\npE: y = w \* y = %d + %d = %d\n", w, temp, y);

    sem\_post(&sem4);

}

void \*ProcessF(void\* mess)

{

    sem\_wait(&sem3);

sem\_wait(&sem3);

int temp = z;

    z = w \* z;

    printf("\npF: z = w \* z = %d + %d = %d\n", w, temp, z);

    sem\_post(&sem4);

}

void \*ProcessG(void\* mess)

{

    sem\_post(&sem4);

sem\_post(&sem4);

    ans = y + z;

    printf("\npG: ans = y + z = %d + %d = %d\n", y, z, ans);

}

int main()

{

    sem\_init(&sem1, 0, 0);

    sem\_init(&sem2, 0, 0);

    sem\_init(&sem3, 0, 0);

    sem\_init(&sem4, 0, 0);

printf(“x1 = %d\n”,x1);

printf(“x2 = %d\n”,x2);

printf(“x3 = %d\n”,x3);

printf(“x4 = %d\n”,x4);

printf(“x5 = %d\n”,x5);

printf(“x6 = %d\n”,x6);

    pthread\_t pA, pB, pC, pD, pE, pF, pG;

    pthread\_create(&pA, NULL, &ProcessA, NULL);

    pthread\_create(&pB, NULL, &ProcessB, NULL);

    pthread\_create(&pC, NULL, &ProcessC, NULL);

    pthread\_create(&pD, NULL, &ProcessD, NULL);

    pthread\_create(&pE, NULL, &ProcessE, NULL);

    pthread\_create(&pF, NULL, &ProcessF, NULL);

    pthread\_create(&pG, NULL, &ProcessG, NULL);

    while (1)

    {

    }

    return 0;

}

**Hình ảnh:**

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing text, computer, indoor, electronics

Description automatically generated

**Kết quả:**

Text

Description automatically generated