Name: Spandan Mukherjee
Registration Number: 21BCE1132
Subject: Operating Systems Lab

Experiment: Process Synchronization

1)

Develop a readers and writers problem with minimum 2 readers and 2 writers, Ensure that synchronisation is done with semaphore and satisfy the 4 below mentioned conditions. Shared data as an integer variable and let the writers do the increment operations and readers do the shared variable read operation.

Use thread functions from pthreaed.h header file to create pthreads which will perform read and write operations.
Use functions from semaphore.h header file to do synchronisation to perform r-w operations with out any data inconsistency and hence avoid the race conditions.

ase	Process 1	Process 2	Allowed/Not Allowed
Case 1	Writing	Writing	Not Allowed
Case 2	Writing	Reading	Not Allowed
Case 3	Reading	Writing	Not Allowed
Case 4	Reading	Reading	Allowed

Reference:
https://www.ibm.com/docs/en/i/
https://www.geeksforgeeks.org/

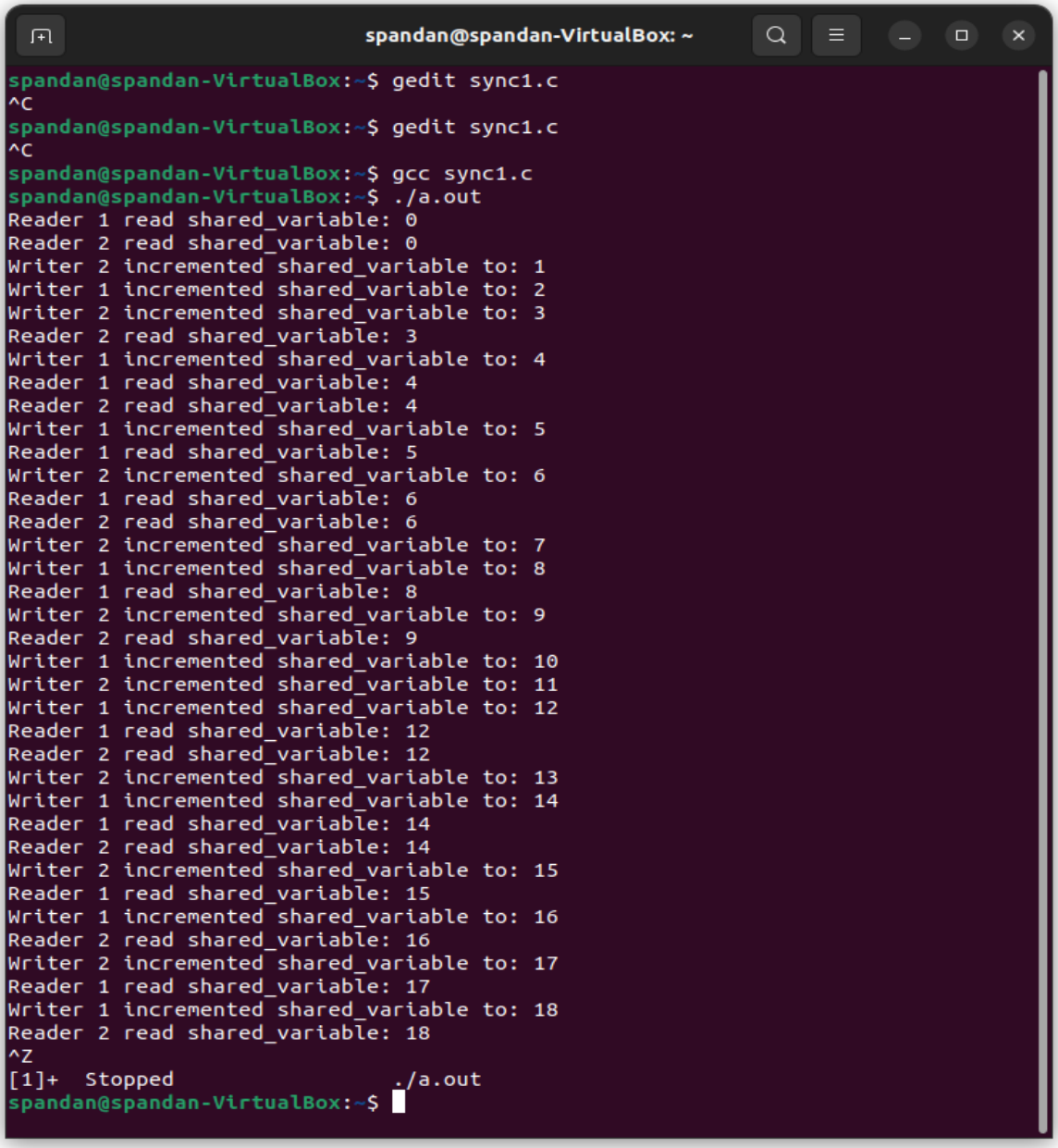
CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define NUM_READERS 2
#define NUM_WRITERS 2

int shared_variable = 0;
sem_t mutex, rw_mutex;
void *reader(void *arg) {
   int id = *(int *)arg;

   while (1) {
      sem_wait(&rw_mutex);
      sem_wait(&mutex);       // acquire mutex to access shared_variable
      printf("Reader %d read shared_variable: %d\n", id, shared_variable);
```

```
        sem_post(&mutex);       // release mutex
        sem_post(&rw_mutex);    // release rw_mutex
        sleep(1);               // sleep for 1 second
    }
}

void *writer(void *arg) {
    int id = *(int *)arg;

    while (1) {
        sem_wait(&mutex);       // acquire mutex to access shared_variable
        shared_variable++;      // increment shared_variable
        printf("Writer %d incremented shared_variable to: %d\n", id, shared_variable);
        sem_post(&mutex);       // release mutex
        sleep(1);               // sleep for 1 second
    }
}

int main() {
    pthread_t readers[NUM_READERS];
    pthread_t writers[NUM_WRITERS];
    int reader_ids[NUM_READERS], writer_ids[NUM_WRITERS];
    int i;

    //semaphores
    sem_init(&mutex, 0, 1);
    sem_init(&rw_mutex, 0, NUM_READERS);

    // create of threads
    for (i = 0; i < NUM_READERS; i++) {
        reader_ids[i] = i + 1;
        pthread_create(&readers[i], NULL, reader, &reader_ids[i]);
    }

    // create of writer threads
    for (i = 0; i < NUM_WRITERS; i++) {
        writer_ids[i] = i + 1;
        pthread_create(&writers[i], NULL, writer, &writer_ids[i]);
    }

    // wait for threads to finish
    for (i = 0; i < NUM_READERS; i++) {
        pthread_join(readers[i], NULL);
    }

    for (i = 0; i < NUM_WRITERS; i++) {
        pthread_join(writers[i], NULL);
```

```
    }

    // destroy semaphores
    sem_destroy(&mutex);
    sem_destroy(&rw_mutex);

    return 0;
}
```

OUTPUT:



2)

Modify the question 1 as 1 writer performs increment operation and another writer performs decrement operation of the same account. Readers read and display the shared variable.

Algorithm:

To solve this problem using semaphores, we can use the following approach:

1. Create two semaphores: a "mutex" semaphore to protect the critical section (i.e., the shared resource) and a "write" semaphore to allow writers to write to the shared resource.
2. Initialize the "mutex" semaphore to 1 and the "write" semaphore to 1 (i.e., allow one writer to write at a time).
3. Create a writer thread that will increment the shared variable by 1 (or any other value as required).
4. Create a reader thread that will read the shared variable.
5. When a writer wants to write to the shared resource, it must first acquire the "write" semaphore. Once it has acquired the semaphore, it can then acquire the "mutex" semaphore to enter the critical section and write to the shared resource. After writing, it releases the "mutex" semaphore and the "write" semaphore.
6. When a reader wants to read from the shared resource, it must first acquire the "mutex" semaphore to enter the critical section. After reading, it releases the "mutex" semaphore.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>
int reader_no = 1;
int writer_no = 1;
int shared_data = 0;
int readers_count = 0;
sem_t mutex, write_mutex, read_mutex;
void *reader(void *arg) {
int id = *(int *) arg;
while (1) {
sem_wait(&read_mutex);
readers_count++;
if (readers_count == 1) {
sem_wait(&write_mutex);
}
sem_post(&read_mutex);
printf("Reader %d read shared data: %d\n", id, shared_data);
sem_wait(&read_mutex);
readers_count--;
```

```
if (readers_count == 0) {
sem_post(&write_mutex);
}
sem_post(&read_mutex);
sleep(1);
}
}
void *increment_writer(void *arg) {
int id = *(int *) arg;
while (1) {
sem_wait(&mutex);
shared_data++;
printf("Increment writer %d wrote to shared data: %d\n", id, shared_data);

sem_post(&mutex);
sleep(1);
}
}
void *decrement_writer(void *arg) {
int id = *(int *) arg;
while (1) {
sem_wait(&mutex);
shared_data--;
printf("Decrement writer %d wrote to shared data: %d\n", id, shared_data);
sem_post(&mutex);
sleep(1);
}
}
int main() {
pthread_t readers[reader_no], increment_writers[writer_no], decrement_writers[writer_no];
int reader_ids[reader_no], increment_writer_ids[writer_no], decrement_writer_ids[writer_no];
int i;
sem_init(&mutex, 0, 1);
sem_init(&write_mutex, 0, 1);
sem_init(&read_mutex, 0, 1);
for (i = 0; i < reader_no; i++) {
reader_ids[i] = i;
pthread_create(&readers[i], NULL, reader, &reader_ids[i]);
}
increment_writer_ids[0] = 0;
pthread_create(&increment_writers[0], NULL, increment_writer, &increment_writer_ids[0]);
decrement_writer_ids[0] = 1;
pthread_create(&decrement_writers[0], NULL, decrement_writer, &decrement_writer_ids[0]);
for (i = 0; i < reader_no; i++) {
pthread_join(readers[i], NULL);
}
pthread_join(increment_writers[0], NULL);
pthread_join(decrement_writers[0], NULL);
sem_destroy(&mutex);
sem_destroy(&write_mutex);
sem_destroy(&read_mutex);
```

```
return 0;
}
```

OUTPUT:

```
spandan@spandan-VirtualBox: ~
spandan@spandan-VirtualBox:~$ gedit sync2.c
^C
spandan@spandan-VirtualBox:~$ gcc sync2.c
spandan@spandan-VirtualBox:~$ ./a.out
Reader 0 read shared data: 0
Increment writer 0 wrote to shared data: 1
Decrement writer 1 wrote to shared data: 0
Increment writer 0 wrote to shared data: 1
Decrement writer 1 wrote to shared data: 0
Reader 0 read shared data: 0
Decrement writer 1 wrote to shared data: -1
Reader 0 read shared data: -1
Increment writer 0 wrote to shared data: 0
Decrement writer 1 wrote to shared data: -1
Increment writer 0 wrote to shared data: 0
Reader 0 read shared data: 0
Decrement writer 1 wrote to shared data: -1
Increment writer 0 wrote to shared data: 0
Reader 0 read shared data: 0
Decrement writer 1 wrote to shared data: -1
Reader 0 read shared data: -1
Increment writer 0 wrote to shared data: 0
Decrement writer 1 wrote to shared data: -1
Reader 0 read shared data: -1
Increment writer 0 wrote to shared data: 0
Decrement writer 1 wrote to shared data: -1
Reader 0 read shared data: -1
Increment writer 0 wrote to shared data: 0
Decrement writer 1 wrote to shared data: -1
Reader 0 read shared data: -1
Increment writer 0 wrote to shared data: 0
Decrement writer 1 wrote to shared data: -1
Reader 0 read shared data: -1
Increment writer 0 wrote to shared data: 0
Decrement writer 1 wrote to shared data: -1
Increment writer 0 wrote to shared data: 0
Reader 0 read shared data: 0
Decrement writer 1 wrote to shared data: -1
Increment writer 0 wrote to shared data: 0
Reader 0 read shared data: 0
Decrement writer 1 wrote to shared data: -1
Increment writer 0 wrote to shared data: 0
Reader 0 read shared data: 0
Decrement writer 1 wrote to shared data: -1
Reader 0 read shared data: -1
Increment writer 0 wrote to shared data: 0
Decrement writer 1 wrote to shared data: -1
Reader 0 read shared data: -1
```