

Lab 5

BÁO CÁO BÀI THỰC HÀNH SỐ 5

ĐỒNG BỘ

TIẾN TRÌNH, TIỂU TRÌNH

Môn học: Hệ điều hành (IT007)

Sinh viên thực hiện	Nguyễn Đức Tấn
Thời gian thực hiện	17/05/2024

THỰC HÀNH:

Bài 1: Hiện thực hóa mô hình trong ví dụ 5.3.1.2, tuy nhiên thay bằng điều kiện sau:
 $\text{sells} \leq \text{products} \leq \text{sells} + [2 \text{ số cuối của MSSV} + 10]$

MSSV: 2(22521303) + 10 = 13

Cách giải: đặt 2 biến semaphore để kiểm soát điều kiện:

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

int sells = 0;
int products = 0;

sem_t sem;
sem_t sem2;
// sells < products < sells + 2(22521303) + 10 = sells + 13
void *processA(void *mess){
    while(1){
        sem_wait(&sem);
        sells++;
        printf("SELLs = %d\n", sells);
        sem_post(&sem2);
    }
}

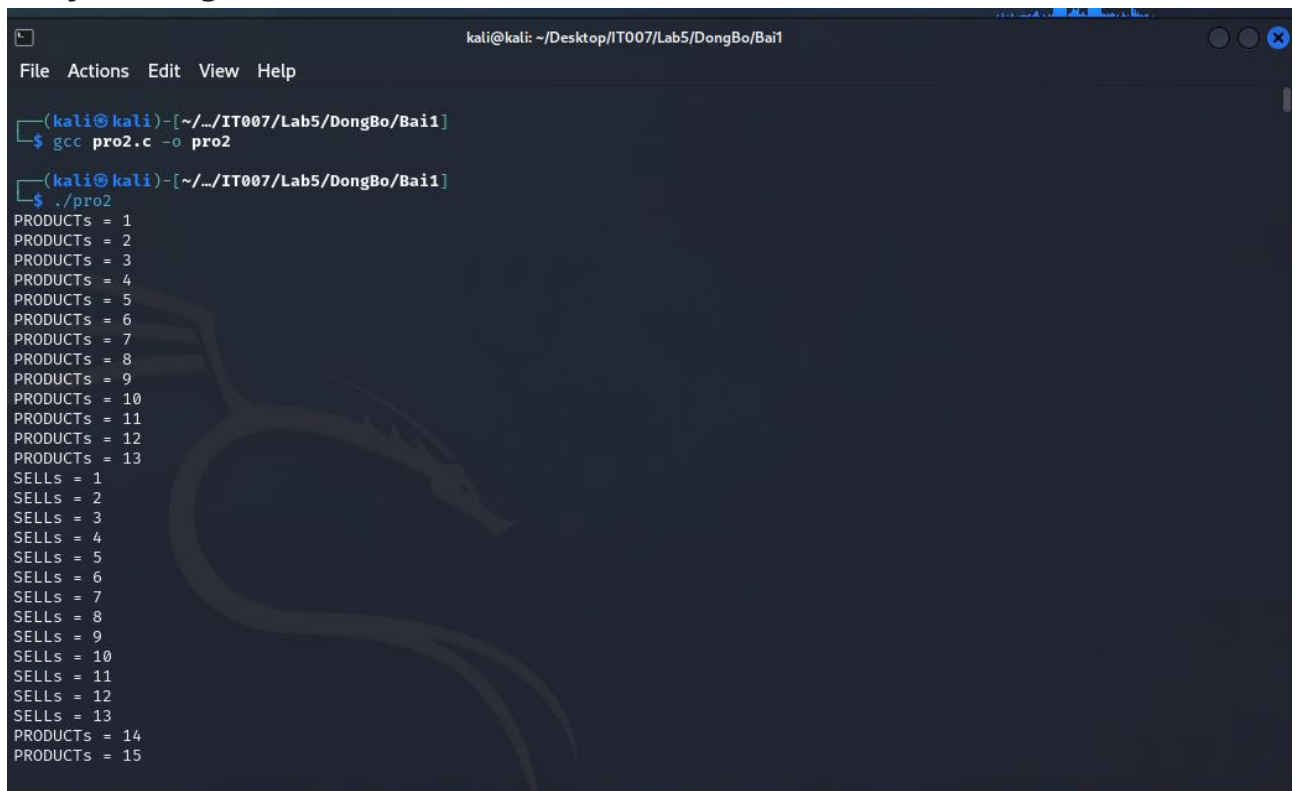
void *processB(void *mess){
    while (1){
        sem_wait(&sem2);
        products++;
        printf("PRODUCTs = %d\n", products);
        sem_post(&sem);
    }
}

int main(){
    sem_init(&sem, 0, 0); // sem = 0
    sem_init(&sem2, 0, 13); // sem = 13
    pthread_t pA, pB;
    pthread_create(&pA, NULL, &processA, NULL);
    pthread_create(&pB, NULL, &processB, NULL);

    while(1){}
    return 0;
}
```

Giải thích: sem đóng vai trò là điều kiện ($\text{sells} \leq \text{products}$), sem2 đóng vai trò là điều kiện ($\text{products} \leq \text{sells} + 13$). Khi chương trình được nạp, giả sử ProcessA được nạp trước. Khi đó A sẽ bị khóa lại với bởi biến sem (khởi tạo = 0). Vì vậy ProcessB sẽ được chạy. Products và sem (bởi hàm `sem_post`) sẽ được tăng lên cho đến khi $\text{sem2}=0$ (được giảm bởi hàm `sem_wait`). Khi đó ProcessA sẽ được chạy. Sells và sem2 (bởi hàm `sem_post`) sẽ tăng cho đến khi $\text{sem}=0$ (được giảm bởi hàm `sem_wait`). Hai tiến trình này sẽ được lặp đi lặp lại trong 1 vòng lặp vô hạn.

Chạy chương trình:



```
kali@kali: ~/Desktop/IT007/Lab5/DongBo/Bai1
File Actions Edit View Help

(kali@kali)~[~/IT007/Lab5/DongBo/Bai1]
$ gcc pro2.c -o pro2

(kali@kali)~[~/IT007/Lab5/DongBo/Bai1]
$ ./pro2
PRODUCTs = 1
PRODUCTs = 2
PRODUCTs = 3
PRODUCTs = 4
PRODUCTs = 5
PRODUCTs = 6
PRODUCTs = 7
PRODUCTs = 8
PRODUCTs = 9
PRODUCTs = 10
PRODUCTs = 11
PRODUCTs = 12
PRODUCTs = 13
SELLs = 1
SELLs = 2
SELLs = 3
SELLs = 4
SELLs = 5
SELLs = 6
SELLs = 7
SELLs = 8
SELLs = 9
SELLs = 10
SELLs = 11
SELLs = 12
SELLs = 13
PRODUCTs = 14
PRODUCTs = 15
```

Nhận thấy, ban đầu product được tăng đến 13 rồi mới chuyển qua tiến trình SELL chạy tiếp.

Bài 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.

Thực hiện:

1/ Khi chưa tiến hành đồng bộ:

Source code:

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

int n = 0; // Số phần tử trong mảng a
int *a; // Mảng các số nguyên

void *producer(void *arg) {
    while (1) {
        int random_number = rand() % 100;
        a[n++] = random_number;
        printf("Added %d to the array. Array size: %d\n", random_number, n);
    }
    return NULL;
}

void *consumer(void *arg) {
    while (1) {
        if (n > 0) {
            int random_index = rand() % n;
            int removed_element = a[random_index];
            n--;
            a[random_index] = a[n]; // Chuyển phần tử cuối cùng về vị trí đã
xóa
            printf("Removed %d from the array. Array size: %d\n",
removed_element, n);
        } else {
            printf("Nothing in array a\n");
        }
    }
    return NULL;
}

int main() {
    srand(time(NULL)); // Khởi tạo seed cho hàm rand()
    printf("Nhap so phan tu \n");
    scanf("%d", &n);
```

```
a = (int *)malloc(sizeof(int) * n); // Khởi tạo mảng a với kích thước
1000

pthread_t producer_thread, consumer_thread;
pthread_create(&producer_thread, NULL, producer, NULL);
pthread_create(&consumer_thread, NULL, consumer, NULL);

while(1){}
return 0;
}
```

Lỗi:

Chạy chương trình khi chưa tiến hành đồng bộ có thể gây ra lỗi chưa có sự đồng bộ giữa biến chia sẻ là `n` (`array_size`) dẫn đến mỗi tiến trình khi chạy lại dùng 1 giá trị `n` khác nhau.

Ở tiến trình sinh phần tử thêm vào mảng, `array_size = 10` thì bên tiến trình lấy phần tử lại có `array_size = 1` (chưa đồng bộ)

```
kali@kali: ~/Desktop/IT007/Lab5/DongBo/Bai2
File Actions Edit View Help
(kali@kali)~[~/IT007/Lab5/DongBo/Bai2]
$ gcc bai2.c -o bai2
(kali@kali)~[~/IT007/Lab5/DongBo/Bai2]
$ ./bai2
Nhap so phan tu
1
Added 50 to the array. Array size: 2
Added 45 to the array. Array size: 2
Added 5 to the array. Array size: 3
Added 27 to the array. Array size: 4 [ ] % n;
Added 3 to the array. Array size: 5 a[random_index];
Added 51 to the array. Array size: 6
Added 64 to the array. Array size: 7
Added 50 to the array. Array size: 8 // chuyen phan tu cuoi cung va vi tri ds cua
Added 64 to the array. Array size: 9 the array, array size: 8, removed element, n);
Added 20 to the array. Array size: 10
Removed 0 from the array. Array size: 1
Removed 51 from the array. Array size: 10
Removed 7 from the array. Array size: 9
Removed 27 from the array. Array size: 8
Removed 50 from the array. Array size: 7
Removed 20 from the array. Array size: 6
Removed 3 from the array. Array size: 5
Removed 50 from the array. Array size: 4
Removed 64 from the array. Array size: 3
Removed 5 from the array. Array size: 2
Removed 45 from the array. Array size: 1
Removed 64 from the array. Array size: 0 // cho ban random
Nothing in array a
Nothing in array a
Nothing in array a
Nothing in array a // (rand() % n); // cho tao mang a voi kich thuoc 1000
Nothing in array a
pthread_create(&producer_thread, &consumer_thread);
pthread_create(&producer_thread, NULL, producer, NULL);
```

Cách giải quyết:

Tiến hành đồng bộ chương trình bằng semaphore: sử dụng 3 biến semaphore full, empty và mutex với source code như sau:

```

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

int n = 0;
int *a;

sem_t full, empty, mutex;

void *producer(void *arg) {
    while (1) {
        int random_number = rand() % 100;
        sem_wait(&empty);
        sem_wait(&mutex);
        a[n++] = random_number;
        printf("Added %d to the array. Array size: %d\n", random_number, n);
        sem_post(&mutex);
        sem_post(&full);
    }
    return NULL;
}

void *consumer(void *arg) {
    while (1) {
        sem_wait(&full);
        sem_wait(&mutex);
        if (n > 0) {
            int random_index = rand() % n;
            int removed_element = a[random_index];
            n--;
            a[random_index] = a[n];
            printf("Removed %d from the array. Array size: %d\n",
removed_element, n);
        } else {
            printf("Nothing in array a\n");
        }
        sem_post(&mutex);
        sem_post(&empty);
    }
    return NULL;
}

int main() {
    srand(time(NULL));
    printf("Nhap so phan tu: ");
    scanf("%d", &n);
    a = (int *)malloc(sizeof(int) * n);

```

```

sem_init(&full, 0, 0);
sem_init(&empty, 0, n);
sem_init(&mutex, 0, 1);

pthread_t producer_thread, consumer_thread;
pthread_create(&producer_thread, NULL, producer, NULL);
pthread_create(&consumer_thread, NULL, consumer, NULL);

while (1) {}
return 0;
}

```

- Khởi tạo các biến semaphore với giá trị ban đầu là: full = 0, empty = n và mutex = 1 (thực chất là khoá mutex)
- Ở tiến trình sinh phần tử, gọi hàm wait(empty) và tiến hành "lock" khoá mutex lại để không có tiến trình nào vào vùng tranh chấp. Tiến trình sinh phần tử sẽ thực hiện cho đến khi nào biến semaphore empty nhận giá trị âm và nó sẽ bị block trong hàng đợi. Sau đó tiến hành gọi hàm post(full) để giải phóng tiến trình xoá phần tử bị block ở hàng đợi.
- Ở tiến trình xoá phần tử, ban đầu khi gọi nó, hàm wait(full) sẽ làm cho biến full có giá trị âm và tiến trình này bị block cho đến khi hàm post(full) ở tiến trình sinh được gọi sau đó nó tiến hành gọi post(empty) để giải phóng tiến trình sinh khi cần thiết.

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

int x = 0;	
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.

Hiện thực:

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

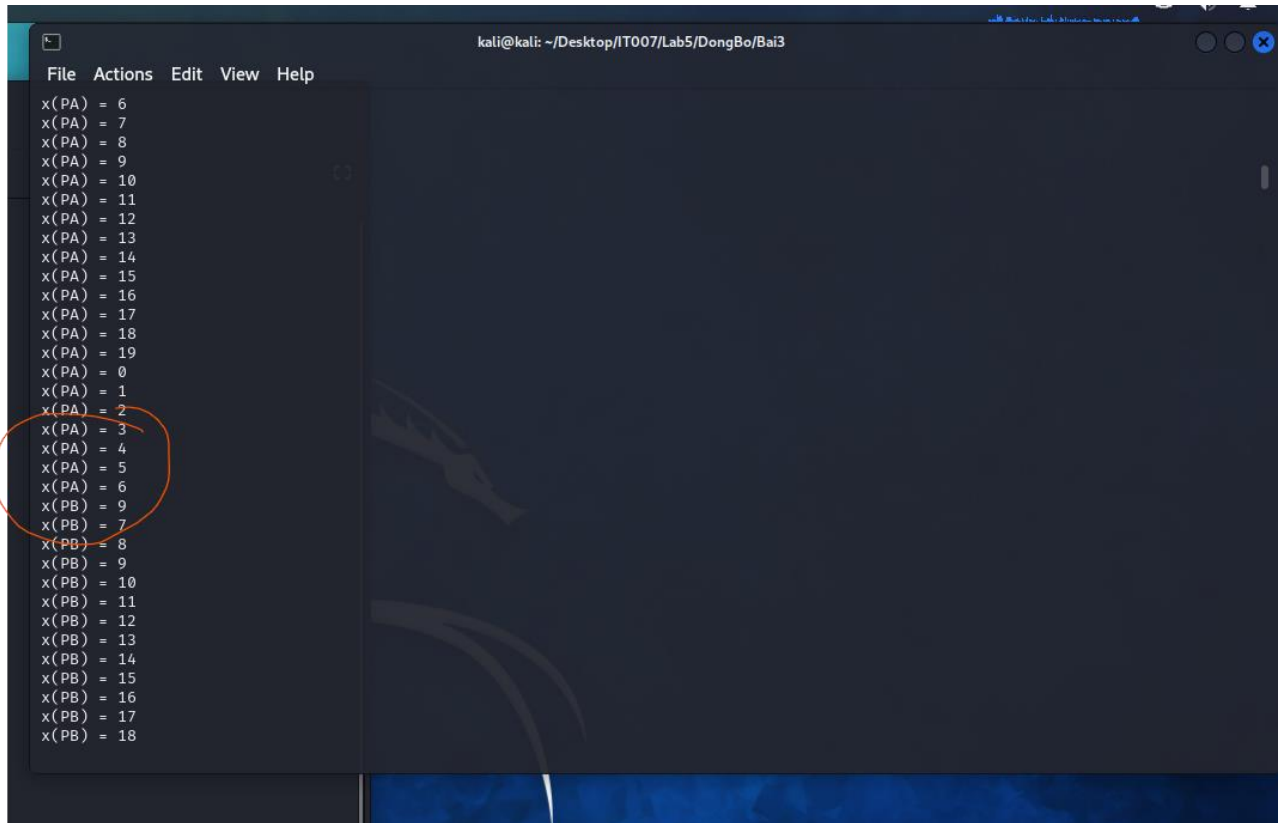
int x = 0;

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

void *processB(void *arg) {
    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;
}
```

Kết quả thực hiện:



```
kali@kali: ~/Desktop/IT007/Lab5/DongBo/Bai3
File Actions Edit View Help
x(PA) = 6
x(PA) = 7
x(PA) = 8
x(PA) = 9
x(PA) = 10
x(PA) = 11
x(PA) = 12
x(PA) = 13
x(PA) = 14
x(PA) = 15
x(PA) = 16
x(PA) = 17
x(PA) = 18
x(PA) = 0
x(PA) = 1
x(PA) = 2
x(PA) = 3
x(PA) = 4
x(PA) = 5
x(PA) = 6
x(PB) = 9
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(PB) = 18
```

Nhận xét: xuất hiện lỗi do chưa có sự đồng bộ về dữ liệu. Nếu đúng thì chỗ từ $x(PA) = 6$ --> qua tiến trình B phải là $x(PB) = 7$ nhưng lại xuất hiện $x(PB) = 9$ (chưa đồng bộ)

Giải pháp: sử dụng khoá mutex.

Bài 4:

Source code:

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

int x = 0;
pthread_mutex_t mutex;

void *processA(void *arg){
    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 *arg) {  
    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_mutex_init(&mutex, NULL);  
    pthread_t pA, pB;  
    pthread_create(&pA, NULL, &processA, NULL);  
    pthread_create(&pB, NULL, &processB, NULL);  
    while(1) {}  
    pthread_mutex_destroy(&mutex);  
    return 0;  
}
```

Kết quả: dữ liệu biến chia sẻ x được đồng bộ với nhau và trả về kết quả chính xác.

```

kali@kali: ~/Desktop/IT007/La
File Actions Edit View Help
x(PB) = 15
x(PB) = 16
x(PB) = 17
x(PB) = 18
x(PB) = 19
x(PB) = 0
x(PB) = 1
x(PB) = 2
x(PB) = 3
x(PB) = 4
x(PB) = 5
x(PB) = 6
x(PB) = 7
x(PB) = 8
x(PB) = 9
x(PB) = 10
x(PA) = 11
x(PA) = 12
x(PA) = 13
x(PA) = 14
x(PA) = 15
x(PA) = 16
x(PA) = 17
x(PA) = 18
x(PA) = 19
x(PA) = 0
x(PA) = 1
x(PA) = 2
x(PA) = 3
x(PA) = 4
x(PA) = 5
x(PA) = 6
x(PA) = 7
x(PA) = 8

```

LUYỆN TẬP:

Bài 1:

1. 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.

Giải:

- Dựa vào yêu cầu đề bài, (c) (d) chỉ tính sau khi có v đồng nghĩa với việc (b) thực hiện trước.
 - (e) tính sau khi w và y được tính mà w được tính ở (a)
- => (a) và (b) được thực hiện trước, lần lượt (c) (d) (e) (g) được chạy. Và c tính trước e và d trước f sau đó mới tính ans.

Source code:

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

int x1, x2, x3, x4, x5, x6;
int w, v, y, z, ans;

sem_t sem_v, sem_v2, sem_wy, sem_yz, sem_ans1, sem_ans2;

void* processA(void* arg) {
    w = x1 * x2;
    printf("w = %d\n", w);
    return NULL;
}

void* processB(void* arg) {
    v = x3 * x4;
    printf("v = %d\n", v);
    sem_post(&sem_v);
    sem_post(&sem_v2);
    return NULL;
}

void* processC(void* arg) {
    sem_wait(&sem_v);
    y = v * x5;
    printf("y = %d\n", y);
    sem_post(&sem_wy);
    return NULL;
}

void* processD(void* arg) {
    sem_wait(&sem_v2);
    z = v * x6;
    printf("z = %d\n", z);
    sem_post(&sem_yz);
    return NULL;
}
```

```

}

void* processE(void* arg) {
    sem_wait(&sem_wy);
    y = w * y;
    printf("y = %d\n", y);
    sem_post(&sem_ans2);
    return NULL;
}

void* processF(void* arg) {
    sem_wait(&sem_yz);
    z = w * z;
    printf("z = %d\n", z);
    sem_post(&sem_ans1);
    return NULL;
}

void* processAns(void* arg) {
    sem_wait(&sem_ans1);
    sem_wait(&sem_ans2);
    ans = y + z;
    printf("ans = %d\n", ans);
    return NULL;
}

int main() {
    pthread_t t1, t2, t3, t4, t5, t6, t7;

    // Khởi tạo các semaphore
    sem_init(&sem_v, 0, 0);
    sem_init(&sem_v2, 0, 0);
    sem_init(&sem_wy, 0, 0);
    sem_init(&sem_yz, 0, 0);
    sem_init(&sem_ans1, 0, 0);
    sem_init(&sem_ans2, 0, 0);

    // Nhập các giá trị cho x1, x2, x3, x4, x5, x6
    printf("Enter the values of x1, x2, x3, x4, x5, x6:\n");
    scanf("%d %d %d %d %d %d", &x1, &x2, &x3, &x4, &x5, &x6);

    // Tạo các thread
    pthread_create(&t1, NULL, processA, NULL);
    pthread_create(&t2, NULL, processB, NULL);
    pthread_create(&t3, NULL, processC, NULL);
    pthread_create(&t4, NULL, processD, NULL);
    pthread_create(&t5, NULL, processE, NULL);
    pthread_create(&t6, NULL, processF, NULL);
    pthread_create(&t7, NULL, processAns, NULL);
}

```

```
while (1){  
  
    return 0;  
}
```

Ví dụ:

Kết quả thực hiện với x1 – x6 lần lượt là 1 2 3 4 5 6

Tính toán bằng tay:

$$w = 1 * 2 = 2$$

$$v = 3 * 4 = 12$$

$$y = 12 * 5 = 60$$

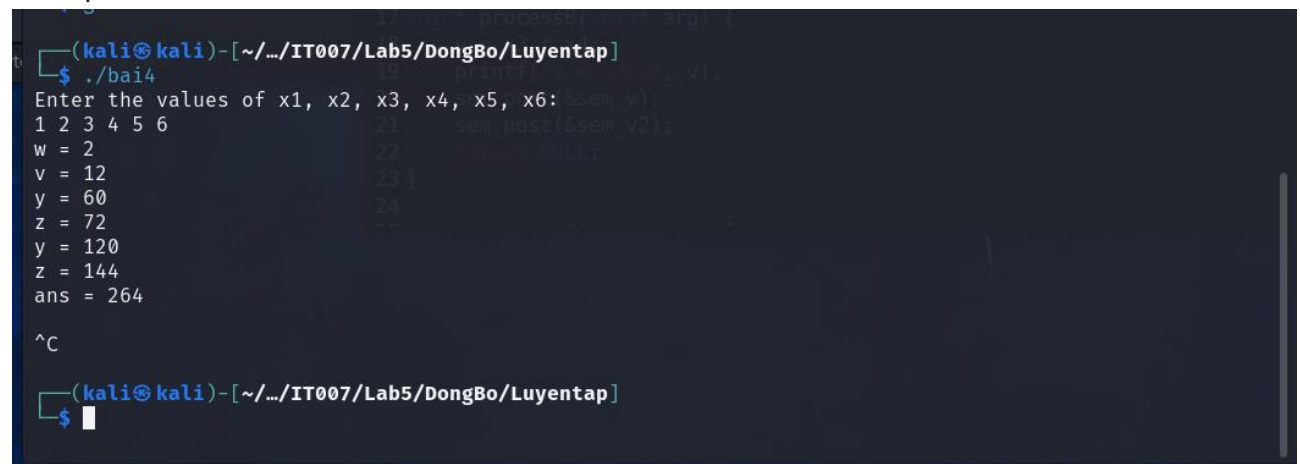
$$z = 12 * 6 = 72$$

$$y = 2 * 60 = 120 \text{ (c tính trước e)}$$

$$z = 2 * 72 = 144 \text{ (d trước f)}$$

$$\text{ans} = 120 + 144 = 264$$

Kết quả thực hiện:



```
(kali㉿kali)-[~/.../IT007/Lab5/DongBo/Luyentap]  
$ ./bai4  
Enter the values of x1, x2, x3, x4, x5, x6:  
1 2 3 4 5 6  
w = 2  
v = 12  
y = 60  
z = 72  
y = 120  
z = 144  
ans = 264  
^C  
(kali㉿kali)-[~/.../IT007/Lab5/DongBo/Luyentap]  
$
```