

Name: Nguyễn Khoa Nguyễn

ID: 24521190

Class: IT007.Q15.1

OPERATING SYSTEM LAB 05'S REPORT

SUMMARY

	Task	Status	Page
Section 1.5	Task name 1	Hoàn thành	1
	Task name 2	Hoàn thành	4
	Task name 3	Hoàn thành	6
	Task name 4	Hoàn thành	8

Self-scores:

*Note: Export file to **PDF** and name the file by following format:
Student ID _LABx.pdf

Section 5.5

1. Task name 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]

- Ta có sells <= products <= sells + 1190

- Nội dung code:

```
1  #include<stdio.h>
2  #include<semaphore.h>
3  #include<pthread.h>
4
5  int sells, products;
6  int MSSV = 1190; //24521190
7  sem_t sem;
8
9  void* Sells(void *arg)
10 {
11     while(sells <= products)
12     {
13         sem_wait(&sem);
14         sells++;
15         printf("Sells: %d\n", sells);
16     }
17     return NULL;
18 }
19 void *Products(void *arg)
20 {
21     while(products <= sells + MSSV)
22     {
23         products++;
24         printf("Products: %d\n", products);
25         sem_post(&sem);
26     }
27     return NULL;
28 }
29 int main()
30 {
31     pthread_t t1, t2;
32     sells = 0;
33     products = 0;
34     sem_init(&sem, 0, 0);
35     pthread_create(&t1, NULL, Sells, NULL);
36     pthread_create(&t2, NULL, Products, NULL);
37
38     pthread_join(t1, NULL);
39     pthread_join(t2, NULL);
40     sem_destroy(&sem);
41     return 0;
42 }
```

Hình 1 Nội dung code bài 1

- Kết quả:

```
nguyenkhoanguyen@nguyenkhoanguyen-Inspiron-16-Plus-7630: ~ cd Documents/IT007/LAB05  
nguyenkhoanguyen@nguyenkhoanguyen-Inspiron-16-Plus-7630: ~/Documents/IT007/LAB05$ gcc Bai1.c -o Bai1  
nguyenkhoanguyen@nguyenkhoanguyen-Inspiron-16-Plus-7630: ~/Documents/IT007/LAB05$ ./Bai1  
Products: 1  
Products: 2  
Products: 3  
Products: 4  
Products: 5  
Products: 6  
Products: 7  
Products: 8  
Products: 9  
Products: 10  
Products: 11  
Products: 12  
Sells: 1  
Sells: 2  
Sells: 3  
Sells: 4  
Sells: 5  
Sells: 6  
Sells: 7  
Sells: 8  
Sells: 9  
Sells: 10  
Sells: 11  
Sells: 12  
Products: 13  
Products: 14  
Products: 15  
Sells: 13  
Sells: 14
```

Hình 2 Kết quả bài 1 (1)

- Có hai luồng là Products và Sells chạy song song (Products tăng số lượng sản phẩm mỗi khi có thể đồng thời cho luồng Sells biết có thêm sản phẩm thông qua hàm sem_post.

```
nguyenkhoanguyen@nguyenkhoanguyen-Inspiron-16-Plus-7630: ~ cd Documents/IT007/LAB05  
nguyenkhoanguyen@nguyenkhoanguyen-Inspiron-16-Plus-7630: ~/Documents/IT007/LAB05$ gcc Bai1.c -o Bai1  
nguyenkhoanguyen@nguyenkhoanguyen-Inspiron-16-Plus-7630: ~/Documents/IT007/LAB05$ ./Bai1  
Products: 1757  
Products: 1758  
Products: 1759  
Products: 1760  
Products: 1761  
Products: 1762  
Products: 1763  
Products: 1764  
Products: 1765  
Products: 1766  
Products: 1767  
Sells: 576  
Sells: 577  
Sells: 578  
Sells: 579  
Sells: 580  
Sells: 581  
Sells: 582  
Sells: 583  
Sells: 584  
Sells: 585  
Sells: 586  
Sells: 587  
Sells: 588  
Sells: 589  
Sells: 590  
Sells: 591  
Sells: 592  
Sells: 593  
Sells: 594  
Sells: 595  
Sells: 596  
Sells: 597
```

Hình 3 Kết quả bài 1 (2)

- Luồng Products dừng lại khi số lượng sản phẩm vượt quá số lượng hàng đã bán cộng với 4 số cuối của MSSV. Cụ thể, Products dừng lại ở giá trị 1767 khi Sells đạt được 577 cộng với 1190 chính là 4 số cuối của MSSV.
- Sells chờ hàm sem_wait() trước khi tăng lượng hàng và sẽ bị chặn nếu không có tín hiệu
- Kết quả được in ra thể hiện quá trình sản xuất và buôn bán với số lượng sản phẩm sản xuất ra tăng theo thời gian cùng với số lượng sản phẩm được bán ra.

2. Task name 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”.
- Nội dung code:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <unistd.h>
5 #include <semaphore.h>
6 #include <time.h>
7 #define N 100
8 int a[N];
9 int count = 0;
10 sem_t sem;
11
12 // --- Thread 1: Producer ---
13 void* Producer(void* arg) {
14     while(1) {
15         int num = rand() % 100 + 1;
16         sem_wait(&sem);
17         if (count < N) {
18             a[count] = num;
19             count++;
20             printf("[Producer] Them %d. So phan tu: %d\n", num, count);
21         } else {
22             printf("[Producer] Mang day (Full).\n");
23         }
24         sem_post(&sem);
25         sleep(1);
26     }
27     return NULL;
28 }

```

Hình 4 Nội dung code bài 2 (1)

The screenshot shows a code editor window with the file 'Bai2.c' open. The code implements a producer-consumer pattern using threads and semaphores. The 'Consumer' function reads elements from an array 'a' until it is empty, printing each element. The 'main' function creates two threads, 't1' and 't2', where 't1' is the producer adding values to 'a' and 't2' is the consumer reading them. The code uses 'sem_init' to initialize a semaphore with value 1, 'sem_wait' to wait for it, and 'sem_post' to increment it. It also includes a sleep operation for thread synchronization.

```
30 void* Consumer(void* arg) {
31     while(1) {
32         sem_wait(&sem);
33         if (count > 0) {
34             int val = a[count - 1];
35             count--;
36             printf("[Consumer] Lay %d. So phan tu: %d\n", val, count);
37         } else {
38             printf("Nothing in array a\n");
39         }
40         sem_post(&sem);
41         sleep(2);
42     }
43     return NULL;
44 }
45
46 int main() {
47     pthread_t t1, t2;
48     srand(time(NULL));
49     sem_init(&sem, 0, 1);
50     pthread_create(&t1, NULL, Producer, NULL);
51     pthread_create(&t2, NULL, Consumer, NULL);
52     pthread_join(t1, NULL);
53     pthread_join(t2, NULL);
54     sem_destroy(&sem);
55
56     return 0;
57 }
```

Hình 5 Nội dung code bài 2 (2)

- Kết quả:

The screenshot shows a terminal window on an SSH connection to a Linux system. The user runs the command 'gcc Bai2.c -o Bai2 -pthread' to compile the program. After compilation, they run './Bai2'. The output shows the producer thread adding values to the array and the consumer thread reading them. The consumer prints 'Nothing in array a' initially because the array is empty. As values are added, the consumer prints them sequentially. The producer continues to add values, and the consumer continues to read them, demonstrating a continuous loop of production and consumption.

```
nguyenkhoanguyen@nguyenkhoanguyen-Inspiron-16-Plus-7630:~/Documents/IT007/LAB05
nguyenkhoanguyen@nguyenkhoanguyen-Inspiron-16-Plus-7630:~/Documents/IT007/LAB05$ gcc Bai2.c -o Bai2 -pthread
nguyenkhoanguyen@nguyenkhoanguyen-Inspiron-16-Plus-7630:~/Documents/IT007/LAB05$ ./Bai2
Nothing in array a
[Producer] Them 29. So phan tu: 1
[Producer] Them 65. So phan tu: 2
[Consumer] Lay 65. So phan tu: 1
[Producer] Them 43. So phan tu: 2
[Producer] Them 52. So phan tu: 3
[Consumer] Lay 52. So phan tu: 2
[Producer] Them 80. So phan tu: 3
[Producer] Them 92. So phan tu: 4
[Consumer] Lay 92. So phan tu: 3
[Producer] Them 32. So phan tu: 4
[Producer] Them 11. So phan tu: 5
[Consumer] Lay 11. So phan tu: 4
[Producer] Them 61. So phan tu: 5
[Producer] Them 16. So phan tu: 6
[Consumer] Lay 16. So phan tu: 5
[Producer] Them 26. So phan tu: 6
[Producer] Them 54. So phan tu: 7
[Consumer] Lay 54. So phan tu: 6
[Producer] Them 16. So phan tu: 7
[Producer] Them 21. So phan tu: 8
[Consumer] Lay 21. So phan tu: 7
[Producer] Them 26. So phan tu: 8
[Producer] Them 29. So phan tu: 9
[Consumer] Lay 29. So phan tu: 8
[Producer] Them 66. So phan tu: 9
[Producer] Them 13. So phan tu: 10
[Consumer] Lay 13. So phan tu: 9
[Producer] Them 97. So phan tu: 10
^Z
```

Hình 6 Kết quả bài 2

- Giải thích:
 - Đầu tiên khởi tạo ra 1 khóa semaphore bằng lệnh sem_init().
 - Với luồng sản xuất, ta lấy khóa bằng hoặc đợi nếu không có khóa bằng lệnh sem_wait(). Sau đó thêm các số ngẫu nhiên vào mảng a và trả khóa lại sau khi thêm xong bằng lệnh sem_post()
 - Với luồng tiêu thụ, lấy chia khóa bằng lệnh sem_wait() và lấy từng số trong mảng ra. Sau khi lấy xong thì trả lại khóa bằng lệnh sem_post()

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

PROCESS A	PROCESS B
<pre>processA() { while(1){ x = x + 1; if (x == 20) x = 0; print(x); } }</pre>	<pre>processB() { while(1){ x = x + 1; if (x == 20) x = 0; print(x); } }</pre>

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

- Nội dung code:

```
1 #include<stdio.h>
2 #include<pthread.h>
3 int x = 0;
4 void *ProcessA(void *arg)
5 {
6     while(1){
7         x = x + 1;
8         if (x == 20)
9             x = 0;
10        printf("ProcessA: %d\n",x);
11    }
12 }
13 void *ProcessB(void *arg)
14 {
15     while(1)
16     {
17         x = x + 1;
18         if (x == 20)
19             x = 0;
20         printf("ProcessB: %d\n",x);
21     }
22 }
23 int main()
24 {
25     pthread_t threadA, threadB;
26     pthread_create(&threadA, NULL, ProcessA, NULL);
27     pthread_create(&threadB, NULL, ProcessB, NULL);
28     pthread_join(threadA, NULL);
29     pthread_join(threadB, NULL);
30     return 0;
31 }
```

Hình 7 Nội dung code bài 3

- Kết quả:

```
ProcessB: 15
ProcessB: 16
ProcessB: 17
ProcessB: 18
ProcessB: 19
ProcessB: 0
ProcessB: 1
ProcessB: 2
ProcessB: 3
ProcessB: 4
ProcessB: 5
ProcessB: 6
ProcessB: 7
ProcessB: 8
ProcessB: 9
ProcessB: 10
ProcessA: 12
ProcessA: 12
ProcessA: 13
ProcessA: 14
ProcessA: 15
ProcessA: 16
ProcessA: 17
ProcessB: 11
ProcessB: 19
ProcessB: 0
ProcessB: 1
ProcessB: 2
ProcessB: 3
ProcessB: 4
ProcessB: 5
ProcessB: 6
ProcessB: 7
```

Hình 8 Kết quả bài 3

- Nhận xét: Đoạn code trên đang gặp vấn đề Race Condition vì hai luồng truy cập vào biến x mà không có cơ chế bảo vệ. A có thể đọc x, chưa kịp tăng thì B cũng đọc x và cả hai cùng tăng x lên. Từ đó làm sai lệch giá trị mong muốn của x. Hơn thế nữa việc in ra màn hình kết quả cũng bị chen ngang và các dòng lẩn lộn không theo thứ tự

4. Task name 4: Đồng bộ với mutex để sửa lỗi bất hợp lý trong kết quả của mô hình bài 3

- Nội dung code:

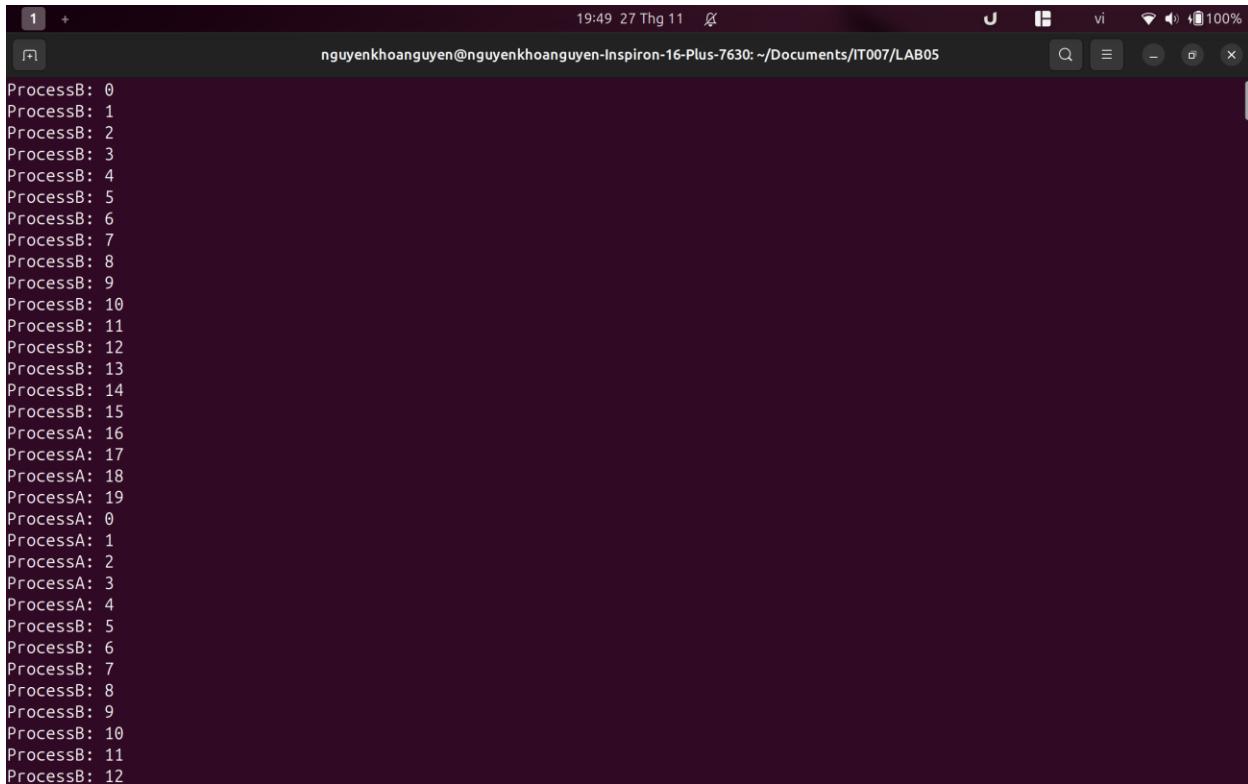
```

1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4
5 int x = 0;
6 pthread_mutex_t lock;
7
8 void *ProcessA(void *arg)
9 {
10     while(1){
11         pthread_mutex_lock(&lock);
12         x = x + 1;
13         if (x == 20)
14             x = 0;
15         printf("ProcessA: %d\n", x);
16         pthread_mutex_unlock(&lock);
17     }
18     return NULL;
19 }
20
21 void *ProcessB(void *arg)
22 {
23     while(1){
24         pthread_mutex_lock(&lock);
25         x = x + 1;
26         if (x == 20)
27             x = 0;
28         printf("ProcessB: %d\n", x);
29         pthread_mutex_unlock(&lock);
30     }
31     return NULL;
32 }
33
34 int main()
35 {
36     pthread_t threadA, threadB;
37     if (pthread_mutex_init(&lock, NULL) != 0) {
38         printf("Khởi tạo Mutex thất bại\n");
39         return 1;
40     }
41     pthread_create(&threadA, NULL, ProcessA, NULL);
42     pthread_create(&threadB, NULL, ProcessB, NULL);
43     pthread_join(threadA, NULL);
44     pthread_join(threadB, NULL);
45     pthread_mutex_destroy(&lock);
46
47     return 0;
48 }

```

Hình 9 Nội dung code bài 4

- Kết quả:



```
ProcessB: 0
ProcessB: 1
ProcessB: 2
ProcessB: 3
ProcessB: 4
ProcessB: 5
ProcessB: 6
ProcessB: 7
ProcessB: 8
ProcessB: 9
ProcessB: 10
ProcessB: 11
ProcessB: 12
ProcessB: 13
ProcessB: 14
ProcessB: 15
ProcessA: 16
ProcessA: 17
ProcessA: 18
ProcessA: 19
ProcessA: 0
ProcessA: 1
ProcessA: 2
ProcessA: 3
ProcessA: 4
ProcessB: 5
ProcessB: 6
ProcessB: 7
ProcessB: 8
ProcessB: 9
ProcessB: 10
ProcessB: 11
ProcessB: 12
```

Hình 10 Kết quả bài 4

- Giải thích:
 - pthread_mutex_lock: Thread nào nhanh hơn sẽ lấy khóa và chốt khóa lại. Thread còn lại đến sau bắt buộc phải đứng chờ bên ngoài
 - Thread giữ khóa tăng giá trị x và in ra màn hình mà không bị ai can thiệp. Như trong kết quả ta thấy B in ra giá trị x từ 0 đến 15 mà không bị ai can thiệp, sau đó mới cho A tiếp tục công việc
 - pthread_mutex_unlock: làm xong việc thì thread mở khóa để nhường quyền cho thread đang chờ bên ngoài. Ta thấy trong kết quả thì A tiếp tục công việc của B, vì vậy ta có được giá trị x tăng theo tuần tự mà không bị lỗi tính toán và in lộn xộn