Laboratory Assignment 5

On

Design Principles of Operating System

(CSE 3249)

Submitted by

Name	**: Dinanath Dash**
Reg. No.	**: 2241004161**
Semester	**: 5th**
Branch	**: CSE**
Section	**: 2241026**
Session	**: 2024-2025**
Admission Batch	**: 2022**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY (ITER)

SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY

BHUBANESWAR, ODISHA – 751030

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

Output -

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define BUFFER_SIZE 10
#define ITERATIONS 20

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 <= ITERATIONS; i++) {
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = i;
        printf("Producer produced: %d\n", buffer[in]);
        in = (in + 1) % BUFFER_SIZE;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
        sleep(1); // Simulate production time
    }
    return NULL;
}
void* consumer(void* arg) {
    for (int i = 1; i <= ITERATIONS; 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); // Simulate consumption time
    }
    return NULL;
}

int main() {
    pthread_t prod_thread, cons_thread;
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);
    pthread_create(&prod_thread, NULL, producer, NULL);
    pthread_create(&cons_thread, NULL, consumer, NULL);
    pthread_join(prod_thread, NULL);
    pthread_join(cons_thread, NULL);
    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);
    return 0;
}
```

```
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ gedit Q1.c&
[2] 591
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ gcc Q1.c
[2]+  Done                    gedit Q1.c
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ ./a.out
Producer produced: 1
Consumer consumed: 1
Producer produced: 2
Consumer consumed: 2
Producer produced: 3
Consumer consumed: 3
Producer produced: 4
Consumer consumed: 4
Producer produced: 5
Consumer consumed: 5
Producer produced: 6
Consumer consumed: 6
Producer produced: 7
Consumer consumed: 7
Producer produced: 8
Consumer consumed: 8
Producer produced: 9
Consumer consumed: 9
Producer produced: 10
Consumer consumed: 10
Producer produced: 11
Consumer consumed: 11
Producer produced: 12
Consumer consumed: 12
Producer produced: 13
Consumer consumed: 13
Producer produced: 14
Consumer consumed: 14
Producer produced: 15
Consumer consumed: 15
Producer produced: 16
Consumer consumed: 16
Producer produced: 17
Consumer consumed: 17
Producer produced: 18
Consumer consumed: 18
Producer produced: 19
Consumer consumed: 19
Producer produced: 20
Consumer consumed: 20
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ gedit Q2.c&
```

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.

Output-

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define LIMIT 20
sem_t sem_odd;
sem_t sem_even;
void* print_odd(void* arg) {
    for (int i = 1; i <= LIMIT; i += 2) {
        sem_wait(&sem_odd);
        printf("Thread A (Odd): %d\n", i);
        sem_post(&sem_even);
    }
    return NULL;
}
void* print_even(void* arg) {
    for (int i = 2; i <= LIMIT; i += 2) {
        sem_wait(&sem_even);
        printf("Thread B (Even): %d\n", i);
        sem_post(&sem_odd);
    }
    return NULL;
}
int main() {
    pthread_t thread_odd, thread_even;
    sem_init(&sem_odd, 0, 1);
    sem_init(&sem_even, 0, 0);
    pthread_create(&thread_odd, NULL, print_odd, NULL);
    pthread_create(&thread_even, NULL, print_even, NULL);
    pthread_join(thread_odd, NULL);
    pthread_join(thread_even, NULL);
    sem_destroy(&sem_odd);
    sem_destroy(&sem_even);
    return 0;
}
```

```
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ gedit Q2.c&
[2] 750
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ gcc Q2.c
[2]+  Done                    gedit Q2.c
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ ./a.out
Thread A (Odd): 1
Thread B (Even): 2
Thread A (Odd): 3
Thread B (Even): 4
Thread A (Odd): 5
Thread B (Even): 6
Thread A (Odd): 7
Thread B (Even): 8
Thread A (Odd): 9
Thread B (Even): 10
Thread A (Odd): 11
Thread B (Even): 12
Thread A (Odd): 13
Thread B (Even): 14
Thread A (Odd): 15
Thread B (Even): 16
Thread A (Odd): 17
Thread B (Even): 18
Thread A (Odd): 19
Thread B (Even): 20
dinanath@DINANATH:~/DOS_2241004161/DOSass5$
```

3. Alternating Characters

Problem: Write a program to create two threads that print characters (A and B) alternately such as ABABABABA…. up to 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.

Output-

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define LIMIT 20
sem_t sem_A;
sem_t sem_B;
void* print_A(void* arg) {
    for (int i = 0; i < LIMIT; i++) {
        sem_wait(&sem_A);
        printf("A");
        fflush(stdout);
        sem_post(&sem_B);
    }
    return NULL;
}
void* print_B(void* arg) {
    for (int i = 0; i < LIMIT; i++) {
        sem_wait(&sem_B);
        printf("B");
        fflush(stdout);
        sem_post(&sem_A);
    }
    return NULL;
}
```

```c
int main() {
    pthread_t thread_A, thread_B;
    sem_init(&sem_A, 0, 1); sem_init(&sem_B, 0, 0);
    pthread_create(&thread_A, NULL, print_A, NULL);
    pthread_create(&thread_B, NULL, print_B, NULL);
    pthread_join(thread_A, NULL); pthread_join(thread_B, NULL);
    sem_destroy(&sem_A); sem_destroy(&sem_B);
    printf("\n");
    return 0;
}
```

```
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ gedit Q3.c&
[2] 770
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ gcc Q3.c
[2]+  Done                    gedit Q3.c
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ ./a.out
ABABABABABABABABABABABABABABABABABABAB
dinanath@DINANATH:~/DOS_2241004161/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.

   Output-

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define LIMIT 10
sem_t sem_countdown;
sem_t sem_countup;
void* countdown(void* arg) {
    for (int i = LIMIT; i >= 1; i--) {
        sem_wait(&sem_countdown);
        printf("Countdown: %d\n", i);
        sem_post(&sem_countup);
    }
    return NULL;
}
void* countup(void* arg) {
    for (int i = 1; i <= LIMIT; i++) {
        sem_wait(&sem_countup);
        printf("Countup: %d\n", i);
        sem_post(&sem_countdown);
    }
    return NULL;
}
int main() {
    pthread_t thread_countdown, thread_countup;
    sem_init(&sem_countdown, 0, 1);
    sem_init(&sem_countup, 0, 0);
    pthread_create(&thread_countdown, NULL, countdown, NULL);
    pthread_create(&thread_countup, NULL, countup, NULL);
    pthread_join(thread_countdown, NULL);
    pthread_join(thread_countup, NULL);
    sem_destroy(&sem_countdown);
    sem_destroy(&sem_countup);
    return 0;
}
```

```
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ gedit Q4.c&
[2] 1212
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ gcc Q4.c
[2]+  Done                    gedit Q4.c
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ ./a.out
Countdown: 10
Countup: 1
Countdown: 9
Countup: 2
Countdown: 8
Countup: 3
Countdown: 7
Countup: 4
Countdown: 6
Countup: 5
Countdown: 5
Countup: 6
Countdown: 4
Countup: 7
Countdown: 3
Countup: 8
Countdown: 2
Countup: 9
Countdown: 1
Countup: 10
dinanath@DINANATH:~/DOS_2241004161/DOSass5$
```

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.

Output-

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define LIMIT 20
sem_t sem_A, sem_B, sem_C;
void* print_A(void* arg) {
    for (int i = 1; i <= LIMIT; i += 3) {
        sem_wait(&sem_A);
        printf("A%d\n", i);
        sem_post(&sem_B);
    }
    return NULL;
}
void* print_B(void* arg) {
    for (int i = 2; i <= LIMIT; i += 3) {
        sem_wait(&sem_B);
        printf("B%d\n", i);
        sem_post(&sem_C);
    }
    return NULL;
}


void* print_C(void* arg) {
    for (int i = 3; i <= LIMIT; i += 3) {
        sem_wait(&sem_C);
        printf("C%d\n", i);
        sem_post(&sem_A);
    }
    return NULL;
}
int main() {
    pthread_t thread_A, thread_B, thread_C;
    sem_init(&sem_A, 0, 1);
    sem_init(&sem_B, 0, 0);
    sem_init(&sem_C, 0, 0);
    pthread_create(&thread_A, NULL, print_A, NULL);
    pthread_create(&thread_B, NULL, print_B, NULL);
    pthread_create(&thread_C, NULL, print_C, NULL);
    pthread_join(thread_A, NULL);
    pthread_join(thread_B, NULL);
    pthread_join(thread_C, NULL);
    sem_destroy(&sem_A);
    sem_destroy(&sem_B);
    sem_destroy(&sem_C);
    return 0;
}
```

```
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ gedit Q5.c&
[2] 1231
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ gcc Q5.c
[2]+  Done                    gedit Q5.c
dinanath@DINANATH:~/DOS_2241004161/DOSass5$ ./a.out
A1
B2
C3
A4
B5
C6
A7
B8
C9
A10
B11
C12
A13
B14
C15
A16
B17
C18
A19
B20
dinanath@DINANATH:~/DOS_2241004161/DOSass5$
```