REPORT

Assignment 4: Classical Problems of Synchronization

Shashwat Johri
12041380

PART A-

The main parts of the code are:

Making the two threads consumer and producer:

```
int main(){
    pthread_t thread[2];
    pthread_create(&thread[0],NULL,producer,NULL);
    pthread_create(&thread[1],NULL,consumer,NULL);
    pthread_join(thread[0],NULL);
    pthread_join(thread[1],NULL);

    return 0;
}
```

Inside the producer function, it waits for the consumer to finish doing its work and gives up the lock:

```
void *producer(void *arg) {

    pthread_mutex_lock(&mutex);
    while(consdone==0){
        pthread_cond_wait(&wakepro,&mutex);
    }
    prodone=0;
```

for the next 1 ms , producer writers numbers onto the buffer

```
t = clock();
while(timeelapsed(t)<1){
    //instead of 10 ms, we're doing 1 ms
    //its possible to change to 10 ms
    writebuf(i);
    i=(i+1)%MAX;
    sleep(0.1);

}
```

.

after this , producer sets the prodone flag to 1 and signals the waiting consumer thread that its work is done

```
prodone=1;
pthread_cond_signal(&wakecon);
pthread_mutex_unlock(&mutex);
```

Similarly in consumer, buffer is read and flushed:.

```
void *consumer(void *arg) {
    pthread_mutex_lock(&mutex);
    while(prodone==0){
        pthread_cond_wait(&wakecon,&mutex);
    }
    consdone=0;

    readbuf();
    flush();

    consdone=1;
    pthread_cond_signal(&wakepro);
    pthread_mutex_unlock(&mutex);

}
```

PART B-

the write thread aquires sthe write semaphore and increments the count , after doing so it incremenets the read semphore, which can now operate.

```
void * writethread(void *args) {

   for(int i=0;i<=20;i++){
        sem_wait(&semwrite); //aquire the write semaphore
        //sleep(1);
        count++;
        printf("Count is: %d and incrementing thread id:
        %d\n", count,gettid());
        sem_post(&semread); //release the read semphore, now
        read thread can operate
    }

}
```

.

The read thread does similar thing , it aquires the read semaphore, and after dinishes it releases the write semphore:

```
void * readthread(void *args) {


   for(int i=0;i<=20;i++) {
        sem_wait(&semread);//aquire the read semaphore
        //sleep(2);
        printf("Count is: %d and reading thread id:      %d
        \n\n",count ,gettid());
        sem_post(&semwrite);//release the write semphore so
        that the next write can take place
    }

}
```

.

PART C-

The writer thread aquires the writer semaphore and displays the number of writers present write now . It then incremenets the shared variable and finally exits after giving up the writer semaphore

```
void *writer(void *args)
{
    sem_wait(&writers_smph);//aquire the writer semaphore
    writer_count++;
    printf("Writer count : %d\n",writer_count);
    shared_variable++;//incrementing the shared variable
    printf("Writer is writing the variable : %d\n",
    shared_variable);
    writer_count--;
    if(writer_count==0){
        printf("Writer count : %d\n",writer_count);
    }
    sem_post(&writers_smph);//give up the writer semaphore

}
```

The reader thread firstly increases the reader count , it then checks if it the first reader, in that case it aquires the writer semaphore.

Then it displays the shared variable and read count

```
void *reader(void *rid)
{

    pthread_mutex_lock(&mutex);
    reader_count++;//increase the number of readers
    if(reader_count == 1) {
        sem_wait(&writers_smph); //the first reader aquires
        the writer semaphore
    }
    printf("Reader count %d\n", reader_count);
    pthread_mutex_unlock(&mutex);
    printf("Reader %d: reading the variable: %d\n",*((int *)
    rid),shared_variable);//any number of readers can read the
    shared variable at the same time

    pthread_mutex_lock(&mutex);
```

.

The reader then decrements the reader count and checks if the reader is the last reader, in which case it gives up the writer sempaphore and a waiting writer can write again!

```
  printf("Reader %d left\n",*((int *)rid));

  reader_count--;//decrement the number of readers as the
  reader leaves
  if(reader_count == 0) {
      sem_post(&writers_smph);//the last reader gives up the
      writer semaphore, so that writer can aquire it
  }
  pthread_mutex_unlock(&mutex);


t main()
```