Báo cáo thực hành môn Hệ điều hành - Giảng viên: Trần Hoàng Lộc.

Họ và tên: Dương Thuận Trí Mã số sinh viên: 22521517 Lớp: 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: 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

1. **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]**

   - **Code:**

```c
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>

int sells=0;
int products=0;
sem_t sem1, sem2;

void *processA(void* messenge){
while(1){
    sem_wait(&sem1);
    sells++;
    printf("Sell: %d\n", sells);
    sem_post(&sem2);
}
}

void *processB(void* messenge){
while(1){
    sem_wait(&sem2);
    products++;
    printf("Products: %d\n", products);
    sem_post(&sem1);
}
}

int main(){
sem_init(&sem1, 0,0);
sem_init(&sem2, 0,1517);
pthread_t pA,pB;
pthread_create(&pA,NULL,&processA,NULL);
pthread_create(&pB, NULL, &processB, NULL);
while(1){}
}
```

MSSV: 22521517

- **Output:**

```
PS D:\Study\Classes\HK3\HĐH\ThucHanh\Lab5> ./sem
Products: 1
Products: 2
Products: 3
Products: 4
Products: 5
Products: 6
Products: 7
Products: 8
Products: 9
Products: 10
Products: 11
Products: 12
Sell: 1
Sell: 2
Sell: 3
Sell: 4
Sell: 5
Sell: 6
Sell: 7
Sell: 8
Sell: 9
Sell: 10
Sell: 11
Sell: 12
Products: 13
Products: 14
Products: 15
Products: 16
Products: 17
Products: 18
Products: 19
```

➔ Sell khi đạt giá trị = Products sẽ dừng lại cho processB (tăng Products) chạy.

- **Giải thích:**
  - Biến sells và products lưu trữ số lượng sản phẩm đã bán và đã sản xuất. (phải luôn đảm bảo sells ≤ products và products không vượt quá sells + 1517)
  - Hai semaphore sem1 và sem2, được sử dụng để đồng bộ hóa giữa hai luồng: sem1 đại diện cho điều kiện sells <= products, và sem2 đại diện cho điều kiện products <= sells + 1517. (vì vậy sem1 có value là 0 đại diện cho số sản phẩm còn lại phải

luôn không âm, sem2 có value là 1517 đại diện cho số sản phẩm tối đa còn lại không vượt quá 1517)

- Hàm processA:
  - Hàm này mô phỏng quá trình bán hàng.
  - Trong vòng lặp vô hạn, sử dụng sem_wait(&sem1) để đợi đến khi được phép bán hàng (nếu sem1 = 0, processA phải tạm dừng).
  - Nếu sem1>0 (products đang lớn hơn sells), tăng giá trị của sells và in ra thông báo về số lượng sản phẩm đã bán.
  - Sử dụng sem_post(&sem2) để tăng sem2- tức số lượng sản phẩm tối đa hiện tại được sản xuất thêm được tăng thêm 1.
- Hàm processB:
  - Hàm này mô phỏng quá trình sản xuất.
  - Trong vòng lặp vô hạn, sử dụng sem_wait(&sem2) để đợi đến khi được phép sản xuất thêm (sem2 = 0 có nghĩa là đã đạt số lượng tối đa sản phẩm được sản xuất, processB phải tạm dừng).
  - Nếu sem2 > 0, tăng giá trị của products và in ra thông báo về số lượng sản phẩm đã sản xuất.
  - Sử dụng sem_post(&sem1) để tăng sản phẩm còn lại.
- Hàm main:
  - Trong hàm main, hai semaphore được khởi tạo: sem1 = 0 và sem2 = 1517 (4 số cuối MSSV)
  - Hai thread, pA và pB, được tạo và bắt đầu chạy hàm processA và processB tương ứng.
  - Vòng lặp while(1) ở cuối để giữ cho main thread chạy mãi mãi.

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.

Trả lời:

- **Code (chưa đồng bộ):**

```c
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdlib.h>
#include <time.h>

int n=0;
int arr[1000000];

void remove_arr(int pos)
{
    for(int i=pos; i<n-1;i++)
    {
        arr[pos] = arr[pos+1];
    }
}

void *processA(void* messenge){
while(1){
    srand(time(0));
    ++n;
    arr[n-1]= rand();
    printf("After add:%d\n",n); // so phan tu trong arr
}

}

void *processB(void* messenge){
    int pos=0;
while(1){
    if(n==0) {printf("Nothing in array\n");}
    else{
     srand(time(0));
     int pos= rand()% n;
     remove_arr(pos);
    --n;
```

```
    printf("After remove:%d\n",n);
    }
}
}

int main(){
pthread_t pA,pB;
pthread_create(&pA,NULL,&processA,NULL);
pthread_create(&pB, NULL, &processB, NULL);
while(1){}
}
```

➢ **Output (chưa đồng bộ):**

```
After add:64
After add:65
After add:66
After add:67
After add:68
After add:69
After add:70
After add:71
After remove:0
After remove:71
After remove:70
After remove:69
After remove:68
After remove:67
After remove:66
After remove:65
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

After remove:3
After remove:2
After remove:1
After remove:0
Nothing in array
Nothing in array
Nothing in array
Nothing in array
Nothing in array
Nothing in array
Nothing in array
Nothing in array
Nothing in array
Nothing in array
Nothing in array
Nothing in array
```

- Xuất hiện 2 lỗi:
  - Vì không được đồng bộ nên hệ điều hành dễ ngắt một tiến trình đang ở giữa vòng lặp khiến cho một hay nhiều câu lệnh bị sót lại và thực hiện ở lần chiếm CPU sau. (câu lệnh in “After remove:0” chưa được thực hiện đã bị ngắt để nhường tài nguyên cho tiến trình khác chạy, sau đó khi tiến trình được chạy tiếp thì câu lệnh mới được thực hiện).
  - Dù là mảng đã không còn phần tử nào nhưng processB vẫn được chạy liên tục, không nhường lại cho proccessA chạy để thêm phần tử vào mảng.

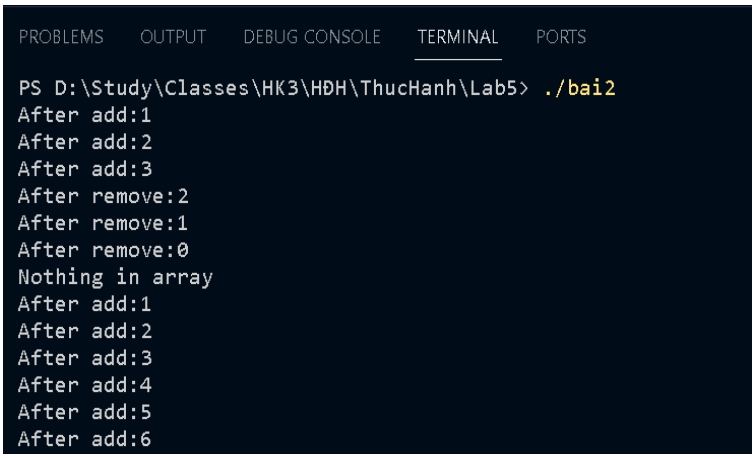
➢ **Giải pháp: đồng bộ bằng Semaphore và Mutex**

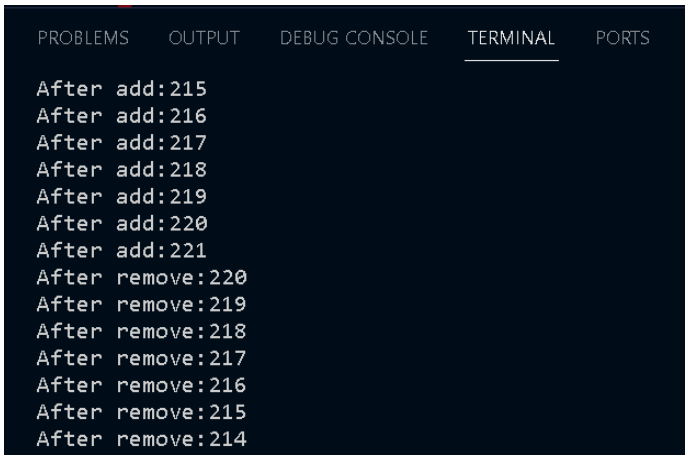
```
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdlib.h>
#include <time.h>

int n=0;
```

```
int arr[1000000];
sem_t sem;
pthread_mutex_t mutex;
void remove_arr(int pos)
{
    for(int i=pos; i<n-1;i++)
    {
        arr[pos] = arr[pos+1];
    }
}

void *processA(void* messenge){
while(1){
    pthread_mutex_lock(&mutex);
    srand(time(0));
    ++n;
    arr[n-1]= rand();
    printf("After add:%d\n",n); // so phan tu trong arr
    sem_post(&sem);
    pthread_mutex_unlock(&mutex);
}

}

void *processB(void* messenge){
    int pos=0;
while(1){
    // neu dat sem o day thi Nothing in array se khong bao gio duoc in ra
    if(n==0) {
        printf("Nothing in array\n");
    }
    sem_wait(&sem);
    pthread_mutex_lock(&mutex);
    srand(time(0));
    int pos= rand()% n;
    remove_arr(pos);
    --n;
    printf("After remove:%d\n",n);
    pthread_mutex_unlock(&mutex);
}
}

int main(){
pthread_t pA,pB;
```

```
sem_init(&sem,0,0);
pthread_mutex_init(&mutex,NULL);
pthread_create(&pA,NULL,&processA,NULL);
pthread_create(&pB, NULL, &processB, NULL);
while(1){}
}
```

➢ **Output (đã đồng bộ):**





➢ **Giải thích:**

- Biến n lưu trữ số lượng phần tử trong mảng arr.
- Mảng arr được sử dụng để lưu trữ các giá trị ngẫu nhiên được thêm vào từ processA và bị loại bỏ bởi processB.
- Semaphore sem được sử dụng để đồng bộ hóa giữa hai luồng, đảm bảo rằng processB không thực hiện loại bỏ khi mảng rỗng.
- Mutex mutex được sử dụng để đảm bảo rằng chỉ có một luồng có thể truy cập và chỉnh sửa mảng arr tại một thời điểm. Điều này giúp tránh tình trạng đọc/ghi đồng thời và xóa phần tử khỏi mảng.
- Hàm remove_arr sử dụng để loại bỏ một phần tử từ mảng tại vị trí pos.
- Hàm processA:
  - Khóa mutex để đảm bảo chỉ một luồng có thể truy cập và chỉnh sửa mảng arr tại một thời điểm.
  - Tăng giá trị của n (số lượng phần tử trong mảng).
  - Thêm một giá trị ngẫu nhiên vào mảng arr và in ra số lượng phần tử sau khi thêm.
  - Gửi một tín hiệu (sem_post) tăng giá trị cho sem (biểu thị số lượng phần tử trong mảng vừa được tăng lên).
  - Mở khóa mutex để cho phép các luồng khác tiếp tục thực hiện.
- Hàm processB:
  - Kiểm tra xem mảng có phần tử nào không. Nếu không, in ra thông báo "Nothing in array".

- Sử dụng sem_wait để xét giá trị sem (cũng như số lượng phần tử còn lại trong mảng), nếu sem=0 processB sẽ tạm dừng, nếu sem>0 thì giá trị của sem sẽ trừ đi 1 và tiếp tục thực hiện.
- Khóa mutex để đảm bảo chỉ một luồng có thể truy cập và chỉnh sửa mảng arr tại một thời điểm.
- Chọn ngẫu nhiên một vị trí pos trong mảng và loại bỏ phần tử tại vị trí đó.
- Giảm giá trị của n (số lượng phần tử trong mảng).
- In ra số lượng phần tử sau khi loại bỏ.
- Mở khóa mutex để cho phép các luồng khác tiếp tục thực hiện.

- Hàm main:
  - Trong hàm main, semaphore và mutex được khởi tạo.
  - Hai luồng, pA và pB, được tạo và bắt đầu chạy hàm processA và processB tương ứng.
  - Vòng lặp while(1) ở cuối để giữ cho luồng main chạy mãi mãi.

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.

Trả lời:

➢ **Code:**

```
#include <stdio.h>
#include <pthread.h>
```

```c
int x=0;

void *processA(void* messenge){
    while(1)
    {
        x = x + 1;
        if (x == 20)
            x = 0;
        printf("x(pA): %d\n",x);
    }

}

void *processB(void* messenge){
while(1){
        x = x + 1;
        if (x == 20)
            x = 0;
        printf("x(pB): %d\n",x);
    }
}

int main(){
    pthread_t pA, pB;
    pthread_create(&pA,NULL, &processA, NULL);
    pthread_create(&pB,NULL,&processB,NULL);
    while(1){}
    return 0;
}
```

- **Output:**

```
x(pA): 8
x(pA): 9
x(pA): 10
x(pA): 11
x(pA): 12
x(pA): 13
x(pB): 3
x(pB): 15
x(pB): 16
x(pB): 17
```

➢ **Nhận xét:**

- Cả hai luồng processA và processB đều thực hiện các phép tăng và kiểm tra giá trị của biến x mà không có cơ chế đồng bộ hóa, điều này có thể dẫn đến tình trạng đọc/ghi đồng thời và giá trị của x không được cập nhật đồng bộ như đã thấy ở output trên.

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

Trả lời:

➢ **Code:**

```c
#include <stdio.h>
#include <pthread.h>

int x=0;
pthread_mutex_t mutex;
void *processA(void* messenge){
    while(1)
    {
        pthread_mutex_lock(&mutex);
        x = x + 1;
        if (x == 20)
            x = 0;
        printf("x(pA): %d\n",x);
        pthread_mutex_unlock(&mutex);

    }

}

void *processB(void* messenge){
while(1){
        pthread_mutex_lock(&mutex);
        x = x + 1;
        if (x == 20)
            x = 0;
        printf("x(pB): %d\n",x);
        pthread_mutex_unlock(&mutex);

    }
}

int main(){
```

```
    pthread_t pA, pB;
    pthread_mutex_init(&mutex,NULL);
    pthread_create(&pA,NULL, &processA, NULL);
    pthread_create(&pB,NULL,&processB,NULL);
    while(1){}
    return 0;
}
```

➢ **Output:**

```
x(pA): 2
x(pA): 3
x(pB): 4
x(pB): 5
x(pB): 6
x(pB): 7
x(pB): 8
x(pB): 9
x(pB): 10
x(pB): 11
x(pB): 12
x(pB): 13
x(pB): 14
x(pB): 15
x(pB): 16
x(pB): 17
x(pA): 18
x(pA): 19
x(pA): 0
x(pA): 1
```

➢ **Giải thích:**

- Khi thêm mutex vào chương trình, ta sử dụng cơ chế đồng bộ hóa để đảm bảo rằng chỉ một luồng có thể thực hiện các phép tăng và kiểm tra giá trị của biến x tại một thời điểm. Điều này giúp ngăn chặn tình trạng đọc/ghi đồng thời và đảm bảo rằng các thay đổi vào biến x được thực hiện một cách an toàn.
- Trước khi thực hiện các phép tăng và kiểm tra giá trị của x, mỗi luồng đều phải khóa mutex bằng pthread_mutex_lock.
- Sau khi hoàn thành các thao tác, luồng sử dụng pthread_mutex_unlock để mở khóa mutex, cho phép các luồng khác tiếp tục thực hiện.

5.6. BÀI TẬP ÔN TẬP

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

$$w = x1 * x2; \text{ (a)}$$

$$v = x3 * x4; \text{ (b)}$$

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

Trả lời:

➢ **Code:**

```c
#include<stdlib.h>
#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>

int x1=5, x2=10, x3=15, x4=20, x5=25, x6=30, w=0, v=0, y=0, z=0, ans=0;
sem_t semAB, semCD, semCD2, semEF, semEF2, semG, semBusy;

void* processAB(void* messenge){
    while(1){
        sem_wait(&semG);
        sem_wait(&semBusy);
        w = x1*x2;
        v = x3*x4;
        printf("w = %d\n",w);
        printf("v = %d\n", v);
        sem_post(&semCD2);
        sem_post(&semAB);
        sem_post(&semBusy);
    }
}

void* processCD(void* messenge){
```

```
    while(1){
        sem_wait(&semCD2);
        sem_wait(&semAB);
        sem_wait(&semBusy);
        y= v*x5;
        z = v*x6;
        printf("y = %d\n",y);
        printf("z = %d\n", z);
        sem_post(&semAB);
        sem_post(&semCD);
        sem_post(&semEF2);
        sem_post(&semBusy);
    }
}

void* processEF(void* messenge){
    while(1){
        sem_wait(&semEF2);
        sem_wait(&semCD);
        sem_wait(&semAB);
        sem_wait(&semBusy);
        y= w * y;
        z= w * z;
        printf("y = %d\n",y);
        printf("z = %d\n", z);
        sem_post(&semEF);
        sem_post(&semBusy);
    }
}

void* processG(void* messenge){
    while(1){
        sem_wait(&semEF);
        sem_wait(&semBusy);
        ans = y+z;
        printf("ans = %d\n", ans);
        sem_post(&semBusy);
        sem_post(&semG);
    }
}
int main() {
pthread_t pAB, pCD, pEF, pG;
sem_init(&semAB, 0, 0);
sem_init(&semCD, 0, 0);
```

```
sem_init(&semCD2, 0, 0);
sem_init(&semEF, 0, 0);
sem_init(&semEF2, 0, 0);
sem_init(&semG, 0, 1);
sem_init(&semBusy, 0, 1);
pthread_create(&pAB, NULL, &processAB, NULL);
pthread_create(&pCD, NULL, &processCD, NULL);
pthread_create(&pEF, NULL, &processEF, NULL);
pthread_create(&pG, NULL, &processG, NULL);
while (1){}
return 0;
}
```

- **Output:**

```
w = 50
v = 300
y = 7500
z = 9000
y = 375000
z = 450000
ans = 825000
w = 50
v = 300
y = 7500
z = 9000
y = 375000
z = 450000
ans = 825000
```