Laboratory Assignments

Subject: Design Principles of Operating Systems

Subject code: CSE 3249

Assignment 5: Implementation of synchronization using semaphore:

Objective of this Assignment:

- To implement the concept of multi-threading in a process.
- To learn the use of semaphore i.e., to control access to shared resources.

1. Producer-Consumer problem

Problem: Write a C program to implement the producer-consumer program where:

- Producer generates integers from 1 to 50.
- Consumer processes the numbers.

Requirements:

- Use a shared buffer with a maximum size of 10.
- Use semaphores and mutex to ensure thread-safe access to the buffer.
- Print the number that producer is producing and consumer is consuming.
- Both producer and consumer will continue for 20 iterations

Ubuntu

```
swayam@Swayam:Lab5$ cat > q1.c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

int buffer[10], in = 0, out = 0;
sem_t empty, full;
pthread_mutex_t mutex;

void* producer(void* arg)
{
    for (int i = 1; i <= 20; i++) {
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = i;
        printf("Produced: %d\n", i);
        in = (in + 1) % 10;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
    return NULL;
}

void* consumer(void* arg)
{
    for (int i = 1; i <= 20; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        printf("Consumed: %d\n", item);
        out = (out + 1) % 10;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }
    return NULL;
}

int main()
{
    pthread_t p, c;
    sem_init(&empty, 0, 10);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&p, NULL, producer, NULL);
    pthread_create(&c, NULL, consumer, NULL);

    pthread_join(p, NULL);
    pthread_join(c, NULL);
    return 0;
}
^C
```

```
swayam@Swayam:Lab5$ gcc q1.c -o q1
swayam@Swayam:Lab5$ ./q1
Produced: 1
Produced: 2
Produced: 3
Produced: 4
Produced: 5
Produced: 6
Produced: 7
Produced: 8
Produced: 9
Produced: 10
Consumed: 1
Consumed: 2
Consumed: 3
Consumed: 4
Consumed: 5
Consumed: 6
Consumed: 7
Consumed: 8
Consumed: 9
Consumed: 10
Produced: 11
Produced: 12
Produced: 13
Consumed: 11
Consumed: 12
Consumed: 13
Produced: 14
Produced: 15
Produced: 16
Produced: 17
Produced: 18
Produced: 19
Produced: 20
Consumed: 14
Consumed: 15
Consumed: 16
Consumed: 17
Consumed: 18
Consumed: 19
Consumed: 20
swayam@Swayam:Lab5$
```

2. Alternating Numbers with Two Threads

Problem: Write a program to print 1, 2, 3 … upto 20. Create threads where two threads print numbers alternately.

- **Thread A** prints odd numbers: 1, 3, 5 ...
- **Thread B** prints even numbers: 2, 4, 6 ...

Requirements:

- Use semaphores to control the order of execution of the threads.
- Ensure no race conditions occur.

Ubuntu

```
swayam@Swayam:Lab5$ cat > q2.c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

sem_t odd, even;

void* printOdd(void* arg)
{
    for (int i = 1; i <= 19; i += 2) {
        sem_wait(&odd);
        printf("%d\n", i);
        sem_post(&even);
    }
    return NULL;
}

void* printEven(void* arg)
{
    for (int i = 2; i <= 20; i += 2) {
        sem_wait(&even);
        printf("%d\n", i);
        sem_post(&odd);
    }
    return NULL;
}

int main()
{
    pthread_t t1, t2;
    sem_init(&odd, 0, 1);
    sem_init(&even, 0, 0);

    pthread_create(&t1, NULL, printOdd, NULL);
    pthread_create(&t2, NULL, printEven, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    return 0;
}
^C
swayam@Swayam:Lab5$ gcc q2.c -o q2
swayam@Swayam:Lab5$ ./q2
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
swayam@Swayam:Lab5$
```

3. Alternating Characters

Problem: Write a program to create two threads that print characters (A and B) alternately such as ABABABABA…. upto 20. Use semaphores to synchronize the threads.

- **Thread A** prints A.
- **Thread B** prints B.

Requirements:

- Use semaphores to control the order of execution of the threads.
- Ensure no race conditions occur.

Ubuntu

```
swayam@Swayam:Lab5$ cat q3.c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

sem_t sa, sb;

void* printA(void* arg)
{
    for (int i = 0; i < 10; i++) {
        sem_wait(&sa);
        printf("A");
        sem_post(&sb);
    }
    return NULL;
}

void* printB(void* arg)
{
    for (int i = 0; i < 10; i++) {
        sem_wait(&sb);
        printf("B");
        sem_post(&sa);
    }
    return NULL;
}

int main()
{
    pthread_t t1, t2;
    sem_init(&sa, 0, 1);
    sem_init(&sb, 0, 0);

    pthread_create(&t1, NULL, printA, NULL);
    pthread_create(&t2, NULL, printB, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    return 0;
}
swayam@Swayam:Lab5$ gcc q3.c -o q3
swayam@Swayam:Lab5$ ./q3
swayam@Swayam:Lab5$ ./q3
ABABABABABABABABABswayam@Swayam:Lab5$
```

4. Countdown and Countup

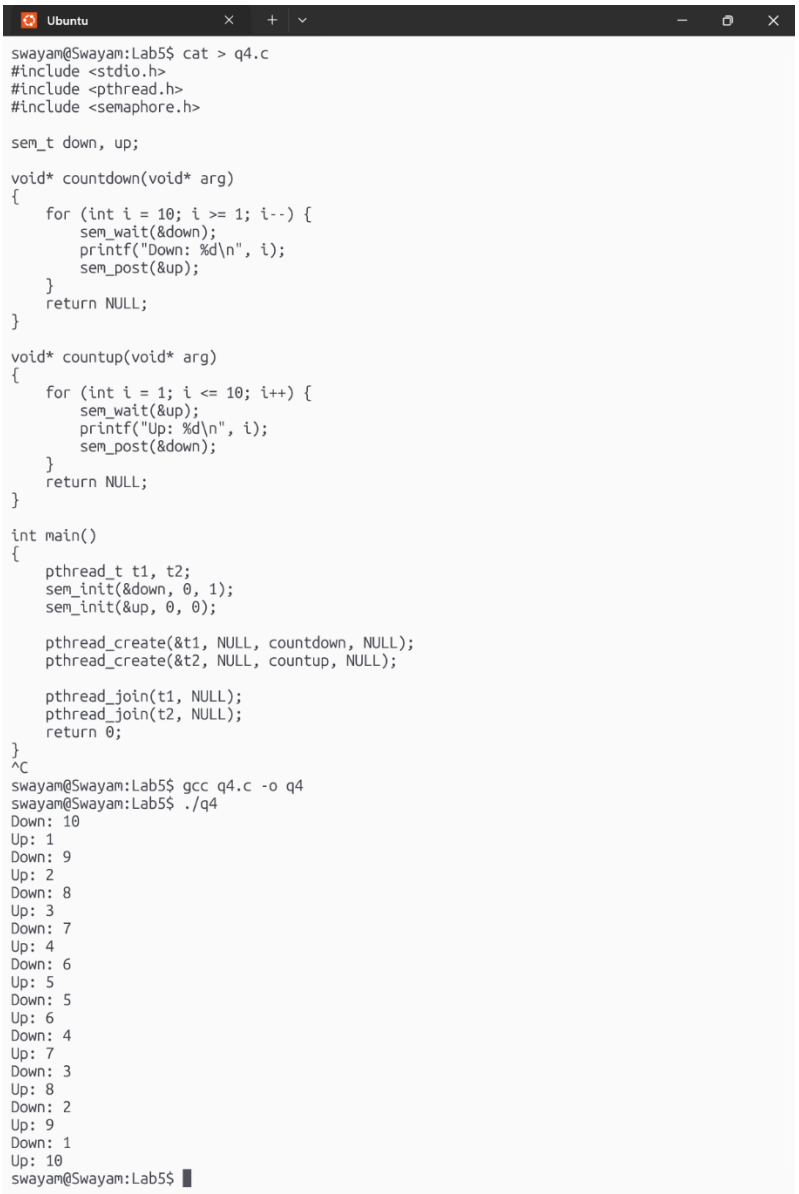
Problem: Write a program create two threads where:

- **Thread A** counts down from 10 to 1.
- **Thread B** counts up from 1 to 10.

Both threads should alternate execution.

Requirements:

- Use semaphores to control the order of execution of the threads.
- Ensure no race conditions occur.



5. Sequence Printing using Threads

Problem: Write a program that creates three threads: Thread A, Thread B, and Thread C. The threads must print numbers in the following sequence: A1, B2, C3, A4, B5, C6 … upto 20 numbers.

- **Thread A** prints A1, A4, A7, …
- **Thread B** prints B2, B5, B8, …
- **Thread C** prints C3, C6, C9, ...

Requirements:

- Use semaphores to control the order of execution of the threads.
- Ensure no race conditions occur

Ubuntu

```
swayam@Swayam:Lab5$ cat > q5.c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

sem_t sa, sb, sc;
int num = 1;

void* printA(void* arg)
{
    while (num <= 20) {
        sem_wait(&sa);
        if (num <= 20) printf("A%d\n", num++);
        sem_post(&sb);
    }
    return NULL;
}

void* printB(void* arg)
{
    while (num <= 20) {
        sem_wait(&sb);
        if (num <= 20) printf("B%d\n", num++);
        sem_post(&sc);
    }
    return NULL;
}

void* printC(void* arg)
{
    while (num <= 20) {
        sem_wait(&sc);
        if (num <= 20) printf("C%d\n", num++);
        sem_post(&sa);
    }
    return NULL;
}

int main()
{
    pthread_t t1, t2, t3;
    sem_init(&sa, 0, 1);
    sem_init(&sb, 0, 0);
    sem_init(&sc, 0, 0);

    pthread_create(&t1, NULL, printA, NULL);
    pthread_create(&t2, NULL, printB, NULL);
    pthread_create(&t3, NULL, printC, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);
    return 0;
}
^C
swayam@Swayam:Lab5$ gcc q5.c -o q5
swayam@Swayam:Lab5$ ./q5
A1
B2
C3
A4
B5
C6
A7
B8
C9
A10
B11
C12
A13
B14
C15
A16
B17
C18
A19
B20
swayam@Swayam:Lab5$
```