

Họ và tên: Nguyễn Gia Bảo

Mã số sinh viên: 22520109

Lớp: IT007.O29.1

## 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	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Chụp hình minh chứng	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Giải thích kết quả	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

#### 5.6. BÀI TẬP ÔN TẬP

	BT 1
Trình bày cách làm	<input checked="" type="checkbox"/>
Chụp hình minh chứng	<input checked="" type="checkbox"/>
Giải thích kết quả	<input checked="" type="checkbox"/>

#### Tự chấm điểm:

\*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]

Trả lời...

- Ý tưởng thực hiện hiện: Dùng semaphore để đồng bộ 2 tiêu trình

```
c bt1.c > product_add1(void *)
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <semaphore.h>
4
5 sem_t s1, s2;
6 int sells = 0;
7 int products = 0;
8
9 void *sell_add1(void *threadid){
10     while(1){
11         sem_wait(&s1);
12         sells++;
13         printf("sells = %d\n", sells);
14         sem_post(&s2);
15     }
16 }
17
18 void *product_add1(void *threadid){
19     while(1){
20         sem_wait(&s2);
21         products++;
22         printf("products = %d\n", products);
23         sem_post(&s1);
24     }
25 }
26
27 int main(){
28     sem_init(&s1, 0, 0);
29     sem_init(&s2, 0, 109);
30     pthread_t idthread1, idthread2;
31     pthread_create(&idthread1, NULL, sell_add1, NULL);
32     pthread_create(&idthread2, NULL, product_add1, NULL);
33     pthread_exit(NULL);
34 }
```

Code bài tập 1

```
products = 73690
products = 73691
products = 73692
products = 73693
products = 73694
sells = 73586
sells = 73587
sells = 73588
sells = 73589
sells = 73590
sells = 73591
sells = 73592
sells = 73593
sells = 73594
sells = 73595
products = 73695
products = 73696
products = 73697
products = 73698
products = 73699
products = 73700
products = 73701
products = 73702
products = 73703
products = 73704
sells = 73596
sells = 73597
sells = 73598
sells = 73599
sells = 73600
sells = 73601
sells = 73602
sells = 73603
sells = 73604
sells = 73605
sells = 73606
sells = 73607
sells = 73608
```

*Kết quả khi chạy*

- Giải thích:
  - Ta có 2 điều kiện : sells  $\leq$  products và products  $\leq$  sells + 109 ( MSSV 22520109)

và ban đầu sells = products = 0 → để tăng sells thì products phải tăng trước và có thể tăng products 109 lần trước khi tăng sells.

- Khởi tạo semaphore s1 với giá trị ban đầu là 0 và semaphore s2 với giá trị ban đầu là 109 bằng sem\_init().
- Đặt sem\_wait(&s1) trước khi thực hiện sells++, sem\_wait(&s2) trước khi thực hiện products++; sem\_post(&s1) sau products++, sem\_post(&s2) sau sells++.
- Sử dụng pthread\_create() để tạo 2 tiêu trình thực hiện 2 công việc trên.

## 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 trước khi đồng bộ bằng semaphore:**

```
c bt2.c > ...
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <semaphore.h>
5 #include <unistd.h>
6
7 int a[50];
8 int n = 0;
9
10 sem_t s;
11
12 void *add(void *threadid)
13 {
14     while(1)
15     {
16         srand(time(NULL));
17         int randomNumber = rand() % 100 + 1;
18
19         //sem_wait(&s);
20
21         a[n] = randomNumber;
22         n++;
23         printf("Add %d \n", randomNumber);
24         printf("Remaining %d \n", n);
25
26         //sem_post(&s);
27         sleep(1);
28     }
29 }
```

```
30  void *remove_(void *threadid)
31  {
32      while (1)
33      {
34          //sem_wait(&s);
35          if (n == 0)
36              printf("Nothing in array a\n");
37          else
38          {
39              int rm = a[n - 1];
40              n--;
41              printf("Remove %d \n", rm);
42              printf("Remaining %d \n", n);
43          }
44          //sem_post(&s);
45          sleep(1);
46      }
47  }
48
49  int main()
50  {
51      //sem_init(&s, 0, 1);
52      pthread_t idthread1, idthread2;
53
54      pthread_create(&idthread1, NULL, add, NULL);
55      pthread_create(&idthread2, NULL, remove_, NULL);
56
57      pthread_join(idthread1, NULL);
58      pthread_join(idthread2, NULL);
59  }
60
```

✿ Thực thi code trước khi đóng bộ :

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Thân Thế Tùng

```
Add 3
Remaining 1
Remove 3
Remaining 0
Nothing in array a
Add 65
Remaining 1
Remove 65
Remaining 1
Add 46
Remaining 1
Remove 46
Remaining 1
Add 33
Remaining 1
Remove 33
Remaining 0
Add 45
Remaining 1
Remove 45
Remaining 0
Add 46
Remaining 1
Remove 46
Remaining 0
Add 28
Remaining 1
Remove 28
Remaining 1
Add 41
Remaining 1
Remove 41
Remaining 0
Add 4
Remaining 1
Remove 4
Remaining 0
Add 32
Remaining 1
Remove 32
Remaining 0
Add 15
Remaining 1
^c
nbg@MST1:~/codeLab5$
```

Thực hiện khi có lệnh sleep(1) trong các hàm add và remove\_

Thực hiện không có lệnh sleep(1) trong các hàm add và remove\_

### ➡ Code sau khi đồng bộ bằng semaphore:

- Khởi tạo một semaphore s, đặt lệnh `sem_wait(&s)` trước khi tiến trình thực hiện công việc và lệnh `sem_post(&s)` sau khi tiến trình hoàn thành công việc.

```
c bt2-semaphore.c ...
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <semaphore.h>
5
6 int a[10000];
7 int n = 0;
8
9 sem_t s;
10
11 void *add(void *threadid)
12 {
13     while (1)
14     {
15         srand(time(NULL));
16         int randomNumber = rand() % 100 + 1;
17
18         sem_wait(&s);
19
20         a[n] = randomNumber;
21         n++;
22         printf("Add %d \n", randomNumber);
23         printf("Remaining %d \n", n);
24         sem_post(&s);
25     }
26 }
27 void *remove_(void *threadid)
```

```
27 void *remove_(void *threadid)
28 {
29     while (1)
30     {
31         sem_wait(&s);
32         if (n == 0)
33             printf("Nothing in array a\n");
34         else
35         {
36             int rm = a[n - 1]; // xóa phần tử cuối của mảng a
37             n--;
38             printf("Remove %d \n", rm);
39             printf("Remaining %d \n", n);
40         }
41         sem_post(&s);
42     }
43 }
44
45 int main()
46 {
47     sem_init(&s, 0, 1);
48     pthread_t idthread1, idthread2;
49
50     pthread_create(&idthread1, NULL, add, NULL);
51     pthread_create(&idthread2, NULL, remove_, NULL);
52
53     pthread_join(idthread1, NULL);
54     pthread_join(idthread2, NULL);
55 }
```

✳ Thực hiện code sau khi đã đồng bộ với semaphore:

```
Remaining 29
Add 1
Remaining 30
Add 1
Remaining 31
Add 1
Remaining 32
Add 1
Remaining 33
Add 1
Remaining 34
Add 1
Remaining 35
Add 1
Remaining 36
Add 1
Remaining 37
Add 1
Remaining 38
Add 1
Remaining 39
Add 1
Remaining 40
Add 1
Remaining 41
Add 1
Remaining 42
Add 1
Remaining 43
Add 1
Remaining 44
Add 1
Remaining 45
Add 1
Remaining 46
Add 1
Remaining 47
Add 1
Remaining 48
Add 1
Remaining 49
Remove 1
Remaining 48
^C
○ ngb@MSI:~/codeLab5$
```

✳ Giải thích:

- Khi chạy chương trình chưa đồng bộ với semaphore, nếu không có lệnh sleep , chương trình sẽ dẫn đến race condition và dẫn đến lỗi
- Khi đồng bộ 2 chương trình với semaphore, chương trình sẽ đảm bảo chạy tốt và không có lỗi, vì lệnh `sem_wait(&s)` và `sem_post(&s)` giúp ngăn chặn việc các tiến trình tranh chấp, một tiến trình sẽ phải chờ đợi đến khi tiến trình kia hoàn tất công việc với các biến a và n rồi mới được thay đổi các biến trên.

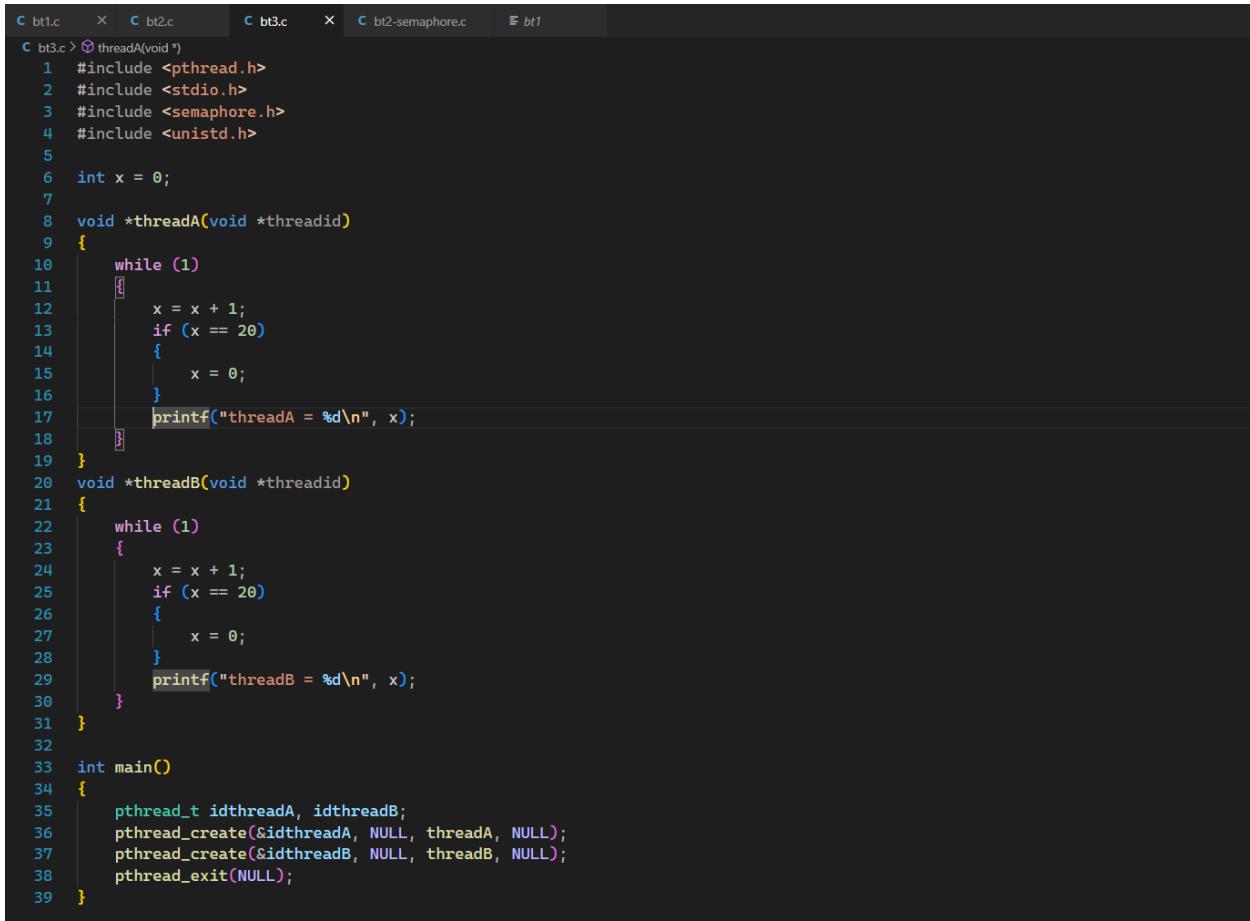
**3. Cho 2 process A và B chạy song song như sau:**

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 thực hiện yêu cầu của đề:



```
C bt1.c    X  C bt2.c    X  C bt3.c    X  C bt2-semaphore.c  bt1
C bt3.c > threadA(void *)
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5
6 int x = 0;
7
8 void *threadA(void *threadid)
9 {
10     while (1)
11     {
12         x = x + 1;
13         if (x == 20)
14         {
15             x = 0;
16         }
17         printf("threadA = %d\n", x);
18     }
19 }
20 void *threadB(void *threadid)
21 {
22     while (1)
23     {
24         x = x + 1;
25         if (x == 20)
26         {
27             x = 0;
28         }
29         printf("threadB = %d\n", x);
30     }
31 }
32
33 int main()
34 {
35     pthread_t idthreadA, idthreadB;
36     pthread_create(&idthreadA, NULL, threadA, NULL);
37     pthread_create(&idthreadB, NULL, threadB, NULL);
38     pthread_exit(NULL);
39 }
```

💡 Kết quả khi chạy chương trình trên:

```
OUTPUT PROBLEMS PORTS TERMINAL bash - codeLab5 + □ ...
```

```
threadB = 9
threadB = 10
threadB = 11
threadB = 12
threadB = 13
threadB = 14
threadB = 15
threadB = 16
threadB = 17
threadB = 18
threadB = 19
threadB = 0
threadB = 1
threadB = 2
threadB = 3
threadB = 4
threadB = 5
threadA = 6
threadA = 7
threadA = 8
threadA = 9
threadA = 10
threadA = 11
threadA = 12
threadA = 13
threadA = 14
threadA = 15
threadA = 16
threadA = 17
threadA = 18
threadA = 19
threadA = 0
threadA = 1
threadA = 2
threadA = 3
threadA = 4
threadA = 5
threadA = 6
threadA = 7
threadA = 8
threadA = 9
threadA = 10
threadA = 11
threadA = 12
threadA = 13
```

💡 Nhận xét:

- Kết quả in ra các giá trị bị ngắt quãng do 2 tiến trình đều thực hiện đồng thời và cùng tác động thay đổi giá trị biến x mà không có sự đồng bộ, dẫn đến việc các tiến trình in kết quả chồng lên nhau, không nhất quán từ 1 → 19 rồi lặp lại như ta muốn.

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

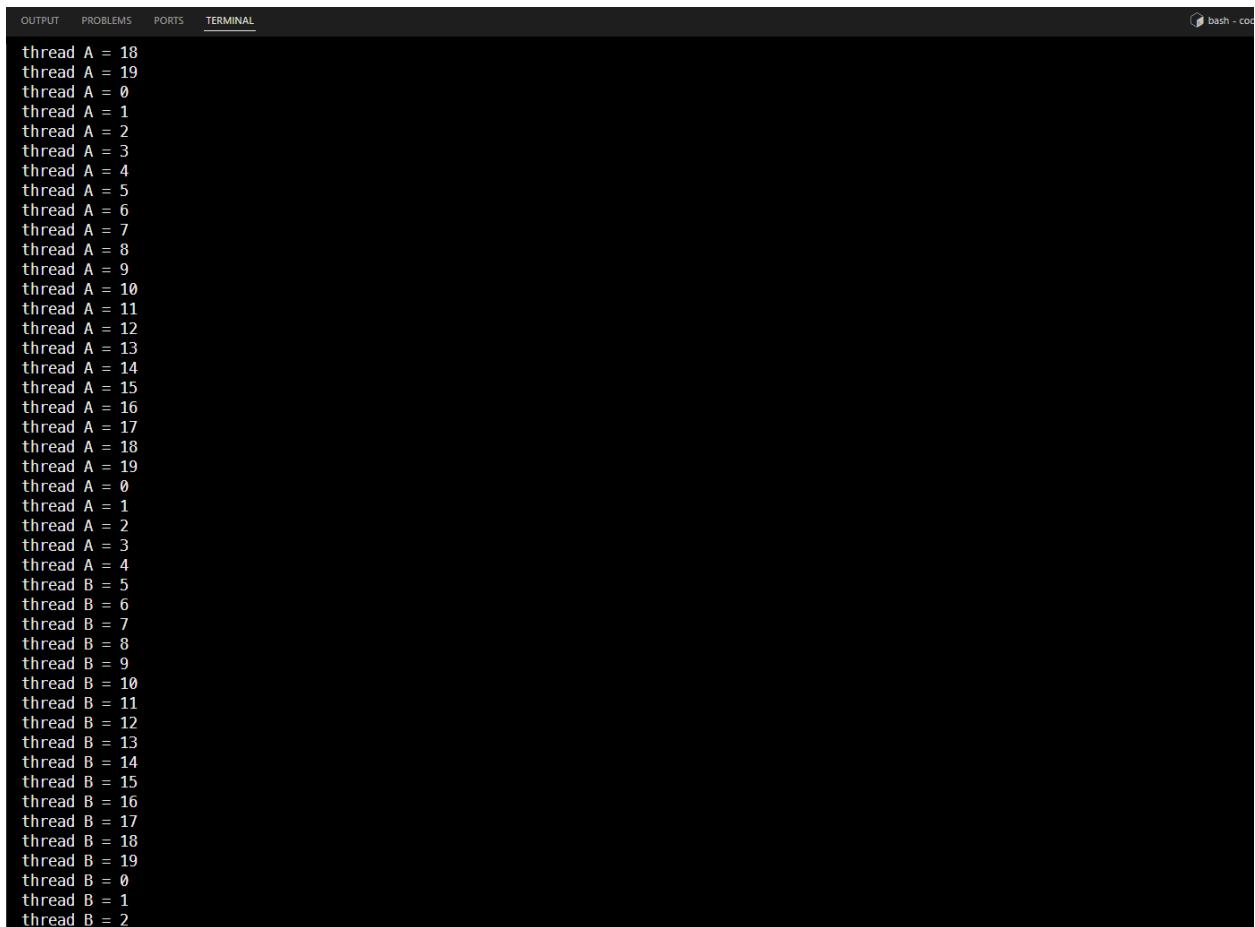
Trả lời...

✚ **Ý tưởng thực hiện:** dùng mutex để đồng bộ 2 tiêu trình A và B

✚ **Code thực hiện :**

```
C bt1.c          C bt2.c          C bt3.c          C bt4.c          C bt2-semaphore.c      bt1
C bt4.c > threadA(void *)
  1 #include <pthread.h>
  2 #include <stdio.h>
  3 #include <semaphore.h>
  4 #include <unistd.h>
  5
  6 int x = 0;
  7 pthread_mutex_t m;
  8
  9 void *threadA(void *threadid)
10 {
11     while (1)
12     {
13         pthread_mutex_lock(&m);
14         x = x + 1;
15         if (x == 20)
16         {
17             x = 0;
18         }
19         printf("thread A = %d\n", x);
20         pthread_mutex_unlock(&m);
21     }
22 }
23 void *threadB(void *threadid)
24 {
25     while (1)
26     {
27         pthread_mutex_lock(&m);
28         x = x + 1;
29         if (x == 20)
30         {
31             x = 0;
32         }
33         printf("thread B = %d\n", x);
34         pthread_mutex_unlock(&m);
35     }
36 }
37
38 √ int main()
39 {
40     pthread_mutex_init(&m, NULL);
41     pthread_t idthreadA, idthreadB;
42     pthread_create(&idthreadA, NULL, threadA, NULL);
43     pthread_create(&idthreadB, NULL, threadB, NULL);
44     pthread_exit(NULL);
45 }
```

### ➡ Kết quả thực hiện:



```
thread A = 18
thread A = 19
thread A = 0
thread A = 1
thread A = 2
thread A = 3
thread A = 4
thread A = 5
thread A = 6
thread A = 7
thread A = 8
thread A = 9
thread A = 10
thread A = 11
thread A = 12
thread A = 13
thread A = 14
thread A = 15
thread A = 16
thread A = 17
thread A = 18
thread A = 19
thread A = 0
thread A = 1
thread A = 2
thread A = 3
thread A = 4
thread B = 5
thread B = 6
thread B = 7
thread B = 8
thread B = 9
thread B = 10
thread B = 11
thread B = 12
thread B = 13
thread B = 14
thread B = 15
thread B = 16
thread B = 17
thread B = 18
thread B = 19
thread B = 0
thread B = 1
thread B = 2
```

### ➡ Giải thích:

- Khai báo một mutex  $m$ , trước khi mỗi tiến trình thực hiện công việc, chúng ta sẽ khóa mutex lại bằng lệnh “`pthread_mutex_lock(&m)`”, khiến cho tiến trình còn lại không thể can thiệp thay đổi dữ liệu của biến  $x$ . Sau khi một tiến trình thực hiện xong, ta sẽ mở khóa mutex bằng lệnh “`pthread_mutex_unlock(&m)`”, để tiến trình còn lại có thể làm việc với biến  $x$ .
- Sau khi áp dụng mutex, kết quả của biến  $x$  đã lần lượt được in ra từ  $0 \rightarrow 19$  rồi lặp lại.

## 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...

✚ **Ý tưởng:** Dùng semaphore để đồng bộ các tiêu trình

✚ **Code thực hiện:**

# Báo cáo thực hành môn Hệ điều hành - Giảng viên: Thân Thê Tùng

```
bt1.c      X  bt2.c      C  bt3.c      C  bt4.c      C  bt5.c      X  bt2-semaphore.c  bt7
C bt5.c > ...
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <semaphore.h>
4
5  int w, v, y, z, ans;
6  int x1 = 2, x2 = 3, x3 = 1, x4 = 5, x5 = 4, x6 = 3;
7
8  sem_t s1, s2, s3, s4, s5;
9
10 void *a(void *threadid)
11 {
12     printf("a :\n");
13     w = x1 * x2;
14     sem_post(&s4);
15 }
16
17 void *b(void *threadid)
18 {
19     printf("b :\n");
20     v = x3 * x4;
21     sem_post(&s1);
22     sem_post(&s1);
23 }?
24
25 void *c(void *threadid)
26 {
27     sem_wait(&s1);
28     printf("c :\n");
29     y = v * x5;
30     sem_post(&s2);
31 }
32
33 void *d(void *threadid)
34 {
35     sem_wait(&s1);
36     printf("d :\n");
37     z = v * x6;
38 }

39 void *e(void *threadid)
40 {
41     sem_wait(&s2);
42     sem_wait(&s4);
43     printf("e :\n");
44     y = w * y;
45     sem_post(&s3);
46 }
47
48 void *f(void *threadid)
49 {
50     printf("f :\n");
51     z = w * z;
52     sem_post(&s5);
53 }
54
55 void *g(void *threadid)
56 {
57     sem_wait(&s3);
58     sem_wait(&s5);
59     printf("g :\n");
60
61     ans = y + z;
62 }
```

```
64
65 int main()
66 {
67     sem_init(&s1, 0, 0);
68     sem_init(&s2, 0, 0);
69     sem_init(&s3, 0, 0);
70     sem_init(&s4, 0, 0);
71     sem_init(&s5, 0, 0);
72
73     pthread_t idthread1, idthread2, idthread3, idthread4, idthread5, idthread6, idthread7;
74
75     pthread_create(&idthread1, NULL, a, NULL);
76     pthread_create(&idthread2, NULL, b, NULL);
77     pthread_create(&idthread3, NULL, c, NULL);
78     pthread_create(&idthread4, NULL, d, NULL);
79     pthread_create(&idthread5, NULL, e, NULL);
80     pthread_create(&idthread6, NULL, f, NULL);
81     pthread_create(&idthread7, NULL, g, NULL);
82
83     pthread_join(idthread1, NULL);
84     pthread_join(idthread2, NULL);
85     pthread_join(idthread3, NULL);
86     pthread_join(idthread4, NULL);
87     pthread_join(idthread5, NULL);
88     pthread_join(idthread6, NULL);
89     pthread_join(idthread7, NULL);
90 }
```

## ✳ Kết quả thực hiện:

```
OUTPUT PROBLEMS PORTS TERMINAL
• ngb@MSI:~/codeLab5$ gcc -o bt5 bt5.c
• ngb@MSI:~/codeLab5$ ./bt5
a :
b :
c :
d :
e :
f :
g :
• ngb@MSI:~/codeLab5$
```

*Thực hiện các tiêu trình theo thứ tự*

## ✳ Giải thích:

- Tạo 5 semaphore s1, s2, s3, s4, s5 với giá trị khởi đầu là 0 (có thể sử dụng mutex).
- Để (c), (d) chỉ được thực hiện sau khi v được tính ta đặt sem\_wait(&s1) trước (c), (d) và 2 lần sem\_post(&s1) sau (b).
- Để (e) chỉ được thực hiện sau khi w và y được tính ta đặt sem\_wait(&s2) và sem\_wait(&s4) trước (e), sem\_post(&s2) và sem\_post(&s4) sau (c), (a).
- Để (g) chỉ được thực hiện sau khi y và z được tính ta đặt sem\_wait(&s3) và sem\_wait(&s5) trước (g), sem\_post(&s3) và sem\_post(&s5) sau (e), (f).

## Báo cáo thực hành môn Hệ điều hành - Giảng viên: Thân Thê Tùng

- Sử dụng `pthread_create()` để tạo 7 tiêu trình thực hiện 7 công việc.
- Sử dụng `pthread_join()` để đảm bảo rằng tất cả tiêu trình đều đã thực hiện xong công việc trước khi thoát chương trình.