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

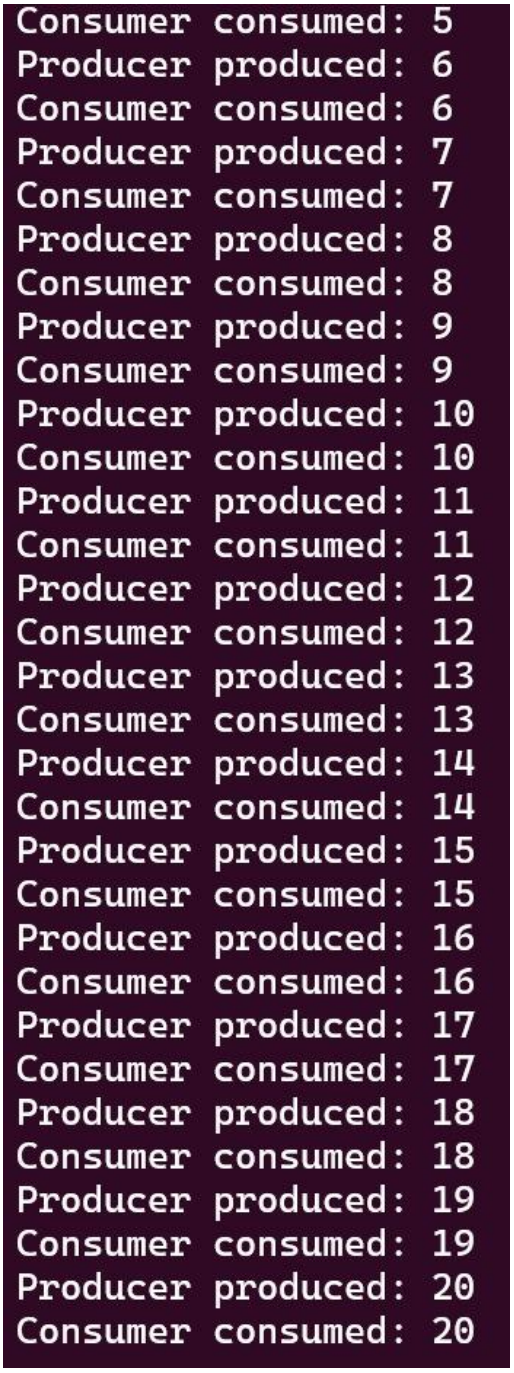
Problem: Write a C program to implement the producer-consumer program where**:**

- 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

```
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117$ mkdir DOSass5
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117$ cd DOSass5
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ cat > a5q1.c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define BUFFER_SIZE 10
int buffer[BUFFER_SIZE];
int in = 0 , out = 0 ;
sem_t empty;
sem_t 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("Producer produced: %d\n", i);
          in = (in +1) % BUFFER_SIZE;
          pthread_mutex_unlock(&mutex);
          sem_post(&full);
          sleep(1);
      }
      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("Consumer consumed: %d\n", item);
        out = (out + 1)% BUFFER_SIZE;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        sleep(1);
    }
    return NULL;
}
int main(){
    pthread_t prod, cons;
    sem_init(&empty , 0 , BUFFER_SIZE);
    sem_init(&full , 0 ,0);
    pthread_mutex_init(&mutex, NULL);
    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);
    pthread_mutex_destroy(&mutex);
    return 0;
}
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ gcc a5q1.c -o a5q1 -lpthread
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ ./a5q1
Producer produced: 1
Consumer consumed: 1
Producer produced: 2
Consumer consumed: 2
Producer produced: 3
Consumer consumed: 3
Producer produced: 4
```



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.

```
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ cat > a5q2.c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
sem_t semA,semB;
void* printOdd(void* arg){
    for(int i = 1 ; i<= 19 ; i+=2){
        sem_wait(&semA);
        printf("%d\n", i);
        sem_post(&semB);
    }
    return NULL;
}
void* printEven(void* arg){
    for(int i = 2 ; i<= 20; i+=2){
        sem_wait(&semB);
        printf("%d\n", i);
        sem_post(&semA);
    }
    return NULL;
}
int main(){
    pthread_t t1,t2;
    sem_init(&semA, 0, 1);
    sem_init(&semB, 0, 0);
    pthread_create(&t1, NULL , printOdd, NULL);
    pthread_create(&t2, NULL , printEven , NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    sem_destroy(&semA);
    sem_destroy(&semB);
```

```
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    sem_destroy(&semA);
    sem_destroy(&semB);
    return 0;
}
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ gcc a5q2.c -o a5q2 -lpthread
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ ./a5q2
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.

```
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ cat > a5q3.c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
sem_t semA, semB;
void* printA(void* arg){
    for(int i = 0 ; i< 20 ;i++){
       sem_wait(&semA);
       printf("A");
       sem_post(&semB);
    }
    return NULL;
}
void* printB(void* arg){
    for(int i = 0 ; i< 20 ; i++){
        sem_wait(&semB);
        printf("B");
        sem_post(&semA);
    }
    return NULL;
}
int main(){
    pthread_t t1,t2;
    sem_init(&semA, 0 , 1);
    sem_init(&semB, 0 , 0);
    pthread_create(&t1, NULL, printA, NULL);
    pthread_create(&t2, NULL, printB, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
```

```
    sem_destroy(&semA);
    sem_destroy(&semB);
    return 0;
}
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ gcc a5q3.c -o a5q3 -lpthread
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ ./a5q3
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ ./a5q3
ABABABABABABABABABABABABABABABABABABABABkoushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$
```

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.

```
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ cat > a5q4.c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
sem_t semDown, semUp;
void* countDown(void* arg){
     for(int i = 10 ; i >= 1 ; i--){
         sem_wait(&semDown);
         printf("Down : %d\n" , i);
         sem_post(&semUp);
      }
      return NULL;
}
void* countUp(void* arg){
      for(int i = 1 ; i<= 10 ; i++){
          sem_wait(&semUp);
          printf("Up : %d\n" ,i);
          sem_post(&semDown);
       }
       return NULL;
}
int main() {
    pthread_t t1,t2;
    sem_init(&semDown , 0 ,1);
    sem_init(&semUp, 0 , 0);
      sem_init(&semDown, 0, 1);
    sem_init(&semUp, 0, 0);

    pthread_create(&t1, NULL, countDown, NULL);
    pthread_create(&t2, NULL, countUp, NULL);
```

```
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    sem_destroy(&semDown);
    sem_destroy(&semUp);
    return 0;
}
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ gcc a5q4.c -o a5q4 -lpthread
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ ./a5q4
Down : 10
Up : 1
Down : 9
Up : 2
Down : 8
Up : 3
Down : 7
Up : 4
Down : 6
Up : 5
Down : 5
Up : 6
Down : 4
Up : 7
Down : 3
Up : 8
Down : 2
Up : 9
Down : 1
Up : 10
```

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.

```
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ cat > a5q5.c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
sem_t semA, semB, semC;
int number = 1;
void* threadA(void* arg){
      while(number <= 20){
          sem_wait(&semA);
          if(number <= 20){
              printf("A%d\n", number++);
          }
          sem_post(&semB);
       }
       return NULL;
}
void* threadB(void* arg){
      while(number <= 20){
           sem_wait(&semB);
           if(number <= 20){
              printf("B%d\n" , number++);
           }
           sem_post(&semC);
       }
       return NULL;
}
void* threadC(void* arg){
      while(number <= 20){
```

```
void* threadC(void* arg){
      while(number <= 20){
           sem_wait(&semC);
           if(number <= 20){
              printf("C%d\n" , number++);
           }
           sem_post(&semA);
       }
       return NULL;
}
int main(){
    pthread_t t1,t2,t3;
    sem_init(&semA, 0, 1);
    sem_init(&semB, 0, 0);
    sem_init(&semC, 0, 0);

    pthread_create(&t1, NULL, threadA, NULL);
    pthread_create(&t2, NULL, threadB, NULL);
    pthread_create(&t3, NULL, threadC, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);

    sem_destroy(&semA);
    sem_destroy(&semB);
    sem_destroy(&semC);

    return 0;
}
```

```
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ gcc a5q5.c -o a5q5 -lpthread
koushik_das@LAPTOP-BIHGEI3G:~/DOS_2341004117/DOSass5$ ./a5q5
A1
B2
C3
A4
B5
C6
A7
B8
C9
A10
B11
C12
A13
B14
C15
A16
B17
C18
A19
B20
```