OPERATING SYSTEM LAB

Name: Thorat Amey Arun Reg No.: 23MCS1004

LAB EXPERIMENT 7
Synchronisation Semaphore

1. Write a basic thread program to demonstrate synchronization semaphore.

Program:

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
sem_t semaphore;
void* wt(void* ti){
int id=*((int*)ti);
printf("Thread %d is trying to acquire the semaphore\n",id);
sem_wait(&semaphore);
printf("Thread %d has acquired the semaphore\n",id);
printf("Thread %d is releasing the semaphore\n",id);
sem_post(&semaphore);
pthread_exit(NULL);
}
int main(){
sem_init(&semaphore,0,2);
pthread_t threads[5];
int threads_ids[5]={0,1,2,3,4};
for(int i=0;i<5;i++){
pthread_create(&threads[i],NULL,wt,&threads_ids[i]);
}
for(int i=0;i<5;i++){
pthread_join(threads[i],NULL);
}
sem_destroy(&semaphore);
printf("All threads have finished\n");
return 0;
}
```

Output:

OPERATING SYSTEM LAB

Name: Thorat Amey Arun Reg No.: 23MCS1004

```
student1@student1-VirtualBox:~/Desktop$ gcc -o 7,1 7,1.c
student1@student1-VirtualBox:~/Desktop$ ./7,1
Thread 1 is trying to acquire the semaphore
Thread 1 has acquired the semaphore
Thread 1 is releasing the semaphore
Thread 3 is trying to acquire the semaphore
Thread 4 is trying to acquire the semaphore
Thread 0 is trying to acquire the semaphore
Thread 0 has acquired the semaphore
Thread 0 is releasing the semaphore
Thread 3 has acquired the semaphore
Thread 3 is releasing the semaphore
Thread 4 has acquired the semaphore
Thread 4 is releasing the semaphore
Thread 2 is trying to acquire the semaphore
Thread 2 has acquired the semaphore
Thread 2 is releasing the semaphore
All threads have finished
student1@student1-VirtualBox:~/Desktop$
```

2. Write a program to synchronize two threads that work on a shared variable X, where one thread increments the X, another thread decrements the value.
   Sequence should be increment followed by decrement always.
   Program:

```
#include <stdio.h>
#include <pthread.h>
int X = 0;
pthread_mutex_t mutex;
void* increment_thread(void* arg) {
   for (int i = 0; i < 5; i++) {
      pthread_mutex_lock(&mutex);
      X++;
      printf("Increment Thread: Incremented X to %d\n", X);
      pthread_mutex_unlock(&mutex);
   }
   return NULL;
}
void* decrement_thread(void* arg) {
   for (int i = 0; i < 5; i++) {
      pthread_mutex_lock(&mutex);
```

```
            X--;
            printf("Decrement Thread: Decremented X to %d\n", X);
            pthread_mutex_unlock(&mutex);
        }
        return NULL;
    }
    int main() {
        pthread_mutex_init(&mutex, NULL);
        pthread_t increment_tid, decrement_tid;
        pthread_create(&increment_tid, NULL, increment_thread, NULL);
        pthread_create(&decrement_tid, NULL, decrement_thread, NULL);
        pthread_join(increment_tid, NULL);
        pthread_join(decrement_tid, NULL);
        pthread_mutex_destroy(&mutex);
        return 0;
    }
```

Output:

```
student1@student1-VirtualBox:~/Desktop$ gcc -o 7,2 7,2.c
student1@student1-VirtualBox:~/Desktop$ ./7,2
Decrement Thread: Decremented X to -1
Decrement Thread: Decremented X to -2
Decrement Thread: Decremented X to -3
Decrement Thread: Decremented X to -4
Decrement Thread: Decremented X to -5
Increment Thread: Incremented X to -4
Increment Thread: Incremented X to -3
Increment Thread: Incremented X to -2
Increment Thread: Incremented X to -1
Increment Thread: Incremented X to 0
student1@student1-VirtualBox:~/Desktop$
```

3. Create two threads Odd and Even where odd prints the odd number in "N" and even prints the even number in "N". Use synchronisation technique so that the output is proper sequence.
   Example: In N=10,
   Output: 1 2 3 4 5 6 7 8 9 10.
   Program:

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

OPERATING SYSTEM LAB

Name: Thorat Amey Arun Reg No.: 23MCS1004

```
int N = 10;
pthread_mutex_t mutex;
pthread_cond_t odd_condition;
pthread_cond_t even_condition;
int next_number = 1;
void* odd_thread(void* arg) {
   while (1) {
      pthread_mutex_lock(&mutex);
      if (next_number > N) {
         pthread_mutex_unlock(&mutex);
         break;
      }
      if (next_number % 2 == 0) {
         pthread_cond_wait(&odd_condition, &mutex);
      } else {
         printf("%d ", next_number);
         next_number++;
      }
      pthread_cond_signal(&even_condition);
      pthread_mutex_unlock(&mutex);
   }
   return NULL;
}
void* even_thread(void* arg) {
   while (1) {
      pthread_mutex_lock(&mutex);
      if (next_number > N) {
         pthread_mutex_unlock(&mutex);
         break;
      }
      if (next_number % 2 != 0) {
         pthread_cond_wait(&even_condition, &mutex);
      } else {
         printf("%d ", next_number);
         next_number++;
      }
      pthread_cond_signal(&odd_condition);
      pthread_mutex_unlock(&mutex);
   }
```

```
    return NULL;
}
int main() {
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&odd_condition, NULL);
    pthread_cond_init(&even_condition, NULL);
    pthread_t odd_tid, even_tid;
    pthread_create(&odd_tid, NULL, odd_thread, NULL);
    pthread_create(&even_tid, NULL, even_thread, NULL);
    pthread_join(odd_tid, NULL);
    pthread_join(even_tid, NULL);
    printf("\n");
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&odd_condition);
    pthread_cond_destroy(&even_condition);
    return 0;
}
```

Output:

```
student1@student1-VirtualBox:~/Desktop$ gcc -o 7,3 7,3.c
student1@student1-VirtualBox:~/Desktop$ ./7,3
1 2 3 4 5 6 7 8 9 10
student1@student1-VirtualBox:~/Desktop$
```

4. Develop a code using Threads to provide synchronisation to reader writer problem where reader can be N(user input) and Writer can be M(user input).
   Run the program multiple times with same input and check the output.
   Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define MAX_READERS 10
#define MAX_WRITERS 5
int sharedData = 0; // Shared data that readers and writers access
int readersCount = 0; // Number of active readers
```

```
int writersCount = 0; // Number of active writers
sem_t readMutex, writeMutex, resourceAccess;
void* reader(void* arg) {
int readerId = *((int*)arg);
while (1) {
sem_wait(&readMutex);
readersCount++;
if (readersCount == 1) {
sem_wait(&resourceAccess);
}
sem_post(&readMutex);
// Read shared data
printf("Reader %d: Read %d\n", readerId, sharedData);
sem_wait(&readMutex);
readersCount--;
if (readersCount == 0) {
sem_post(&resourceAccess);
}
sem_post(&readMutex);
usleep(100000); // Sleep for a short time to simulate reading
}
return NULL;
}

void* writer(void* arg) {
int writerId = *((int*)arg);
while (1) {
sem_wait(&writeMutex);
writersCount++;
if (writersCount == 1) {
sem_wait(&resourceAccess);
}
sem_post(&writeMutex);
// Write to shared data
sharedData++;
printf("Writer %d: Wrote %d\n", writerId, sharedData);
sem_wait(&writeMutex);
writersCount--;
if (writersCount == 0) {
```

```
sem_post(&resourceAccess);
}
sem_post(&writeMutex);
usleep(200000); // Sleep for a short time to simulate writing
}
return NULL;
}

int main() {
int numReaders, numWriters;
printf("Enter the number of readers (1-%d): ", MAX_READERS);
scanf("%d", &numReaders);
if (numReaders < 1 || numReaders > MAX_READERS) {
printf("Invalid number of readers.\n");
return 1;
}
printf("Enter the number of writers (1-%d): ", MAX_WRITERS);
scanf("%d", &numWriters);
if (numWriters < 1 || numWriters > MAX_WRITERS) {
printf("Invalid number of writers.\n");
return 1;
}
// Initialize semaphores
sem_init(&readMutex, 0, 1);
sem_init(&writeMutex, 0, 1);
sem_init(&resourceAccess, 0, 1);
pthread_t readerThreads[MAX_READERS];
pthread_t writerThreads[MAX_WRITERS];
int readerIds[MAX_READERS];
int writerIds[MAX_WRITERS];
// Create reader threads
for (int i = 0; i < numReaders; i++) {
readerIds[i] = i + 1;
pthread_create(&readerThreads[i], NULL, reader, &readerIds[i]);
}
// Create writer threads
for (int i = 0; i < numWriters; i++) {
writerIds[i] = i + 1;
pthread_create(&writerThreads[i], NULL, writer, &writerIds[i]);
```

```
}
// Wait for reader and writer threads to finish (not typically done in a real
scenario)
for (int i = 0; i < numReaders; i++) {
pthread_join(readerThreads[i], NULL);
}
for (int i = 0; i < numWriters; i++) {
pthread_join(writerThreads[i], NULL);
}
// Destroy semaphores
sem_destroy(&readMutex);
sem_destroy(&writeMutex);
sem_destroy(&resourceAccess);
return 0;
}
```

Output:

```
vboxuser@Ubuntu:~/Desktop$ gcc 74.c -o 74
vboxuser@Ubuntu:~/Desktop$ ./74
Enter the number of readers (1-10): 3
Enter the number of writers (1-5): 2
Reader 1: Read 0
Reader 2: Read 0
Reader 3: Read 0
Writer 1: Wrote 1
Writer 2: Wrote 2
Reader 1: Read 2
Reader 2: Read 2
Reader 3: Read 2
Writer 1: Wrote 3
Writer 2: Wrote 4
Reader 2: Read 4
Reader 3: Read 4
Reader 1: Read 4
Reader 2: Read 4
Reader 3: Read 4
Reader 1: Read 4
Reader 3: Read 4
Writer 2: Wrote 5
Reader 1: Read 5
Writer 1: Wrote 6
Reader 2: Read 6
Reader 1: Read 6
Reader 2: Read 6
Reader 3: Read 6
^C
vboxuser@Ubuntu:~/Desktop$
```