

# OPERATING SYSTEMS PRACTICE

## (ASSIGNMENT 7)

**Name** – Mridul Harish

**Roll number** – CED18I034

**Question 1** : Simulate the producer consumer code discussed in the class.

**Algorithm** :-

```
do
{
    //produce an item or consume an item
    wait(empty);
    wait(mutex);
    //place in buffer

    signal(mutex);
    signal(full);
}while(true)
```

**Code** :-

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

#define BufferSize 5 //Size of the buffer

int in = 0; int out = 0; int buffer[BufferSize];

void *producer(void *pno);
void *consumer(void *cno);

int main()
{
    srand(time(0));
    pthread_t pro[5], con[5];

    for(int i = 0; i < 5; i = i+1)
    {
        pthread_create(&pro[i], NULL, (void *)producer, NULL);
```

```

        pthread_create(&con[i], NULL, (void *)consumer, NULL);
    }

    for(int i = 0; i < 5; i = i+1)
    {
        pthread_join(pro[i], NULL);
        pthread_join(con[i], NULL);
    }

    return 0;
}

void *producer(void *pno)
{
    int item;
    item = rand()%101; //Produce an random item
    buffer[in] = item;
    printf("Producer Insert Item %d at %d\n",buffer[in],in);
    in = (in+1)%BufferSize;
}

void *consumer(void *cno)
{
    int item = buffer[out];
    printf("Consumer Remove Item %d from %d\n",item, out);
    out = (out+1)%BufferSize;
}

```

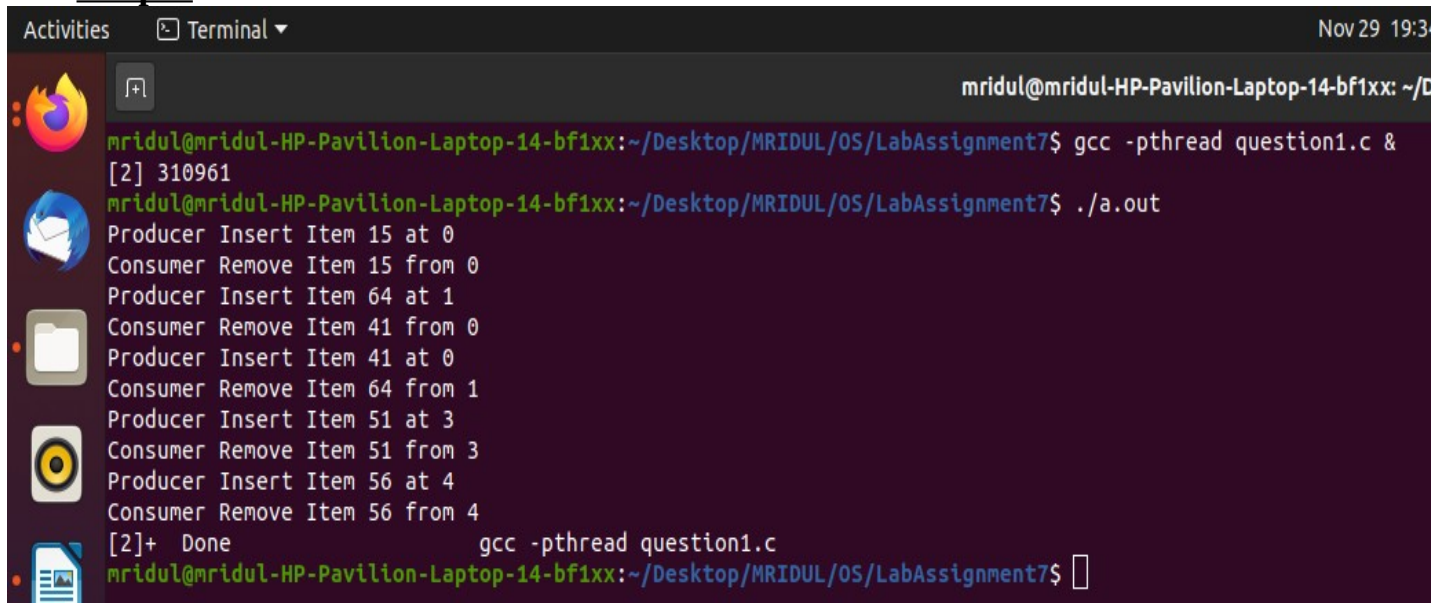
### **Explanation :-**

In the given problem a producer can produce an item and can place it in the buffer while a consumer can pick a item from the buffer and consume them. The buffer in question is of fixed size here.

We need to make sure that when producer is placing an item in the buffer, no consumer is consuming an item at the same time from the buffer. So the critical section is the buffer.

So we here need 2 counting semaphores in the form of “Full” which keeps the tracks of number of items in the buffer at any given time and “Empty” keeps track of number of unoccupied slots.

## Output :



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx: ~/Desktop/MRIDUL/OS/LabAssignment7$ gcc -pthread question1.c &
[2] 310961
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$ ./a.out
Producer Insert Item 15 at 0
Consumer Remove Item 15 from 0
Producer Insert Item 64 at 1
Consumer Remove Item 41 from 0
Producer Insert Item 41 at 0
Consumer Remove Item 64 from 1
Producer Insert Item 51 at 3
Consumer Remove Item 51 from 3
Producer Insert Item 56 at 4
Consumer Remove Item 56 from 4
[2]+ Done gcc -pthread question1.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$
```

**Question 2 :** Extend the producer consumer simulation in Q1 to sync access of critical data using Peterson's algorithm.

### Code :-

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

#define BufferSize 5 //Size of the buffer
#define prod 0
#define cons 1

int in = 0; int out = 0; int buffer[BufferSize]; int flag[2] = {0,0}; int turn;

void *producer(void *pno);
void *consumer(void *cno);

int main()
{
    srand(time(0));
    pthread_t pro[5],con[5];
    for(int i = 0; i < 5; i = i+1)
    {
        pthread_create(&pro[i], NULL, (void *)producer, NULL);
        pthread_create(&con[i], NULL, (void *)consumer, NULL);
    }
}
```

```

    }

    for(int i = 0; i < 5; i = i+1)
    {
        pthread_join(pro[i], NULL);
        pthread_join(con[i], NULL);
    }

    return 0;
}

void *producer(void *pno)
{
    int item;
    flag[prod]=1;
    turn = cons;

    while (flag[cons] == 1 && turn == cons);
    item = rand()%101; //Produce an random item
    buffer[in] = item;
    printf("Producer Insert Item %d at %d\n", buffer[in], in);
    in = (in+1)%BufferSize;
    flag[prod]=0;
}

void *consumer(void *cno)
{
    flag[cons]=1;
    turn = prod;

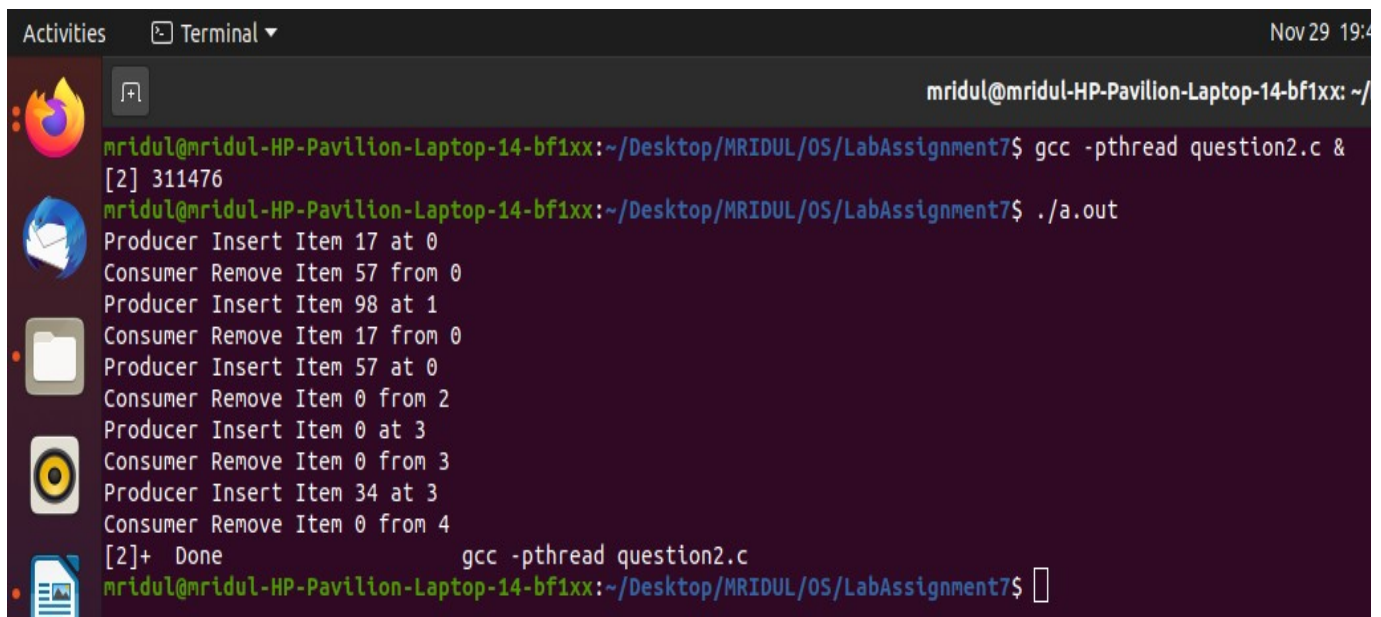
    while (flag[prod] == 1 && turn == prod);
    int item = buffer[out];
    printf("Consumer Remove Item %d from %d\n",item, out);
    out = (out+1)%BufferSize;
    flag[cons]=0;
}

```

### **Explanation :-**

So here first a thread would express its desire to acquire a lock and sets flag[self] = 1 and then gives the other thread a chance to acquire the lock. If the thread desires to acquire a lock then it gets the lock and passed the chance to the 1<sup>st</sup> thread. If it does not desire to get the lock then the while loop breaks and the 1<sup>st</sup> thread gets the chance.

## Output :



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx: ~/Desktop/MRIDUL/OS/LabAssignment7$ gcc -pthread question2.c &
[2] 311476
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$ ./a.out
Producer Insert Item 17 at 0
Consumer Remove Item 57 from 0
Producer Insert Item 98 at 1
Consumer Remove Item 17 from 0
Producer Insert Item 57 at 0
Consumer Remove Item 0 from 2
Producer Insert Item 0 at 3
Consumer Remove Item 0 from 3
Producer Insert Item 34 at 3
Consumer Remove Item 0 from 4
[2]+ Done
gcc -pthread question2.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$
```

**Question 3 :** Dictionary problem – Let the producer set up a dictionary of at least 20 words with three attributes(word, primary meaning, secondary meaning) and let the consumer search for the word and retrieve its respective primary and secondary meaning.

## Code :-

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

sem_t wrt;
char search[100];
pthread_mutex_t mutex;
int numreader = 0; int k=0; int flag;

typedef struct
{
    char word[100];
    char primary[1000];
    char secondary[100];
}dict;

dict dictionary[5];

void *writer(void *wno);
```

```

void *reader(void *rno);

int main()
{
    pthread_t read,write[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&wrt,0,1);

    for(int i = 0; i < 5; i = i+1)
    {
        pthread_create(&write[i], NULL, (void *)writer, NULL);
    }

    pthread_create(&read, NULL, (void *)reader, NULL);
    pthread_join(read, NULL);

    for(int i = 0; i < 5; i = i+1)
    {
        pthread_join(write[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    sem_destroy(&wrt);
    return 0;
}

void *writer(void *wno)
{
    sem_wait(&wrt);
    flag=1;

    printf("Enter word:\n");
    scanf("%s",search);
    int i;

    for(i = 0; i < 5; i = i+1) //duplicacy check
    {
        if(strcmp(dictionary[i].word,search) == 0)
        {
            printf("Word already present.\n");
            flag=0;
            break;
        }
    }
}

```

```

    if(flag==1)
    {
        strcpy(dictionary[k].word,search);
        printf("Enter meaning:\n");
        scanf("%s",dictionary[k].primary);
        printf("Enter secondary meaning:\n");
        scanf("%s",dictionary[k].secondary);
        printf("Writer added a word %s.\n", dictionary[k].word);
        k = k+1;
    }

    sem_post(&wrt);
}

void *reader(void *rno)
{
    //Reader acquire the lock before modifying numreader
    pthread_mutex_lock(&mutex);
    numreader = numreader + 1;

    if(numreader == 1)
    {
        sem_wait(&wrt); //If this id the first reader, then it will block the writer
    }

    pthread_mutex_unlock(&mutex); //Reading Section
    int i;
    printf("Enter word you wanna search: \n");
    scanf("%s",search);
    int low = 0;
    int high = 4;

    while(low <= high)
    {
        int mid = (low+high)/2;
        if(strcmp(search,dictionary[mid].word) == 0)
        {
            printf("Meaning: %s\n", dictionary[mid].primary);
            printf("Secondary Meaning: %s\n", dictionary[mid].secondary);
            exit(0);
        }
        else if(strcmp(search,dictionary[mid].word) > 0)
        {
            high = high;
            low = mid+1;
        }
    }
}

```

```

        else
        {
            low = low;
            high = mid-1;
        }
    }
    printf("Word not found\n"); //Reader acquire the lock before modifying
numreader

    pthread_mutex_lock(&mutex);
    numreader = numreader - 1;

    if(numreader == 0)
    {
        sem_post(&wrt); //If this is the last reader, it will wake up the writer.
    }

    pthread_mutex_unlock(&mutex);
}

```

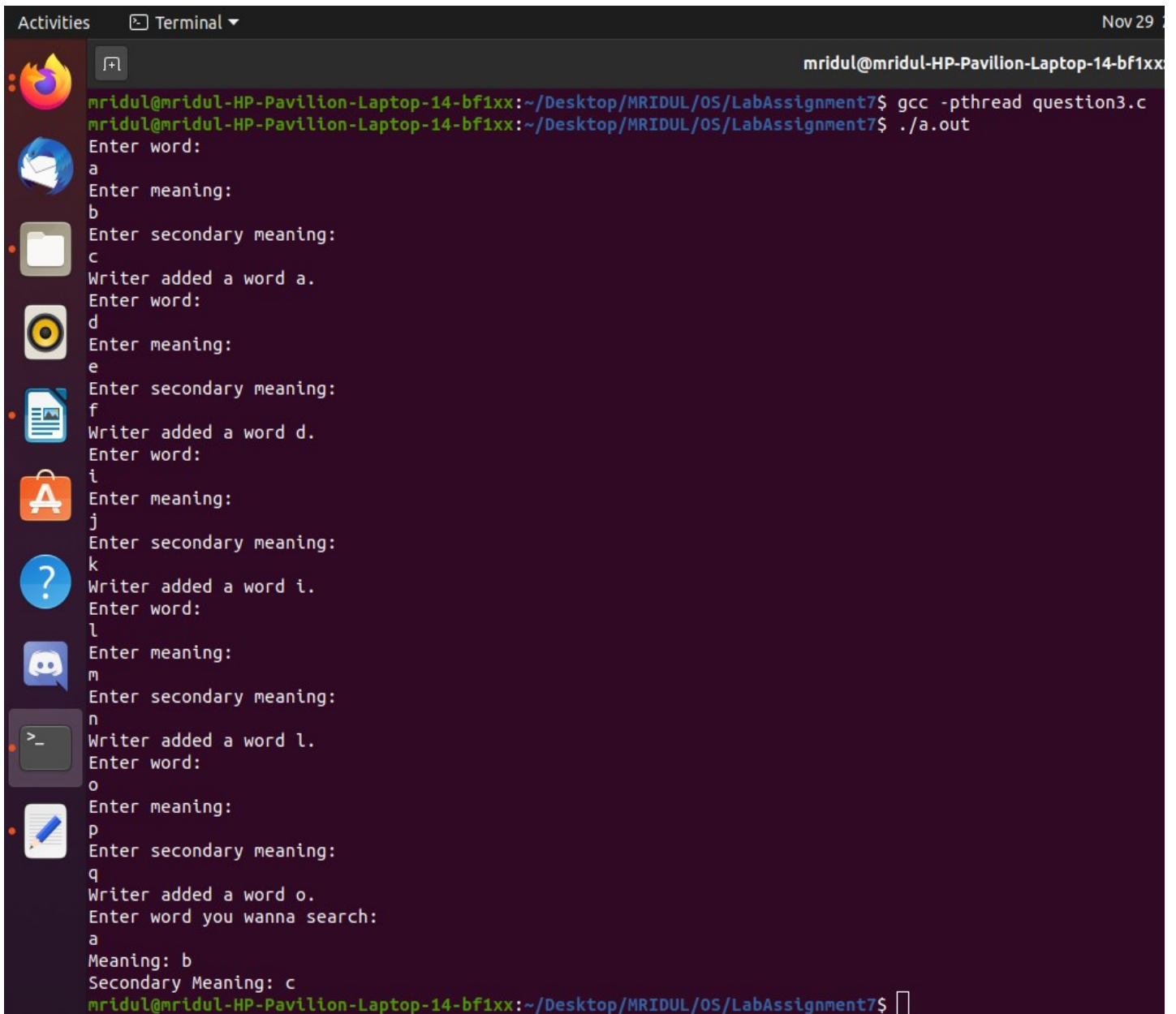
### **Explanation :-**

Here, we use one mutex m and a semaphore w.

1. An integer variable read\_count - used to maintain the number of readers currently accessing the resource. The variable read\_count is initialized to 0.3.
2. A value of 1 is given initially to m and w.
3. Instead of having the process to acquire lock on the shared resource, we use the mutex m to make the process to acquire and release lock whenever it is updating the read\_count variable.
  - a. Writer Process : Writer requests the entry to critical section. If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting. It exits the critical section.
  - b. Reader Process : Reader requests the entry to critical section. If allowed :
    - i. It increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the w semaphore to restrict the entry of writers if any reader is inside.
    - ii. It then, signals mutex as any other reader is allowed to enter while others are already reading.
    - iii. After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore w as now, writer can enter the critical section. If not allowed, it keeps on waiting.



## Output :



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$ gcc -pthread question3.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$ ./a.out
Enter word:
a
Enter meaning:
b
Enter secondary meaning:
c
Writer added a word a.
Enter word:
d
Enter meaning:
e
Enter secondary meaning:
f
Writer added a word d.
Enter word:
i
Enter meaning:
j
Enter secondary meaning:
k
Writer added a word i.
Enter word:
l
Enter meaning:
m
Enter secondary meaning:
n
Writer added a word l.
Enter word:
o
Enter meaning:
p
Enter secondary meaning:
q
Writer added a word o.
Enter word you wanna search:
a
Meaning: b
Secondary Meaning: c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$
```

**Question 4 :** Extend Q3 to avoid duplication of dictionary entries and implement an efficient binary search on the consumer side in a multithreaded fashion.

### Explanation :-

Same as the previous question.

### Code :-

```
#include<pthread.h>
#include<semaphore.h>
#include<string.h>
```

```

#include<stdio.h>
#include<stdlib.h>

sem_t wrt;
char search[100];
pthread_mutex_t mutex;
int numreader = 0; int k=0; int flag;

typedef struct
{
    char word[100];
    char primary[1000];
    char secondary[100];
}dict;

dict dictionary[5];

void *writer(void *wno);
void *reader(void *rno);

int main()
{
    pthread_t read,write[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&wrt,0,1);

    for(int i = 0; i < 5; i = i+1)
    {
        pthread_create(&write[i], NULL, (void *)writer, NULL);
    }

    pthread_create(&read, NULL, (void *)reader, NULL);
    pthread_join(read, NULL);

    for(int i = 0; i < 5; i = i+1)
    {
        pthread_join(write[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    sem_destroy(&wrt);
    return 0;
}

void *writer(void *wno)

```

```

{
    sem_wait(&wrt);
    flag=1;

    printf("Enter word:\n");
    scanf("%s",search);
    int i;

    for(i = 0; i < 5; i = i+1) //duplicacy check
    {
        if(strcmp(dictionary[i].word,search) == 0)
        {
            printf("Word already present.\n");
            flag=0;
            break;
        }
    }

    if(flag==1)
    {
        strcpy(dictionary[k].word,search);
        printf("Enter meaning:\n");
        scanf("%s",dictionary[k].primary);
        printf("Enter secondary meaning:\n");
        scanf("%s",dictionary[k].secondary);
        printf("Writer added a word %s.\n", dictionary[k].word);
        k = k+1;
    }

    sem_post(&wrt);
}

void *reader(void *rno)
{
    //Reader acquire the lock before modifying numreader
    pthread_mutex_lock(&mutex);
    numreader = numreader + 1;

    if(numreader == 1)
    {
        sem_wait(&wrt); //If this id the first reader, then it will block the writer
    }

    pthread_mutex_unlock(&mutex); //Reading Section
    int i;
    printf("Enter word you wanna search: \n");

```

```

scanf("%s",search);
int low = 0;
int high = 4;

while(low <= high)
{
    int mid = (low+high)/2;
    if(strcmp(search,dictionary[mid].word) == 0)
    {
        printf("Meaning: %s\n", dictionary[mid].primary);
        printf("Secondary Meaning: %s\n", dictionary[mid].secondary);
        exit(0);
    }
    else if(strcmp(search,dictionary[mid].word) > 0)
    {
        high = high;
        low = mid+1;
    }
    else
    {
        low = low;
        high = mid-1;
    }
}
printf("Word not found\n"); //Reader acquire the lock before modifying
numreader


pthread_mutex_lock(&mutex);
numreader = numreader - 1;

if(numreader == 0)
{
    sem_post(&wrt); //If this is the last reader, it will wake up the writer.
}

pthread_mutex_unlock(&mutex);
}

```

## Output :



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$ gcc -pthread question3.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$ ./a.out
Enter word:
done
Enter meaning:
finished
Enter secondary meaning:
completed
Writer added a word done.
Enter word:
done
Word already present.
Enter word:

```

## Question 5 : Dekker's algorithm

### Algorithm :-

Main()

{

// to denote which thread will enter next  
int favouredthread = 1;

// flags to indicate if each thread is in queue to enter its critical section  
boolean thread1wantstoenter = false;  
boolean thread2wantstoenter = false;

startThreads();

}

Thread1()

{

do  
{

thread1wantstoenter = true; // entry section  
// wait until thread2 wants to enter its critical section

while (thread2wantstoenter == true)  
{

// if 2nd thread is more favored  
if (favouredthread == 2)

```

        {
            // gives access to other thread
            thread1wantstoenter = false;
            // wait until this thread is favored

            while (favouredthread == 2) ;
            thread1wantstoenter = true;
        }
    }

    // critical section
    // favor the 2nd thread
    favouredthread = 2;

    // exit section
    // indicate thread1 has completed its critical section
    thread1wantstoenter = false;

    // remainder section
}
while (completed == false)
}

Thread2()
{
    do
    {
        thread2wantstoenter = true;
        // entry section
        // wait until thread1 wants to enter its critical section

        while (thread1wantstoenter == true)
        {
            // if 1st thread is more favored
            if (favaouredthread == 1)
            {
                // gives access to other thread
                thread2wantstoenter = false;
                // wait until this thread is favored
                while (favouredthread == 1) ;
                thread2wantstoenter = true;
            }
        }
        // critical section
        // favour the 1st thread

```

```

        favouredthread = 1;
        // exit section
        // indicate thread2 has completed its critical section
        thread2wantstoenter = false;
        // remainder section
    }
    while (completed == false)
}

```

This version guarantees a complete solution to the critical solution problem.

### **Explanation :-**

Dekker's Algorithm : Final and completed Solution – the idea is to use favoured thread notion to determine entry to the critical section. Favoured thread alternates between the thread providing mutual exclusion and avoiding deadlock, indefinite postponement or lockstep synchronization

### **Question 6 : Dining Philosopher's problem**

#### **Code :-**

```

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

#define N 5 //no. of philosophers
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

int state[N]; int phil[N] = { 0, 1, 2, 3, 4 };
sem_t mutex; sem_t S[N];

void test(int phnum);
void take_fork(int phnum);
void put_fork(int phnum);
void* philosopher(void* num);

int main()
{
    int i;

```

```

pthread_t thread_id[N]; //initialize the semaphores

sem_init(&mutex, 0, 1);
for(i = 0; i < N; i = i+1)
    sem_init(&S[i], 0, 0);

for(i = 0; i < N; i = i+1)
{
    // create philosopher processes
    pthread_create(&thread_id[i], NULL, philosopher, &phil[i]);
    printf("Philosopher %d is thinking\n", i + 1);
}

for(i = 0; i < N; i = i+1)
    pthread_join(thread_id[i], NULL);
}

void test(int phnum)
{
    if(state[phnum] == HUNGRY && state[LEFT] != EATING &&
state[RIGHT] != EATING)
    {
        state[phnum] = EATING; //state that eating
        sleep(2); //eating time

        printf("Philosopher %d takes fork %d and %d\n", phnum + 1, LEFT + 1,
phnum + 1);
        printf("Philosopher %d is Eating\n", phnum + 1);
        sem_post(&S[phnum]);
    }
}

// take up chopsticks
void take_fork(int phnum)
{
    sem_wait(&mutex); // state that hungry
    state[phnum] = HUNGRY;
    printf("Philosopher %d is Hungry\n", phnum + 1); // eat if neighbours are not
eating
    test(phnum);

    sem_post(&mutex); // if unable to eat wait to be signalled
    sem_wait(&S[phnum]);
    sleep(1);
}

```



```

// put down chopsticks
void put_fork(int phnum)
{
    sem_wait(&mutex); //state that thinking
    state[phnum] = THINKING;
    printf("Philosopher %d putting fork %d and %d down\n", phnum + 1, LEFT +
1, phnum + 1);
    printf("Philosopher %d is thinking\n", phnum + 1);

    test(LEFT);
    test(RIGHT);
    sem_post(&mutex);
}

void* philosopher(void* num)
{
    while(1)
    {
        int* i = num;
        sleep(1);
        take_fork(*i);
        sleep(0); //immediately drops fork after eating

        put_fork(*i);
    }
}

```

### **Explanation :-**

Solution of Dining Philosophers Problem - A solution of the Dining Philosophers Problem is to use a semaphore to represent a chopstick. A chopstick can be picked up by executing a wait operation on the semaphore and released by executing a signal semaphore.

The structure of the chopstick is shown below – semaphore chopstick [5];

Initially the elements of the chopstick are initialized to 1 as the chopsticks are on the table and not picked up by a philosopher.

The structure of a random philosopher i is given as follows –

```

do
{
    wait( chopstick[i] );
    wait( chopstick[ (i+1) % 5] ); . . . EATING THE RICE .
    signal( chopstick[i] );
    signal( chopstick[ (i+1) % 5] ); . . THINKING
} while(1);

```

## Output :

```
Activities Terminal Nov 29 20:44
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx: ~/D
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$ gcc -pthread question6.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$ ./a.out
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 4 is Hungry
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 5 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
```

## **Question 7 : Readers Writers problem**

### **Code :-**

```
#include<iostream>
#include<pthread.h>
#include<unistd.h>

using namespace std;
class monitor
{
    private:
        int rcnt;
        int wcnt;
        int waitr;
        int waitw;
        pthread_cond_t canread;

        pthread_cond_t canwrite;
        pthread_mutex_t condlock;

    public:
        monitor()
        {
            rcnt = 0;
            wcnt = 0;
            waitr = 0;
            waitw = 0;
            pthread_cond_init(&canread, NULL);
            pthread_cond_init(&canwrite, NULL);
            pthread_mutex_init(&condlock, NULL);
        }

        void beginread(int i)
        {
            pthread_mutex_lock(&condlock);
            if(wcnt == 1 || waitw > 0)
            {
                waitr = waitr + 1;
                pthread_cond_wait(&canread, &condlock);
                waitr = waitr - 1;
            }

            rcnt = rcnt + 1;
            cout << "reader " << i << " is reading\n";
```

```

        pthread_mutex_unlock(&condlock);
        pthread_cond_broadcast(&canread);
    }

```

```

void endread(int i)
{
    pthread_mutex_lock(&condlock);
    if(--rcnt == 0)
        pthread_cond_signal(&canwrite);

    pthread_mutex_unlock(&condlock);
}

```

```

void beginwrite(int i)
{
    pthread_mutex_lock(&condlock);
    if(wcnt == 1 || rcnt > 0)
    {
        ++waitw;
        pthread_cond_wait(&canwrite, &condlock);
        --waitw;
    }

    wcnt = 1;
    cout << "writer " << i << " is writing\n";
    pthread_mutex_unlock(&condlock);
}

```

```

void endwrite(int i)
{
    pthread_mutex_lock(&condlock);
    wcnt = 0;    // if any readers are waiting, threads are unblocked

    if(waitr > 0)
        pthread_cond_signal(&canread);
    else
        pthread_cond_signal(&canwrite);

    pthread_mutex_unlock(&condlock);
}
} M; //object

```

```

void* reader(void* id);

```

```

void* writer(void* id);

```

```

int main()
{
    pthread_t r[5], w[5];
    int id[5];

    for(int i = 0; i < 5; i = i+1)
    {
        id[i] = i;
        pthread_create(&r[i], NULL, &reader, &id[i]);
        pthread_create(&w[i], NULL, &writer, &id[i]);
    }

    for(int i = 0; i < 5; i = i+1 )
    {
        pthread_join(r[i], NULL);
    }

    for(int i = 0; i < 5; i = i+1)
    {
        pthread_join(w[i], NULL);
    }
}

```

```

void* writer(void* id)
{
    int c = 0;
    int i = *(int*)id;

    while(c < 5)
    {
        usleep(1);
        M.beginwrite(i);
        M.endwrite(i);
        c++;
    }

    pthread_exit(0);
}

```

```

void* writer(void* id)
{
    int c = 0;
    int i = *(int*)id;

```

```

while(c < 5)
{
    usleep(1);
    M.beginwrite(i);
    M.endwrite(i);
    c++;
}

pthread_exit(0);
}

```

### **Explanation :-**

To solve this question, a writer should get exclusive access to an object i.e. when a writer is accessing the object, no reader or writer may access it. However, multiple readers can access the object at the same time. This can be implemented using semaphores.

The codes for the reader and writer process in the reader-writer problem are given as follows – Reader Process

The code that defines the reader process is given below –

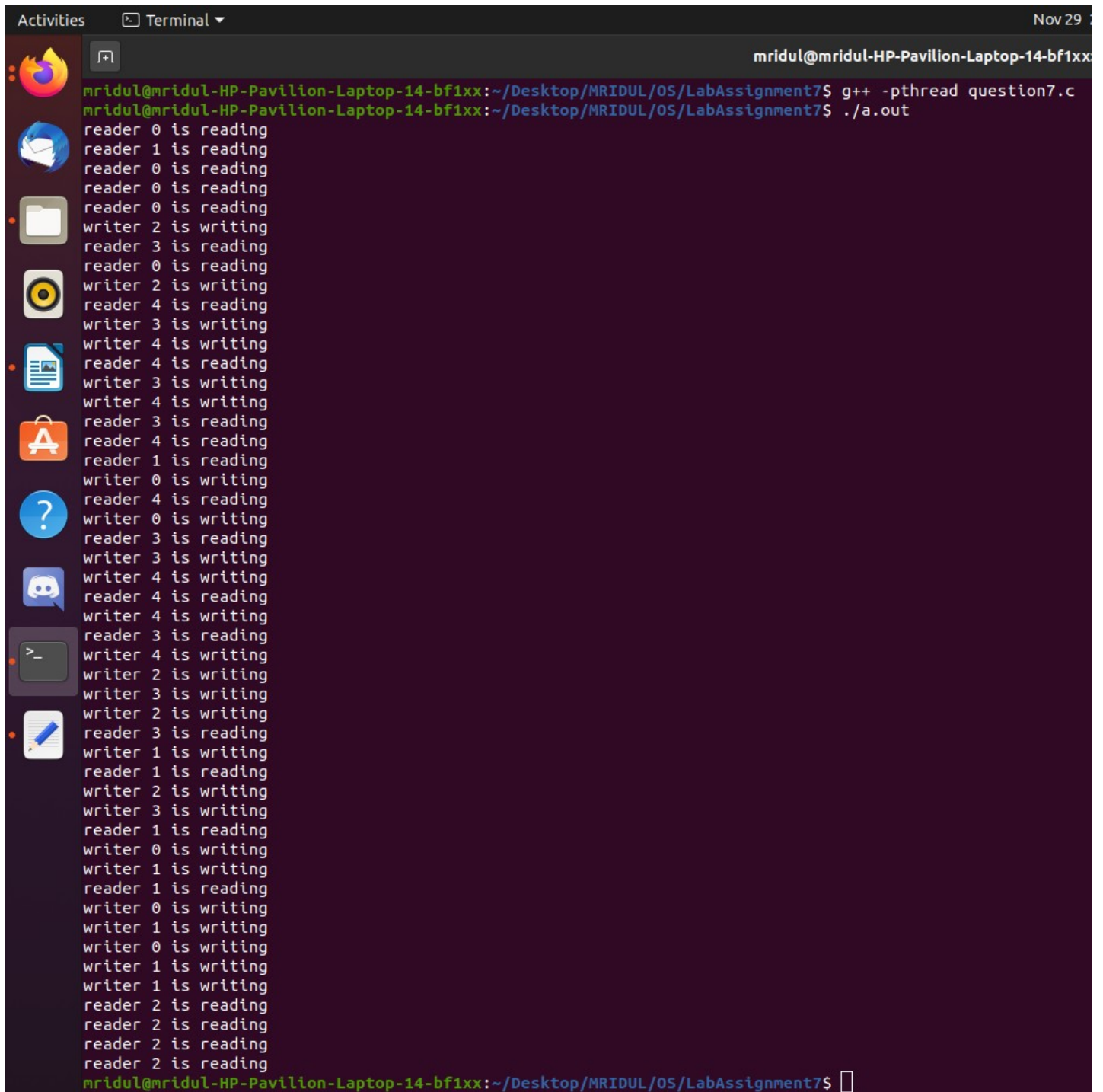
```

wait (mutex);
rc ++;
if(rc == 1)
    wait (wrt);
signal(mutex); ... READ THE OBJECT.
wait(mutex);
rc --;
if(rc == 0)
    signal (wrt);
signal(mutex);

```

In the above code, mutex and wrt are semaphores that are initialized to 1. Also, rc is a variable that is initialized to 0. The mutex semaphore ensures mutual exclusion and wrt handles the writing mechanism and is common to the reader and writer process code.

## Output :

A screenshot of a Linux terminal window. The title bar shows 'Activities', 'Terminal', and 'Nov 29'. The terminal prompt is 'mridul@mridul-HP-Pavilion-Laptop-14-bf1xx'. The user has executed 'g++ -pthread question7.c' and then './a.out'. The output consists of a sequence of messages: 'reader 0 is reading', 'reader 1 is reading', 'reader 0 is reading', 'reader 0 is reading', 'reader 0 is reading', 'writer 2 is writing', 'reader 3 is reading', 'reader 0 is reading', 'writer 2 is writing', 'reader 4 is reading', 'writer 3 is writing', 'writer 4 is writing', 'reader 4 is reading', 'writer 3 is writing', 'writer 4 is writing', 'reader 3 is reading', 'reader 4 is reading', 'reader 1 is reading', 'writer 0 is writing', 'reader 4 is reading', 'writer 0 is writing', 'reader 3 is reading', 'writer 3 is writing', 'writer 4 is writing', 'reader 4 is reading', 'writer 4 is writing', 'reader 3 is reading', 'writer 4 is writing', 'writer 2 is writing', 'writer 3 is writing', 'writer 2 is writing', 'reader 3 is reading', 'writer 1 is writing', 'reader 1 is reading', 'writer 2 is writing', 'writer 3 is writing', 'reader 1 is reading', 'writer 0 is writing', 'writer 1 is writing', 'reader 1 is reading', 'writer 0 is writing', 'writer 1 is writing', 'writer 0 is writing', 'writer 1 is writing', 'writer 1 is writing', 'reader 2 is reading', 'reader 2 is reading', 'reader 2 is reading', and 'reader 2 is reading'. The terminal ends with the prompt 'mridul@mridul-HP-Pavilion-Laptop-14-bf1xx\$' and a cursor.

```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$ g++ -pthread question7.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$ ./a.out
reader 0 is reading
reader 1 is reading
reader 0 is reading
reader 0 is reading
reader 0 is reading
writer 2 is writing
reader 3 is reading
reader 0 is reading
writer 2 is writing
reader 4 is reading
writer 3 is writing
writer 4 is writing
reader 4 is reading
writer 3 is writing
writer 4 is writing
reader 3 is reading
reader 4 is reading
reader 1 is reading
writer 0 is writing
reader 4 is reading
writer 0 is writing
reader 3 is reading
writer 3 is writing
writer 4 is writing
reader 4 is reading
writer 4 is writing
reader 3 is reading
writer 4 is writing
writer 2 is writing
writer 3 is writing
writer 2 is writing
reader 3 is reading
writer 1 is writing
reader 1 is reading
writer 2 is writing
writer 3 is writing
reader 1 is reading
writer 0 is writing
writer 1 is writing
reader 1 is reading
writer 0 is writing
writer 1 is writing
writer 0 is writing
writer 1 is writing
writer 1 is writing
reader 2 is reading
reader 2 is reading
reader 2 is reading
reader 2 is reading
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$
```

## Question 8 : Santa Claus Problem

### Code :-

```
#include<pthread.h>
#include<stdlib.h>
#include<assert.h>
```

```

#include<unistd.h>
#include<stdio.h>
#include<stdbool.h>
#include<semaphore.h>

pthread_t *CreateThread(void *(*f)(void *), void *a)
{
    static long thread;
    pthread_t* t;
    t = (pthread_t*)malloc(sizeof(pthread_t));
    assert(t != NULL);
    int ret = pthread_create(t, NULL, f, a);
    assert(ret == 0);return t;
}

static const int N_ELVES = 10;
static const int N_REINDEER = 9;
static int elves;
static int reindeer;
static sem_t santaSem;
static sem_t reindeerSem;
static sem_t elfTex;
static sem_t mutex;

void *SantaClaus(void *arg)
{
    printf("Santa Claus: Hoho, here I am\n");

    while(true)
    {
        sem_wait(&santaSem);
        sem_wait(&mutex);

        if(reindeer == N_REINDEER)
        {
            printf("Santa Claus: preparing sleigh\n");
            for (int r = 0; r < N_REINDEER; r = r+1)
                sem_post(&reindeerSem);

            printf("Santa Claus: make all kids in the world happy\n");
            reindeer = 0;
        }
        else if(elves == 3)
        {
            printf("Santa Claus: helping elves\n");

```



```

        }

        sem_post(&mutex);
    }

    return arg;
}

void *Reindeer(void *arg)
{
    int id = *((int *)arg);
    printf("This is reindeer %d\n", id);
    while (true)
    {
        sem_wait(&mutex);
        reindeer = reindeer + 1;
        if (reindeer == N_REINDEER)
            sem_post(&santaSem);

        sem_post(&mutex);
        sem_wait(&reindeerSem);

        printf("Reindeer %d getting hitched\n", id);
        sleep(20);
    }

    return arg;
}

void *Elve(void *arg)
{
    int id = *((int *)arg);
    printf("This is elve %d\n", id);

    while (true)
    {
        bool need_help = random() % 100 < 10;
        if(need_help)
        {
            sem_wait(&elfTex);
            sem_wait(&mutex);
            elves = elves + 1;

            if (elves == 3)
                sem_post(&santaSem);
        }
    }
}

```

```

        else
            sem_post(&elfTex);

        sem_post(&mutex);
        printf("Elve %d will get help from Santa Claus\n", id);
        sleep(10);
        sem_wait(&mutex);
        elves = elves - 1;

        if(elves == 0)
            sem_post(&elfTex);
        sem_post(&mutex);
    } // Do some work

    printf("Elve %d at work\n", id);
    sleep(2 + random() % 5);
}

return arg;
}

int main(int ac, char **av)
{
    elves = 0;
    reindeer = 0;
    sem_init(&santaSem, 0, 0);
    sem_init(&reindeerSem, 0, 0);
    sem_init(&elfTex, 0, 1);
    sem_init(&mutex, 0, 1);
    pthread_t *santa_claus = CreateThread(SantaClaus, 0);
    pthread_t *reindeers[N_REINDEER];

    for(int r = 0; r < N_REINDEER; r = r+1)
        reindeers[r] = CreateThread(Reindeer, (void *)r + 1);
    pthread_t *elves[N_ELVES];

    for (int e = 0; e < N_ELVES; e = e+1)
        elves[e] = CreateThread(Elve, (void *)e + 1);

    int ret = pthread_join(*santa_claus, NULL);
    assert(ret == 0);
}

```

## **Question 9** : Building H2O problem

### **Algorithm** :-

```
Oxygen code;
mutex.wait()
oxygen += 1
if hydrogen >= 2:
    hydroQueue.signal(2)
    hydrogen -= 2
    oxyQueue.signal()
    oxygen -= 1
else:
    mutex.signal()
    oxyQueue.wait()
    bond()
    barrier.wait()
    mutex.signal()
```

```
Hydrogen code;
mutex.wait()
hydrogen += 1
if hydrogen >= 2 and oxygen >= 1:
    hydroQueue.signal(2)
    hydrogen -= 2
    oxyQueue.signal()
    oxygen -= 1
else:
    mutex.signal()
    hydroQueue.wait()
    bond()
    barrier.wait()
```

### **Code** :-

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

sem_t smutex, oxyQueue, hydroQueue;
int oxygen=0; int hydrogen=0;
pthread_t oxyThread, hydroThread1, hydroThread2;

int bond();
```

```

void* oxyFn(void* arg);
void* hydroFn(void* arg);

int main()
{
    if(sem_init(&smutex,0,1) == -1)
    {
        perror("error initilalizing semaphore\n");
    }

    if(sem_init(&oxyQueue,0,0) == -1)
    {
        perror("error initilalizing semaphore\n");
    }

    if(sem_init(&hydroQueue,0,0) == -1)
    {
        perror("error initilalizing semaphore\n");
    }

    sleep(2);

    pthread_create(&oxyThread,0,oxyFn, NULL);
    pthread_create(&hydroThread1,0,hydroFn, NULL);
    pthread_create(&hydroThread2,0,hydroFn, NULL);

    for(;;);
}

int bond()
{
    static int i=0;
    i = i+1;

    if((i%3) == 0)
        printf("*** Molecule no. %d created**\n\n",i/3);

    sleep(2);
    return(0);
}

void* oxyFn(void* arg)
{
    while(1)
    {

```

```

sem_wait(&smutex);
oxygen+=1;

if(hydrogen >= 2)
{
    sem_post(&hydroQueue);
    sem_post(&hydroQueue);
    hydrogen-=2;
    sem_post(&oxyQueue);
    oxygen-=1;
}
else
{
    sem_post(&smutex);
}

sem_wait(&oxyQueue);
printf("Oxygen Bond\n");
bond();
sleep(3);
sem_post(&smutex);
}

void* hydroFn(void* arg)
{
    while(1)
    {
        sem_wait(&smutex);
        hydrogen = hydrogen + 1;

        if(hydrogen >= 2 && oxygen >= 1)
        {
            sem_post(&hydroQueue);
            sem_post(&hydroQueue);
            hydrogen = hydrogen - 2;
            sem_post(&oxyQueue);
            oxygen = oxygen - 1;
        }
        else
        {
            sem_post(&smutex);
        }

        sem_wait(&hydroQueue);
    }
}

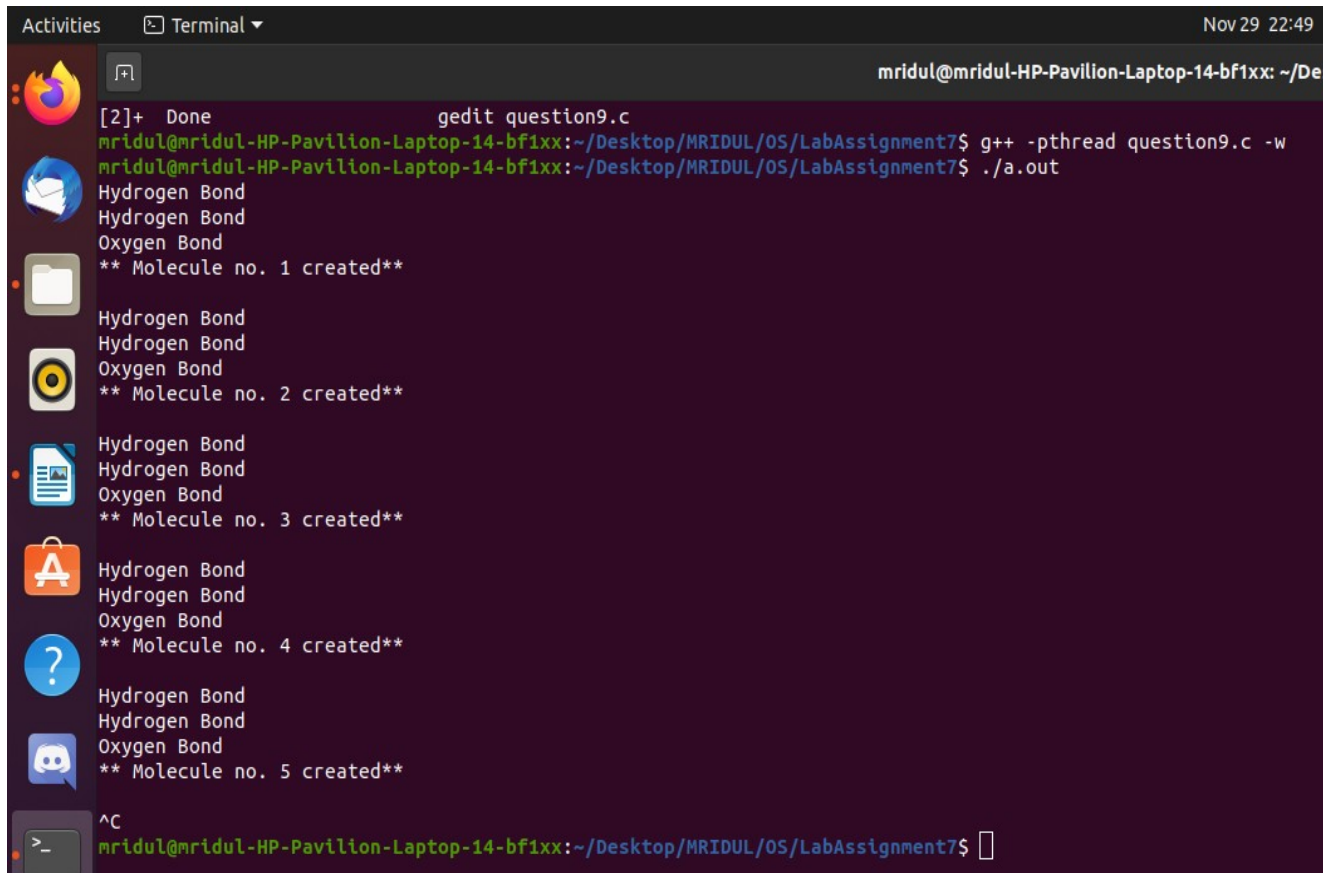
```

```

    printf("Hydrogen Bond\n");
    bond();
    sleep(3);
}
}

```

## Output :



```

Activities  Terminal ▾  Nov 29 22:49
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx: ~/De

[2]+  Done                  gedit question9.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$ g++ -pthread question9.c -w
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$ ./a.out
Hydrogen Bond
Hydrogen Bond
Oxygen Bond
** Molecule no. 1 created**
Hydrogen Bond
Hydrogen Bond
Oxygen Bond
** Molecule no. 2 created**
Hydrogen Bond
Hydrogen Bond
Oxygen Bond
** Molecule no. 3 created**
Hydrogen Bond
Hydrogen Bond
Oxygen Bond
** Molecule no. 4 created**
Hydrogen Bond
Hydrogen Bond
Oxygen Bond
** Molecule no. 5 created**
^C
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment7$ 

```