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 100.
- 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

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>


#define BUFFER_SIZE 10


int buffer[BUFFER_SIZE];

int count = 0;

sem_t empty, full, mutex;
```

```
void *producer(void *param) {
    int item;
    for (int i = 0; i < 20; i++) {
        item = rand() % 100; // Produce an item
        printf("Producer: waiting on empty...\n");
        sem_wait(&empty);
        printf("Producer: acquired mutex...\n");
        sem_wait(&mutex);
        buffer[count++] = item; // Add item to the buffer
        printf("Producer produced %d\n", item);
        sem_post(&mutex);
        sem_post(&full);
    }
    pthread_exit(NULL);
}

void *consumer(void *param)
{
    int item;
    for (int i = 0; i < 20; i++)
    {
        printf("Consumer: waiting on full...\n");
        sem_wait(&full);
        printf("Consumer: acquired mutex...\n");
        sem_wait(&mutex);
        item = buffer[--count]; // Remove item from the buffer
        printf("Consumer consumed %d\n", item);
        sem_post(&mutex);
        sem_post(&empty);
    }
    pthread_exit(NULL);
}
int main()
{
    pthread_t prod, cons;
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    sem_init(&mutex, 0, 1);
    // Disable buffering for immediate output
    setvbuf(stdout, NULL, _IONBF, 0);
    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);
    pthread_join(prod, NULL);
    pthread_join(cons, NULL);
    sem_destroy(&empty);
    sem_destroy(&full);
    sem_destroy(&mutex);
    return 0;
}
```

OUTPUT:

```
Producer: waiting on empty...
Producer: acquired mutex...
Consumer: waiting on full...
Producer produced 83
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 86
Producer: waiting on empty...
Producer: acquired mutex...
Consumer: acquired mutex...
```

```
Producer produced 77
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 15
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 93
Producer: waiting on empty...
Producer: acquired mutex...
Consumer consumed 93
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 15
Consumer: waiting on full...
Consumer: acquired mutex...
Producer produced 35
Producer: waiting on empty...
Producer: acquired mutex...
Consumer consumed 35
Consumer: waiting on full...
Consumer: acquired mutex...
Producer produced 86
Producer: waiting on empty...
Producer: acquired mutex...
Consumer consumed 86
Consumer: waiting on full...
Consumer: acquired mutex...
Producer produced 92
Producer: waiting on empty...
Producer: acquired mutex...
```

```
Consumer consumed 92
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 77
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 86
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 83
Consumer: waiting on full...
Producer produced 49
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 21
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 62
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 27
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 90
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 59
Producer: waiting on empty...
Producer: acquired mutex...
Consumer: acquired mutex...
```

```
Producer produced 63
Producer: waiting on empty...
Producer: acquired mutex...
Consumer consumed 63
Consumer: waiting on full...
Consumer: acquired mutex...
Producer produced 26
Producer: waiting on empty...
Producer: acquired mutex...
Consumer consumed 26
Consumer: waiting on full...
Consumer: acquired mutex...
Producer produced 40
Producer: waiting on empty...
Producer: acquired mutex...
Consumer consumed 40
Consumer: waiting on full...
Consumer: acquired mutex...
Producer produced 26
Producer: waiting on empty...
Producer: acquired mutex...
Consumer consumed 26
Consumer: waiting on full...
Consumer: acquired mutex...
Producer produced 72
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 36
Consumer consumed 36
Consumer: waiting on full...
```

```
Consumer: acquired mutex...
Consumer consumed 72
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 59
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 90
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 27
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 62
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 21
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 49
```

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.

CODE:

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
int current_number = 1;
sem_t semA, semB;
void* print_odd(void* arg)
{
    while (current_number <= 20)
    {
        sem_wait(&semA);
        if (current_number % 2 != 0)
        {
            printf("%d\n", current_number);
            current_number++;
        }
        sem_post(&semB);
    }
    return NULL;
}
void* print_even(void* arg)
{
    while (current_number <= 20)
    {
        sem_wait(&semB);
        if (current_number % 2 == 0)
        {
            printf("%d\n", current_number);
            current_number++;
        }
        sem_post(&semA);
    }
    return NULL;
}
int main()
{
    sem_init(&semA, 0, 1);
    sem_init(&semB, 0, 0);
    pthread_t threadA, threadB;
    pthread_create(&threadA, NULL, print_odd, NULL);
    pthread_create(&threadB, NULL, print_even, NULL);
    pthread_join(threadA, NULL);
    pthread_join(threadB, NULL);
    sem_destroy(&semA);
    sem_destroy(&semB);
    return 0;
}
```

OUTPUT:

```
1
2
3
4
5
6
7
8
9
10
11
12
```

```
13

14

15

16

17

18

19

20
```

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.

CODE:

```
#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

sem_t semA, semB;

void* printA(void* arg)

{

   for (int i = 0; i < 10; i++)

   {

      sem_wait(&semA);

      printf("A");
```

```
        fflush(stdout);
        sem_post(&semB);
    }
    return NULL;
}
void* printB(void* arg)
{
    for (int i = 0; i < 10; i++)
    {
        sem_wait(&semB);
        printf("B");
        fflush(stdout);
        sem_post(&semA);
    }
    return NULL;
}
int main()
{
    sem_init(&semA, 0, 1);
    sem_init(&semB, 0, 0);
    pthread_t threadA, threadB;
    pthread_create(&threadA, NULL, printA, NULL);
    pthread_create(&threadB, NULL, printB, NULL);
    pthread_join(threadA, NULL);
    pthread_join(threadB, NULL);
    sem_destroy(&semA);
    sem_destroy(&semB);
    printf("\n");
    return 0;
}
```

OUTPUT:

```
ABABABABABABABABABAB
```

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.

CODE:

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t semA, semB;
void* countdown(void* arg)
{
   for (int i = 10; i >= 1; i--)
   {
      sem_wait(&semA);
      printf("Thread A: %d\n", i);
      sleep(1);  // Simulate work
      sem_post(&semB);
   }
   return NULL;
}
void* countup(void* arg)
{
   for (int i = 1; i <= 10; i++)
```

```
    {
        sem_wait(&semB);
        printf("Thread B: %d\n", i);
        sleep(1);
        sem_post(&semA);
    }
    return NULL;
}
int main()
{
    sem_init(&semA, 0, 1);
    sem_init(&semB, 0, 0);
    pthread_t threadA, threadB;
    pthread_create(&threadA, NULL, countdown, NULL);
    pthread_create(&threadB, NULL, countup, NULL);
    pthread_join(threadA, NULL);
    pthread_join(threadB, NULL);
    sem_destroy(&semA);
    sem_destroy(&semB);
    printf("Both threads have finished.\n");
    return 0;
}
```

OUTPUT:

```
Thread A: 10
Thread B: 1
Thread A: 9
Thread B: 2
Thread A: 8
Thread B: 3
...
```

```
Thread A: 1

Thread B: 10

Both threads have finished.
```

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.

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define NUM_COUNT 20

sem_t semA, semB, semC;

void* print_A(void* param)

{

    for (int i = 1; i <= NUM_COUNT; i += 3)

    {

        sem_wait(&semA);

        printf("A%d\n", i);

        sem_post(&semB);

    }

    pthread_exit(NULL);
```

```
}
void* print_B(void* param)
{
    for (int i = 2; i <= NUM_COUNT; i += 3)
    {
        sem_wait(&semB);
        printf("B%d\n", i);
        sem_post(&semC);
    }
    pthread_exit(NULL);
}
void* print_C(void* param)
{
    for (int i = 3; i <= NUM_COUNT; i += 3)
    {
        sem_wait(&semC);
        printf("C%d\n", i);
        sem_post(&semA);
    }
    pthread_exit(NULL);
}
int main()
{
    pthread_t threadA, threadB, threadC;
    sem_init(&semA, 0, 1);
    sem_init(&semB, 0, 0);
    sem_init(&semC, 0, 0);
    pthread_create(&threadA, NULL, print_A, NULL);
    pthread_create(&threadB, NULL, print_B, NULL);
    pthread_create(&threadC, NULL, print_C, NULL);
    pthread_join(threadA, NULL);
    pthread_join(threadB, NULL);
    pthread_join(threadC, NULL);
    sem_destroy(&semA);
    sem_destroy(&semB);
    sem_destroy(&semC);
    return 0;
}
```

OUTPUT:

```
A1
B2
C3
A4
B5
C6
A7
B8
C9
A10
B11
C12
A13
B14
C15
A16
B17
C18
A19
B20
```