OPERATING SYSTEMS - COM301P

LAB ASSIGNMENT - 8

SYNCHRONIZATION AND SEMAPHORES

VIKNESH RAJARAMON

COE18B060

1) Implement the Dining Philosophers and Reader Writer Problem of Synchronization (test drive the codes discussed in the class).

DINING PHILOSOPHERS PROBLEM

LOGIC:

- We get the number of philosophers from the user.
- There are three states of each philosopher: **THINKING, HUNGRY and EATING**.
- There are two semaphores: **MUTEX** and a **semaphore array for the philosophers**.
- **Mutex** is used in such a way that **no 2 philosophers can pick up or put down a fork at the same time**.
- The **array** is used to **control the behaviour of each philosopher**.
- But, semaphores can result in **DEADLOCK**.
- To avoid **DEADLOCK**, we **assume that there is at least one philosopher who picks up the right fork** first while **other philosophers pick up the left fork first**.
- This **avoids DEADLOCK** because **one of the philosophers will not be able to pick both the forks**.
- Also, there is **no starvation** in the below implementation as **EATING time is limited for each philosopher**.
- **If 2 philosophers are HUNGRY** and are **fighting for the same fork**, then the **philosopher who was HUNGRY first** gets the fork.

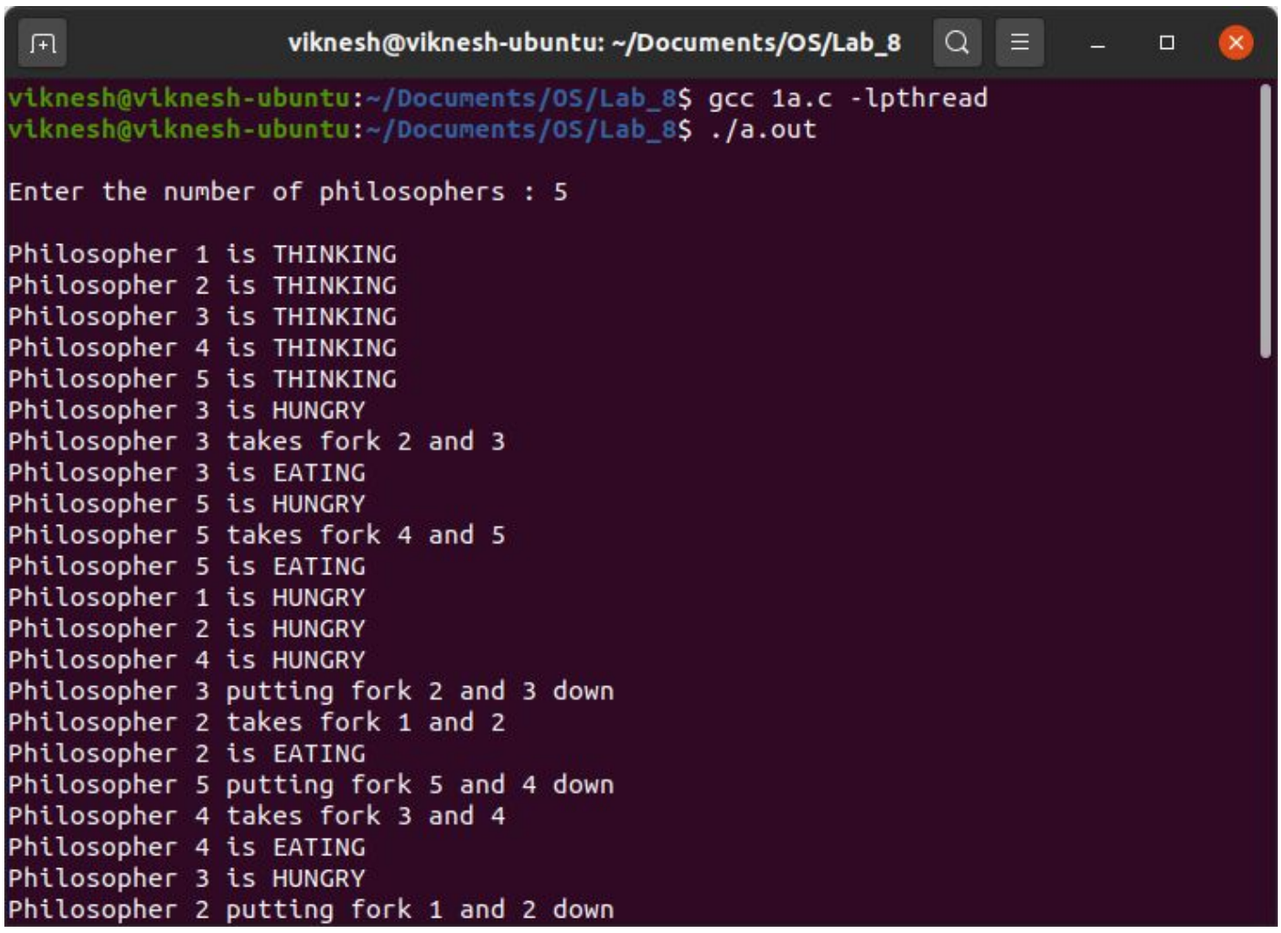
C CODE:

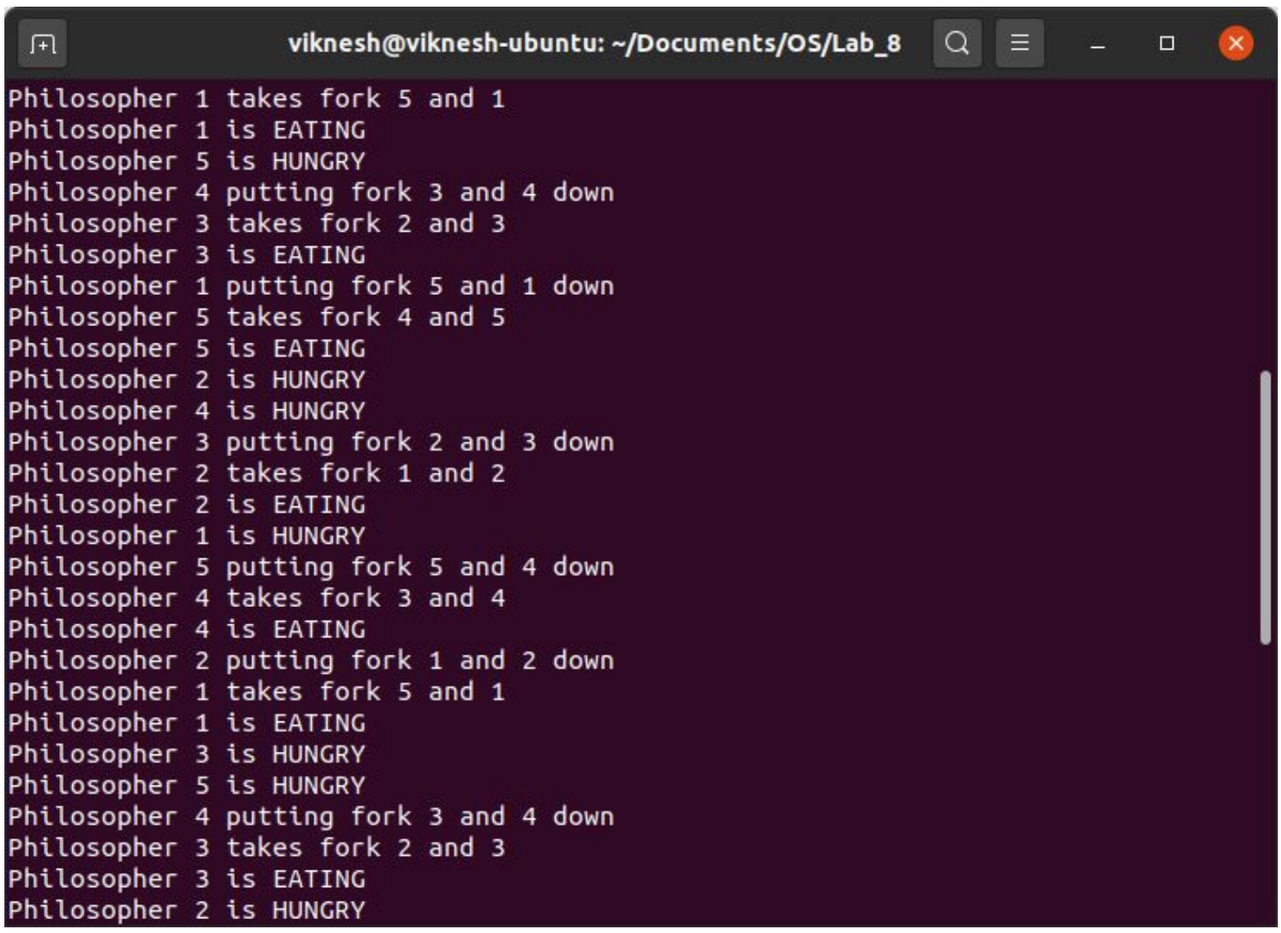
```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<semaphore.h>
#include<pthread.h>

#define THINKING 0
#define HUNGRY 1
#define EATING 2

#define LEFT (ph_num+4)%n
#define RIGHT (ph_num+1)%n

int n;
int *state;
int *phil_num;

sem_t mutex;
sem_t *S;

void test(int ph_num)
{
    if((state[ph_num]==HUNGRY) && (state[LEFT]!=EATING) && (state[RIGHT]!=EATING))
    {
            state[ph_num]=EATING;
            sleep(2);
```

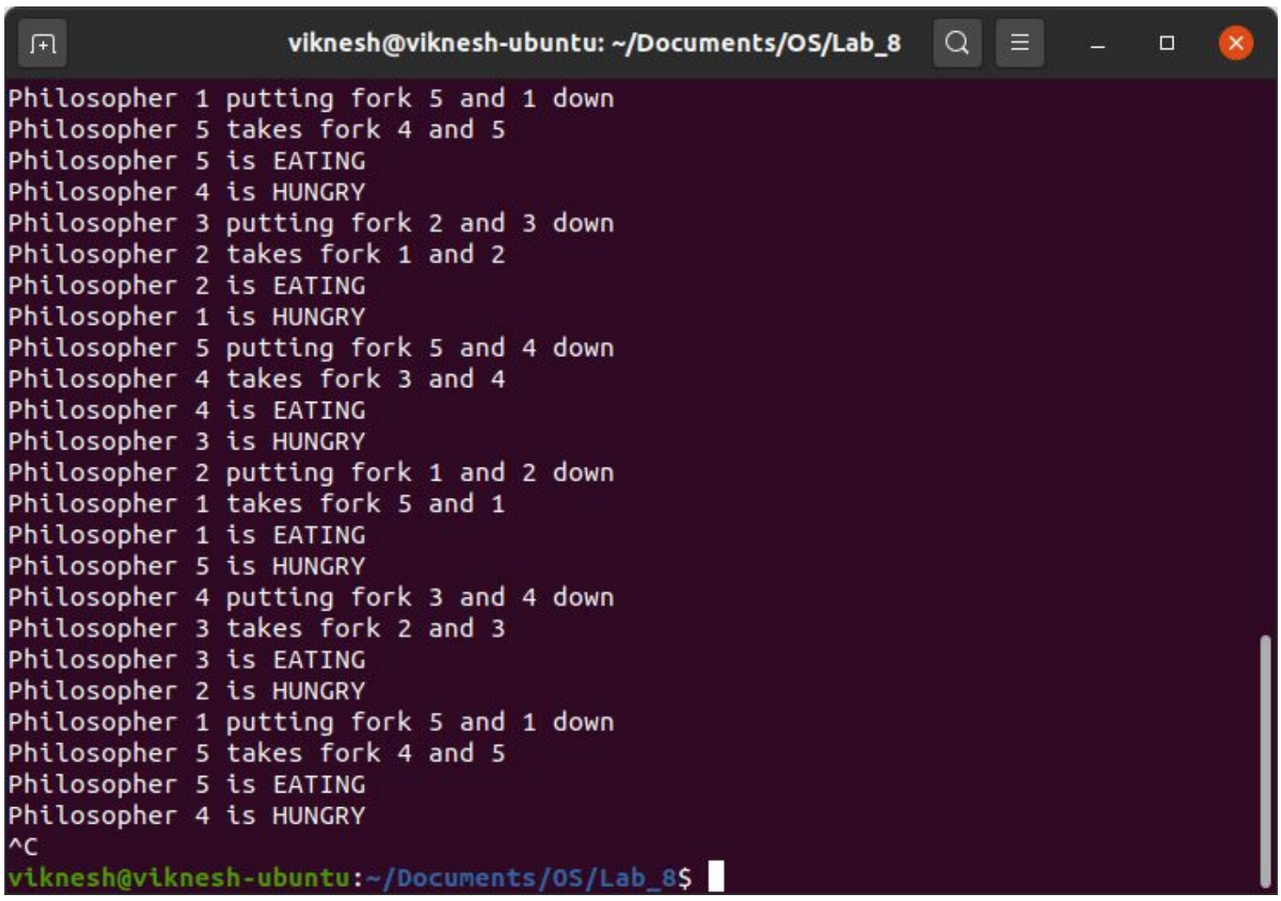
```
            printf("\nPhilosopher %d takes fork %d and %d",ph_num+1,LEFT+1,ph_num+1);
            printf("\nPhilosopher %d is EATING",ph_num+1);
            sem_post(&S[ph_num]);
    }
}

void take_fork_left(int ph_num)
{
    sem_wait(&mutex);
    state[ph_num]=HUNGRY;
    printf("\nPhilosopher %d is HUNGRY",ph_num+1);
    test(ph_num);
    sem_post(&mutex);
    sem_wait(&S[ph_num]);
    sleep(1);
}

void put_fork_left(int ph_num)
{
    sem_wait(&mutex);
    state[ph_num]=THINKING;
    printf("\nPhilosopher %d putting fork %d and %d down",ph_num+1,LEFT+1,ph_num+1);
    test(LEFT);
    test(RIGHT);
    sem_post(&mutex);
}

void *philosopher_left(void *num)
{
    while(1)
    {
            int *i=num;
            sleep(1);
            take_fork_left(*i);
            sleep(0);
            put_fork_left(*i);
    }
}

void take_fork_right(int ph_num)
{
    sem_wait(&mutex);
    state[ph_num]=HUNGRY;
    printf("\nPhilosopher %d is HUNGRY",ph_num+1);
    test(ph_num);
    sem_post(&mutex);
    sem_wait(&S[ph_num]);
    sleep(1);
}

void put_fork_right(int ph_num)
{
    sem_wait(&mutex);
    state[ph_num]=THINKING;
    printf("\nPhilosopher %d putting fork %d and %d down",ph_num+1,ph_num+1,LEFT+1);
    test(RIGHT);
    test(LEFT);
```

```
    sem_post(&mutex);
}

void *philosopher_right(void *num)
{
    while(1)
    {
            int *i=num;
            sleep(1);
            take_fork_right(*i);
            sleep(0);
            put_fork_right(*i);
    }
}

int main()
{
    printf("\nEnter the number of philosophers : ");
    scanf("%d",&n);
    sem_init(&mutex,0,1);

    pthread_t tid[n];
    S=(sem_t *)malloc(sizeof(sem_t)*n);;
    phil_num=(int *)malloc(sizeof(int)*n);
    state=(int *)malloc(sizeof(int)*n);

    for(int i=0;i<n;++i)
    {
            sem_init(&S[i],0,0);
            phil_num[i]=i;
    }

    for(int i=0;i<n-1;++i)
    {
            pthread_create(&tid[i],NULL,philosopher_left,&phil_num[i]);
            printf("\nPhilosopher %d is THINKING",i+1);
    }
    pthread_create(&tid[n-1],NULL,philosopher_right,&phil_num[n-1]);
    printf("\nPhilosopher %d is THINKING",n);
    for(int i=0;i<n;++i)
    {
            pthread_join(tid[i],NULL);
    }
    return 0;
}
```

OUTPUT:


viknesh@viknesh-ubuntu: ~/Documents/OS/Lab_8
viknesh@viknesh-ubuntu:~/Documents/OS/Lab_8$ gcc 1a.c -lpthread
viknesh@viknesh-ubuntu:~/Documents/OS/Lab_8$ ./a.out
Enter the number of philosophers : 5
Philosopher 1 is THINKING
Philosopher 2 is THINKING
Philosopher 3 is THINKING
Philosopher 4 is THINKING
Philosopher 5 is THINKING
Philosopher 3 is HUNGRY
Philosopher 3 takes fork 2 and 3
Philosopher 3 is EATING
Philosopher 5 is HUNGRY
Philosopher 5 takes fork 4 and 5
Philosopher 5 is EATING
Philosopher 1 is HUNGRY
Philosopher 2 is HUNGRY
Philosopher 4 is HUNGRY
Philosopher 3 putting fork 2 and 3 down
Philosopher 2 takes fork 1 and 2
Philosopher 2 is EATING
Philosopher 5 putting fork 5 and 4 down
Philosopher 4 takes fork 3 and 4
Philosopher 4 is EATING
Philosopher 3 is HUNGRY
Philosopher 2 putting fork 1 and 2 down


viknesh@viknesh-ubuntu: ~/Documents/OS/Lab_8
Philosopher 1 takes fork 5 and 1
Philosopher 1 is EATING
Philosopher 5 is HUNGRY
Philosopher 4 putting fork 3 and 4 down
Philosopher 3 takes fork 2 and 3
Philosopher 3 is EATING
Philosopher 1 putting fork 5 and 1 down
Philosopher 5 takes fork 4 and 5
Philosopher 5 is EATING
Philosopher 2 is HUNGRY
Philosopher 4 is HUNGRY
Philosopher 3 putting fork 2 and 3 down
Philosopher 2 takes fork 1 and 2
Philosopher 2 is EATING
Philosopher 1 is HUNGRY
Philosopher 5 putting fork 5 and 4 down
Philosopher 4 takes fork 3 and 4
Philosopher 4 is EATING
Philosopher 2 putting fork 1 and 2 down
Philosopher 1 takes fork 5 and 1
Philosopher 1 is EATING
Philosopher 3 is HUNGRY
Philosopher 5 is HUNGRY
Philosopher 4 putting fork 3 and 4 down
Philosopher 3 takes fork 2 and 3
Philosopher 3 is EATING
Philosopher 2 is HUNGRY



READER WRITER PROBLEM

LOGIC:

- We get the **number of READERS, WRITERS** and **number of times each reader or writer should enter CRITICAL SECTION** from the user.
- **Any number of READERS** can be in the **CRITICAL SECTION simultaneously**.
- **WRITERS** must have **exclusive access to the CRITICAL SECTION**.
- There are three semaphores: **MUTEX, WRITEBLOCK** and **TURNS_TILE**.
- **Mutex** is used to **block and queue the READERS coming in when a WRITER is in the CRITICAL SECTION**.
- **Writeblock** is used to **block and queue the WRITERS coming in when READERS are in the CRITICAL SECTION**.
- For this problem, **DEADLOCK is not possible** since **at least one of the threads is being processed**.
- But, it can result in **STARVATION**. **STARVATION** occurs **when one or more readers enter the critical section before the last of the current readers exit**.
- To **avoid STARVATION**, we **consider another semaphore - Turns_tile**. This semaphore **blocks any number of readers entering while a writer is queued**.

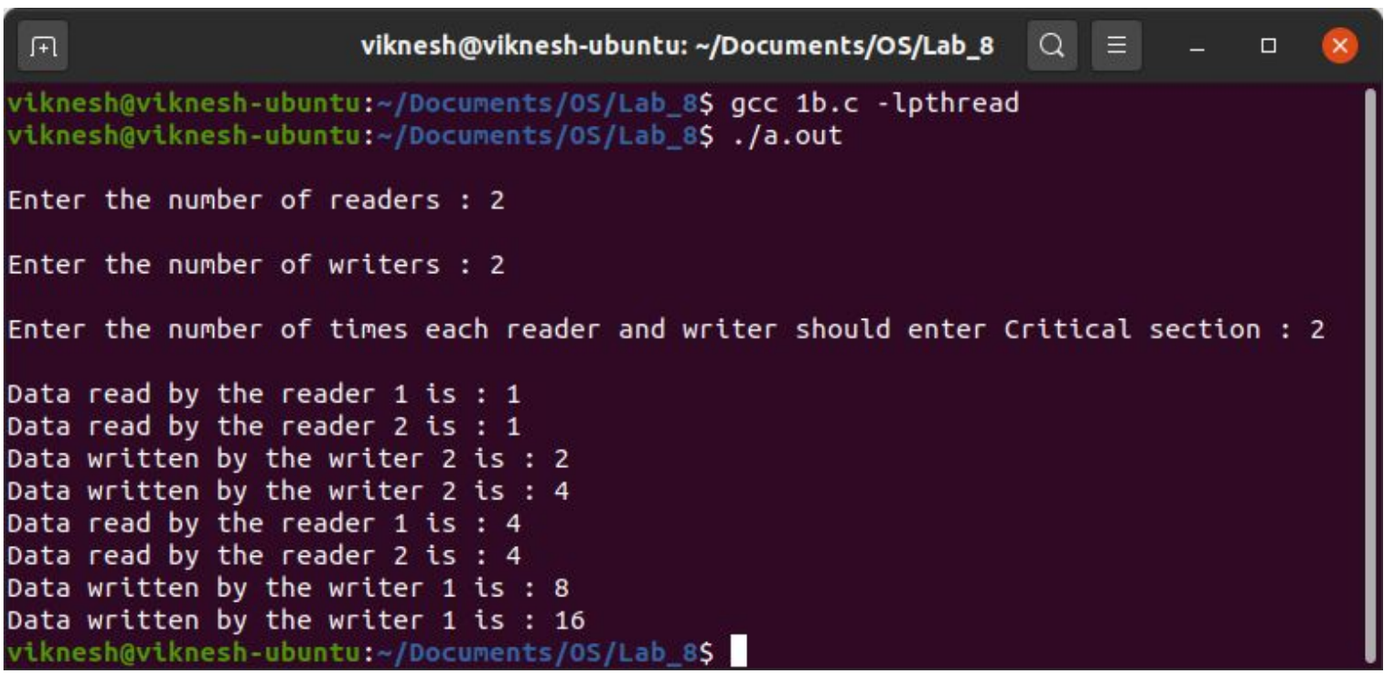
C CODE:

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

```
int no_of_readers,no_of_writers;
int data=1,rcount=0;
int count;

sem_t mutex,write_block,turns_tile;

void *reader(void *arg)
{
    for(int i=0;i<count;++i)
    {
            int f=*((int *)arg);
            sem_wait(&turns_tile);
            sem_post(&turns_tile);
            sem_wait(&mutex);
            rcount+=1;
            if(rcount==1)
            {
                    sem_wait(&write_block);
            }
            sem_post(&mutex);
            printf("\nData read by the reader %d is : %d",f,data);
            sleep(1);
            sem_wait(&mutex);
            rcount-=1;
            if(rcount==0)
            {
                    sem_post(&write_block);
            }
            sem_post(&mutex);
    }
}

void *writer(void *arg)
{
    for(int i=0;i<count;++i)
    {
            int f=*((int *)arg);
            sem_wait(&turns_tile);
            sem_wait(&write_block);
            data=data*2;
            printf("\nData written by the writer %d is : %d",f,data);
            sleep(1);
            sem_post(&turns_tile);
            sem_post(&write_block);
    }
}

int main()
{
    sem_init(&mutex,0,1);
    sem_init(&write_block,0,1);
    sem_init(&turns_tile,0,1);
    printf("\nEnter the number of readers : ");
    scanf("%d",&no_of_readers);
    printf("\nEnter the number of writers : ");
    scanf("%d",&no_of_writers);
```

```
    printf("\nEnter the number of times each reader and writer should enter Critical section : ");
    scanf("%d",&count);
    pthread_t read[no_of_readers];
    pthread_t write[no_of_writers];
    int r[no_of_readers];
    int w[no_of_writers];

    for(int i=0;i<no_of_readers;++i)
    {
        r[i]=i+1;
        pthread_create(&read[i],NULL,reader,&r[i]);
    }

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

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

    for(int i=0;i<no_of_readers;++i)
    {
        pthread_join(read[i],NULL);
    }
    printf("\n");

    return 0;
}
```

OUTPUT:



```
viknesh@viknesh-ubuntu: ~/Documents/OS/Lab_8
viknesh@viknesh-ubuntu:~/Documents/OS/Lab_8$ gcc 1b.c -lpthread
viknesh@viknesh-ubuntu:~/Documents/OS/Lab_8$ ./a.out

Enter the number of readers : 10

Enter the number of writers : 5

Enter the number of times each reader and writer should enter Critical section : 3

Data read by the reader 2 is : 1
Data read by the reader 3 is : 1
Data read by the reader 5 is : 1
Data read by the reader 6 is : 1
Data read by the reader 4 is : 1
Data read by the reader 7 is : 1
Data read by the reader 8 is : 1
Data read by the reader 9 is : 1
Data written by the writer 2 is : 2
Data written by the writer 2 is : 4
Data written by the writer 2 is : 8
Data written by the writer 3 is : 16
Data written by the writer 3 is : 32
Data written by the writer 3 is : 64
Data written by the writer 5 is : 128
Data read by the reader 2 is : 128
Data written by the writer 5 is : 256
Data read by the reader 5 is : 256
```

```
viknesh@viknesh-ubuntu: ~/Documents/OS/Lab_8
Data read by the reader 6 is : 256
Data written by the writer 5 is : 512
Data read by the reader 8 is : 512
Data read by the reader 9 is : 512
Data read by the reader 7 is : 512
Data read by the reader 10 is : 512
Data written by the writer 1 is : 1024
Data written by the writer 1 is : 2048
Data written by the writer 1 is : 4096
Data read by the reader 3 is : 4096
Data read by the reader 2 is : 4096
Data read by the reader 4 is : 4096
Data read by the reader 6 is : 4096
Data read by the reader 5 is : 4096
Data read by the reader 7 is : 4096
Data read by the reader 8 is : 4096
Data read by the reader 9 is : 4096
Data read by the reader 10 is : 4096
Data written by the writer 4 is : 8192
Data written by the writer 4 is : 16384
Data written by the writer 4 is : 32768
Data read by the reader 3 is : 32768
Data read by the reader 10 is : 32768
Data read by the reader 1 is : 32768
Data read by the reader 4 is : 32768
Data read by the reader 1 is : 32768
Data read by the reader 1 is : 32768
viknesh@viknesh-ubuntu:~/Documents/OS/Lab_8$
```

2) Choose any 2 of the following problems whose details are available in the Downy Book on Semaphores (attached) and implement semaphores based solutions to the same.

   a) Santa Claus Problem
   b) H2O Problem
   c) Baboon Crossing Problem
   d) Dining Hall Problem
   e) Senate Bus Problem

SANTA CLAUS PROBLEM

LOGIC:

- We get the **number of REINDEERS** and **ELVES** from the user.
- If **all the reindeers arrive**, then **Santa must prepare the sleigh** and **all the reindeers must get hitched**.
- After the **third elf arrives**, Santa must **help the three elves concurrently that have approached him**.
- The three elves must have finished getting help from Santa before the next elf enters.
- Elves and reindeer are counters protected by mutex. Elves and reindeer get mutex to modify the value while Santa gets it to check them.
- **Santa keeps sleeping** (waits on santaSem) until either **an elf** or **a reindeer signals him**.
- Reindeer waits on reindeerSem until Santa signals them to get hitched. The last reindeer releases the lock.
- Elves use elfTex to prevent additional elves from entering while three elves are getting help from Santa. The third elf releases the lock.
- Elves use elfTex to prevent additional elves from entering while three elves are getting help from Santa. The third elf releases the lock.

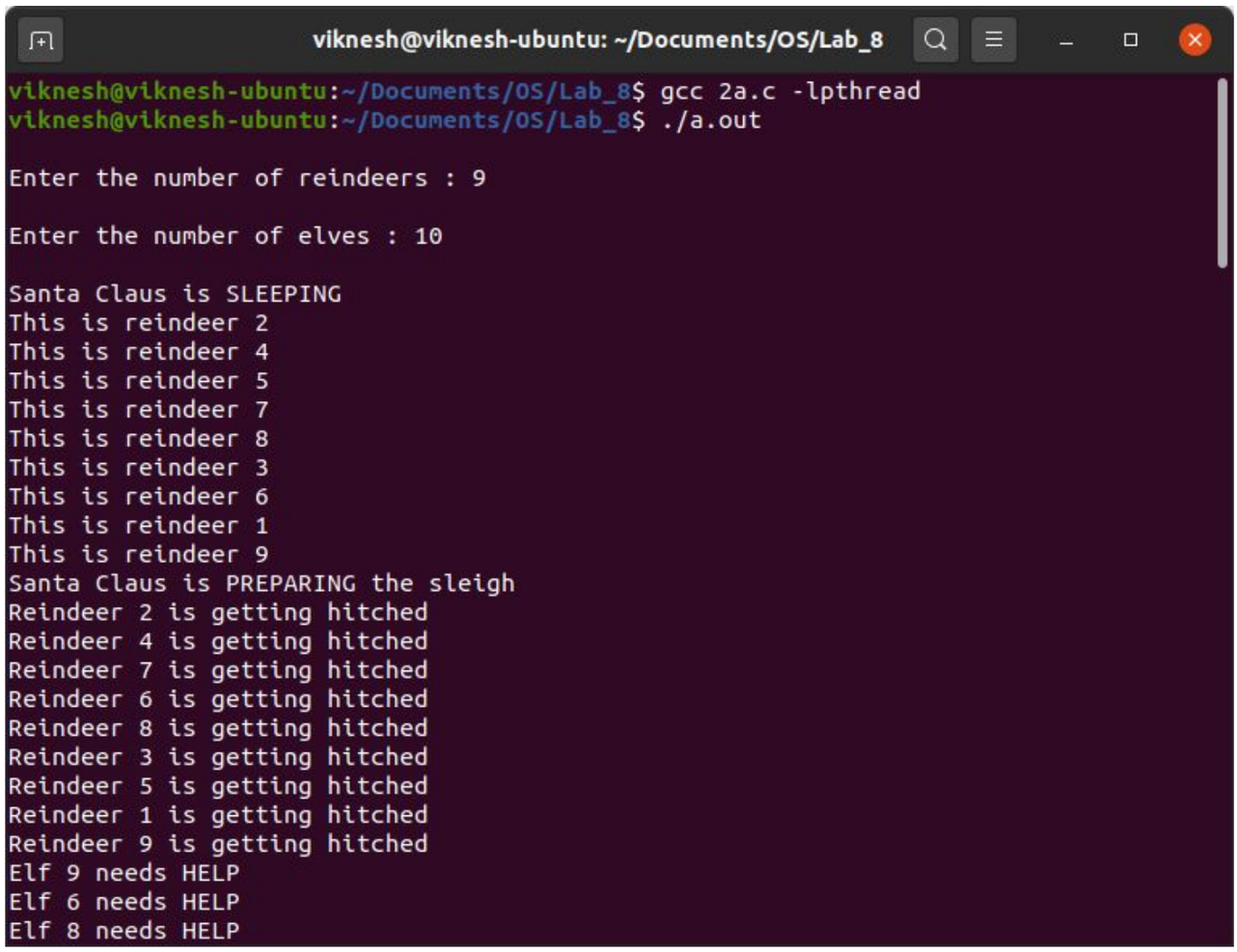
C CODE:

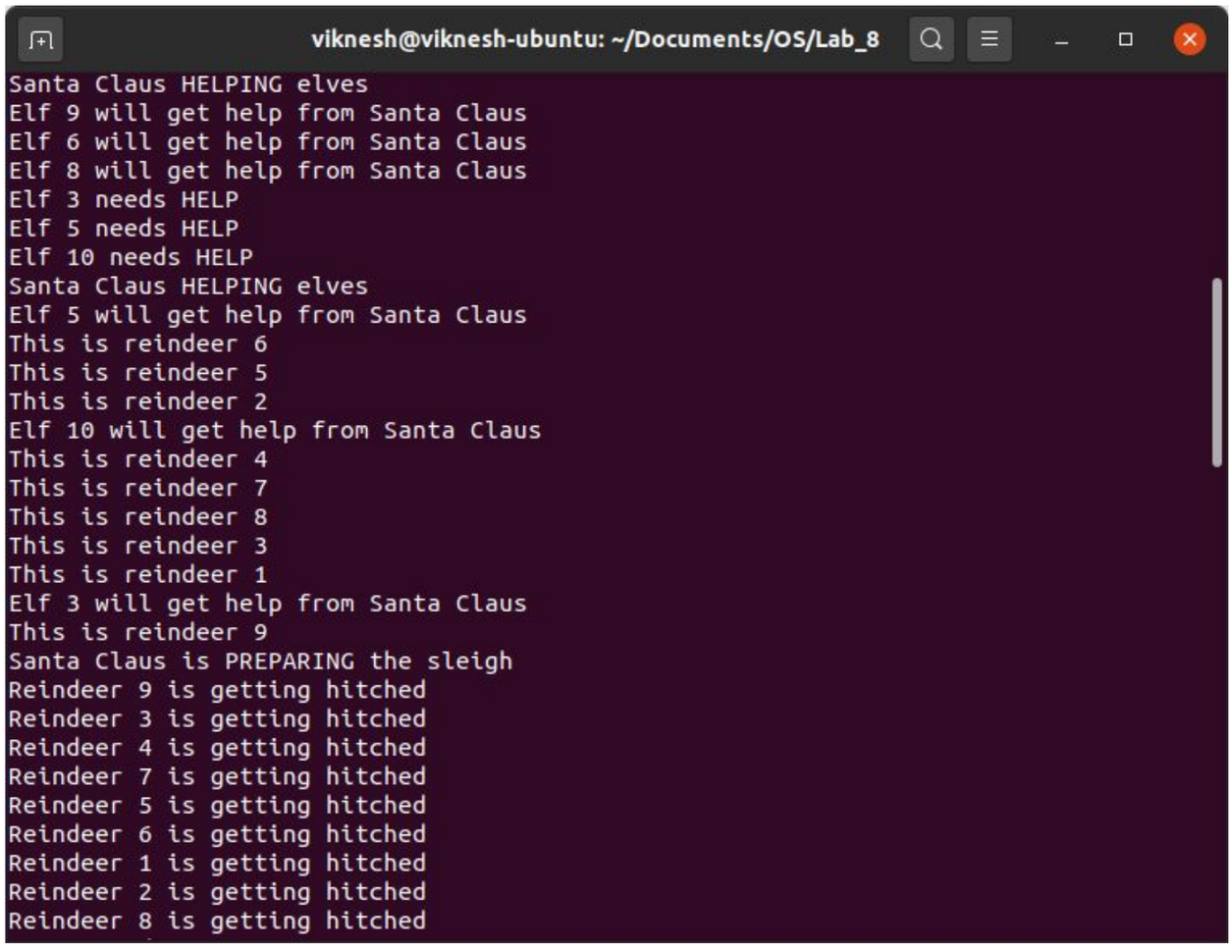
```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>

int N_ELVES,N_REINDEER;
int elves=0,reindeer=0;
sem_t mutex,santaSem,reindeerSem,elfTex;

void *Santa_Claus()
{
    printf("\nSanta Claus is SLEEPING");
    while(1)
    {
            sem_wait(&santaSem);
            sem_wait(&mutex);
            if(reindeer==N_REINDEER)
```

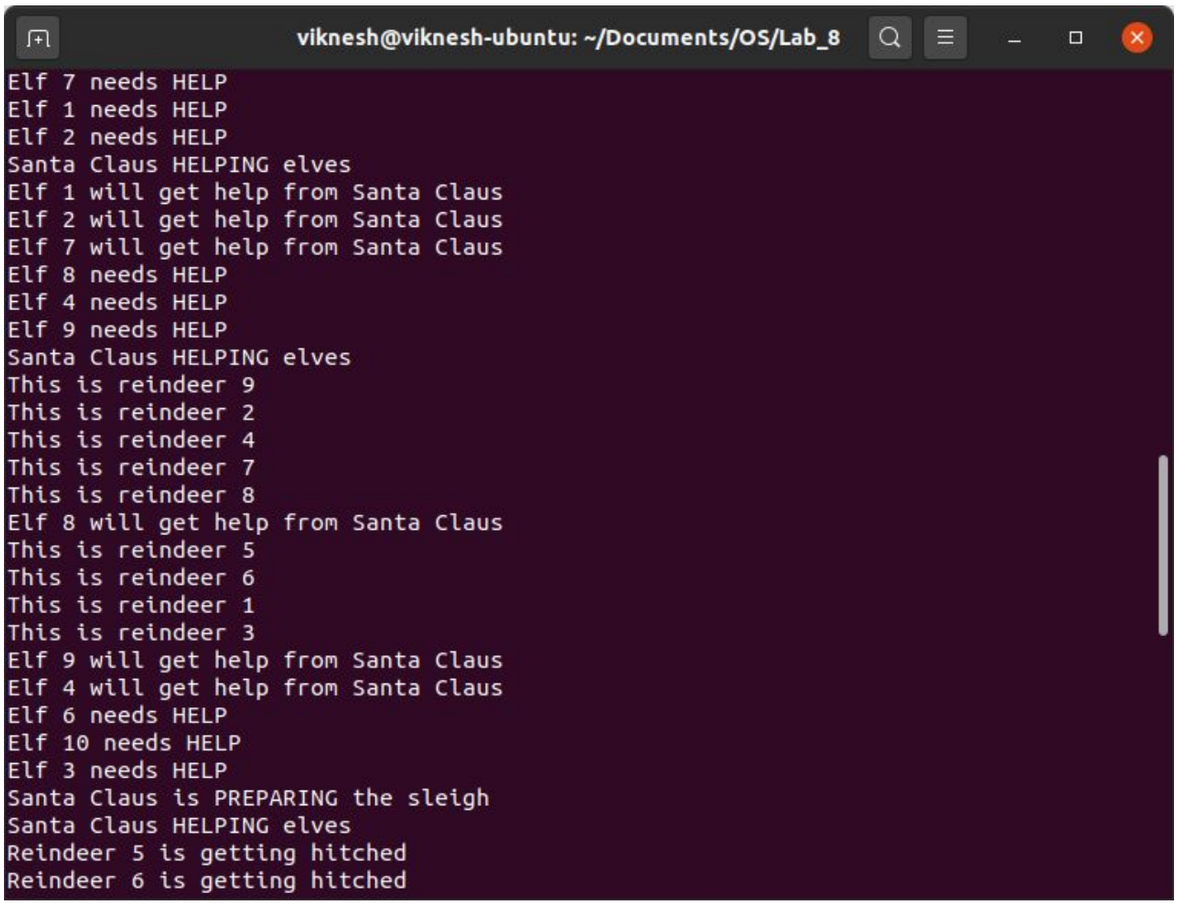
```
		{
			printf("\nSanta Claus is PREPARING the sleigh");
			for(int i=0;i<N_REINDEER;++i)
			{
				sem_post(&reindeerSem);
			}
			reindeer=0;
		}
		else if(elves==3)
		{
			printf("\nSanta Claus HELPING elves");
		}
		sem_post(&mutex);
	}
}

void *Reindeer(void *arg)
{
	int id=*((int *)arg);
	while(1)
	{
		sem_wait(&mutex);
		printf("\nThis is reindeer %d",id);
		++reindeer;
		if(reindeer==N_REINDEER)
		{
			sem_post(&santaSem);
		}
		sem_post(&mutex);
		sem_wait(&reindeerSem);
		printf("\nReindeer %d is getting hitched",id);
		sleep(20);
	}
}

void *Elves(void *arg)
{
	int id=*((int *)arg);
	while(1)
	{
		bool need_help=random()%100<50;
		if(need_help)
		{
			sem_wait(&elfTex);
			sem_wait(&mutex);
			printf("\nElf %d needs HELP",id);
			++elves;
			if(elves==3)
			{
				sem_post(&santaSem);
			}
			else
			{
				sem_post(&elfTex);
			}
			sem_post(&mutex);
			sleep(10);
```

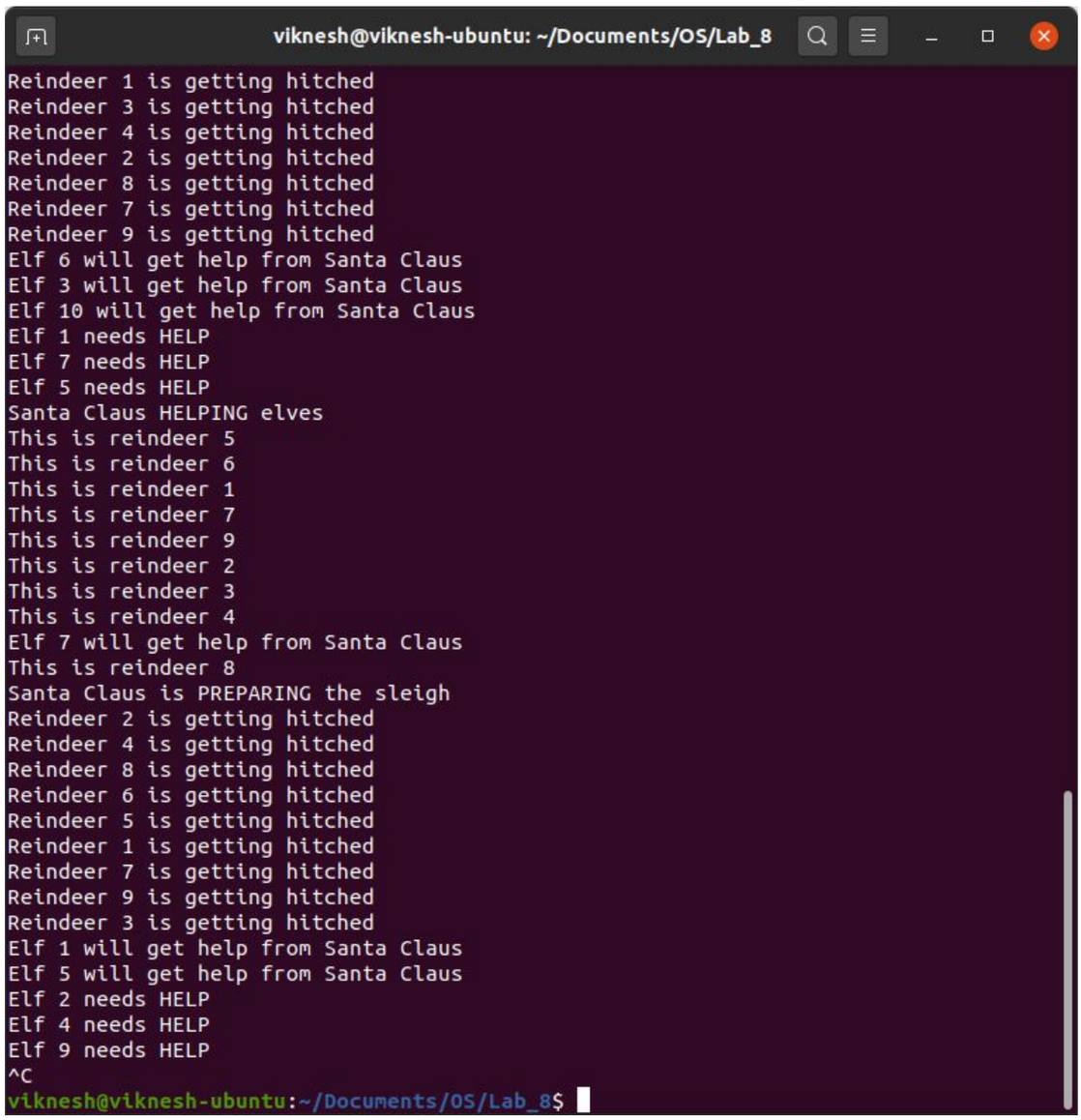
```
                sem_wait(&mutex);
                printf("\nElf %d will get help from Santa Claus",id);
                --elves;
                if(elves==0)
                {
                        sem_post(&elfTex);
                }
                sem_post(&mutex);
        }
        sleep(2+random()%5);
    }
}

int main()
{
    sem_init(&mutex,0,1);
    sem_init(&santaSem,0,0);
    sem_init(&reindeerSem,0,0);
    sem_init(&elfTex,0,1);

    printf("\nEnter the number of reindeers : ");
    scanf("%d",&N_REINDEER);
    printf("\nEnter the number of elves : ");
    scanf("%d",&N_ELVES);

    int r[N_REINDEER],e[N_ELVES];
    pthread_t santa_claus,reindeers[N_REINDEER],elf[N_ELVES];
    pthread_create(&santa_claus,NULL,Santa_Claus,NULL);

    for(int i=0;i<N_REINDEER;++i)
    {
        r[i]=i+1;
        pthread_create(&reindeers[i],NULL,Reindeer,&r[i]);
    }

    for(int i=0;i<N_ELVES;++i)
    {
        e[i]=i+1;
        pthread_create(&elf[i],NULL,Elves,&e[i]);
    }

    pthread_join(santa_claus,NULL);

    for(int i=0;i<N_REINDEER;++i)
    {
        pthread_join(reindeers[i],NULL);
    }

    for(int i=0;i<N_ELVES;++i)
    {
        pthread_join(elf[i],NULL);
    }

    return 0;
}
```

OUTPUT:


viknesh@viknesh-ubuntu: ~/Documents/OS/Lab_8
viknesh@viknesh-ubuntu:~/Documents/OS/Lab_8$ gcc 2a.c -lpthread
viknesh@viknesh-ubuntu:~/Documents/OS/Lab_8$ ./a.out
Enter the number of reindeers : 9
Enter the number of elves : 10
Santa Claus is SLEEPING
This is reindeer 2
This is reindeer 4
This is reindeer 5
This is reindeer 7
This is reindeer 8
This is reindeer 3
This is reindeer 6
This is reindeer 1
This is reindeer 9
Santa Claus is PREPARING the sleigh
Reindeer 2 is getting hitched
Reindeer 4 is getting hitched
Reindeer 7 is getting hitched
Reindeer 6 is getting hitched
Reindeer 8 is getting hitched
Reindeer 3 is getting hitched
Reindeer 5 is getting hitched
Reindeer 1 is getting hitched
Reindeer 9 is getting hitched
Elf 9 needs HELP
Elf 6 needs HELP
Elf 8 needs HELP


viknesh@viknesh-ubuntu: ~/Documents/OS/Lab_8
Santa Claus HELPING elves
Elf 9 will get help from Santa Claus
Elf 6 will get help from Santa Claus
Elf 8 will get help from Santa Claus
Elf 3 needs HELP
Elf 5 needs HELP
Elf 10 needs HELP
Santa Claus HELPING elves
Elf 5 will get help from Santa Claus
This is reindeer 6
This is reindeer 5
This is reindeer 2
Elf 10 will get help from Santa Claus
This is reindeer 4
This is reindeer 7
This is reindeer 8
This is reindeer 3
This is reindeer 1
Elf 3 will get help from Santa Claus
This is reindeer 9
Santa Claus is PREPARING the sleigh
Reindeer 9 is getting hitched
Reindeer 3 is getting hitched
Reindeer 4 is getting hitched
Reindeer 7 is getting hitched
Reindeer 5 is getting hitched
Reindeer 6 is getting hitched
Reindeer 1 is getting hitched
Reindeer 2 is getting hitched
Reindeer 8 is getting hitched


viknesh@viknesh-ubuntu: ~/Documents/OS/Lab_8
Elf 7 needs HELP
Elf 1 needs HELP
Elf 2 needs HELP
Santa Claus HELPING elves
Elf 1 will get help from Santa Claus
Elf 2 will get help from Santa Claus
Elf 7 will get help from Santa Claus
Elf 8 needs HELP
Elf 4 needs HELP
Elf 9 needs HELP
Santa Claus HELPING elves
This is reindeer 9
This is reindeer 2
This is reindeer 4
This is reindeer 7
This is reindeer 8
Elf 8 will get help from Santa Claus
This is reindeer 5
This is reindeer 6
This is reindeer 1
This is reindeer 3
Elf 9 will get help from Santa Claus
Elf 4 will get help from Santa Claus
Elf 6 needs HELP
Elf 10 needs HELP
Elf 3 needs HELP
Santa Claus is PREPARING the sleigh
Santa Claus HELPING elves
Reindeer 5 is getting hitched
Reindeer 6 is getting hitched


viknesh@viknesh-ubuntu: ~/Documents/OS/Lab_8
Reindeer 1 is getting hitched
Reindeer 3 is getting hitched
Reindeer 4 is getting hitched
Reindeer 2 is getting hitched
Reindeer 8 is getting hitched
Reindeer 7 is getting hitched
Reindeer 9 is getting hitched
Elf 6 will get help from Santa Claus
Elf 3 will get help from Santa Claus
Elf 10 will get help from Santa Claus
Elf 1 needs HELP
Elf 7 needs HELP
Elf 5 needs HELP
Santa Claus HELPING elves
This is reindeer 5
This is reindeer 6
This is reindeer 1
This is reindeer 7
This is reindeer 9
This is reindeer 2
This is reindeer 3
This is reindeer 4
Elf 7 will get help from Santa Claus
This is reindeer 8
Santa Claus is PREPARING the sleigh
Reindeer 2 is getting hitched
Reindeer 4 is getting hitched
Reindeer 8 is getting hitched
Reindeer 6 is getting hitched
Reindeer 5 is getting hitched
Reindeer 1 is getting hitched
Reindeer 7 is getting hitched
Reindeer 9 is getting hitched
Reindeer 3 is getting hitched
Elf 1 will get help from Santa Claus
Elf 5 will get help from Santa Claus
Elf 2 needs HELP
Elf 4 needs HELP
Elf 9 needs HELP
^C
viknesh@viknesh-ubuntu:~/Documents/OS/Lab_8$

H2O PROBLEM

LOGIC:

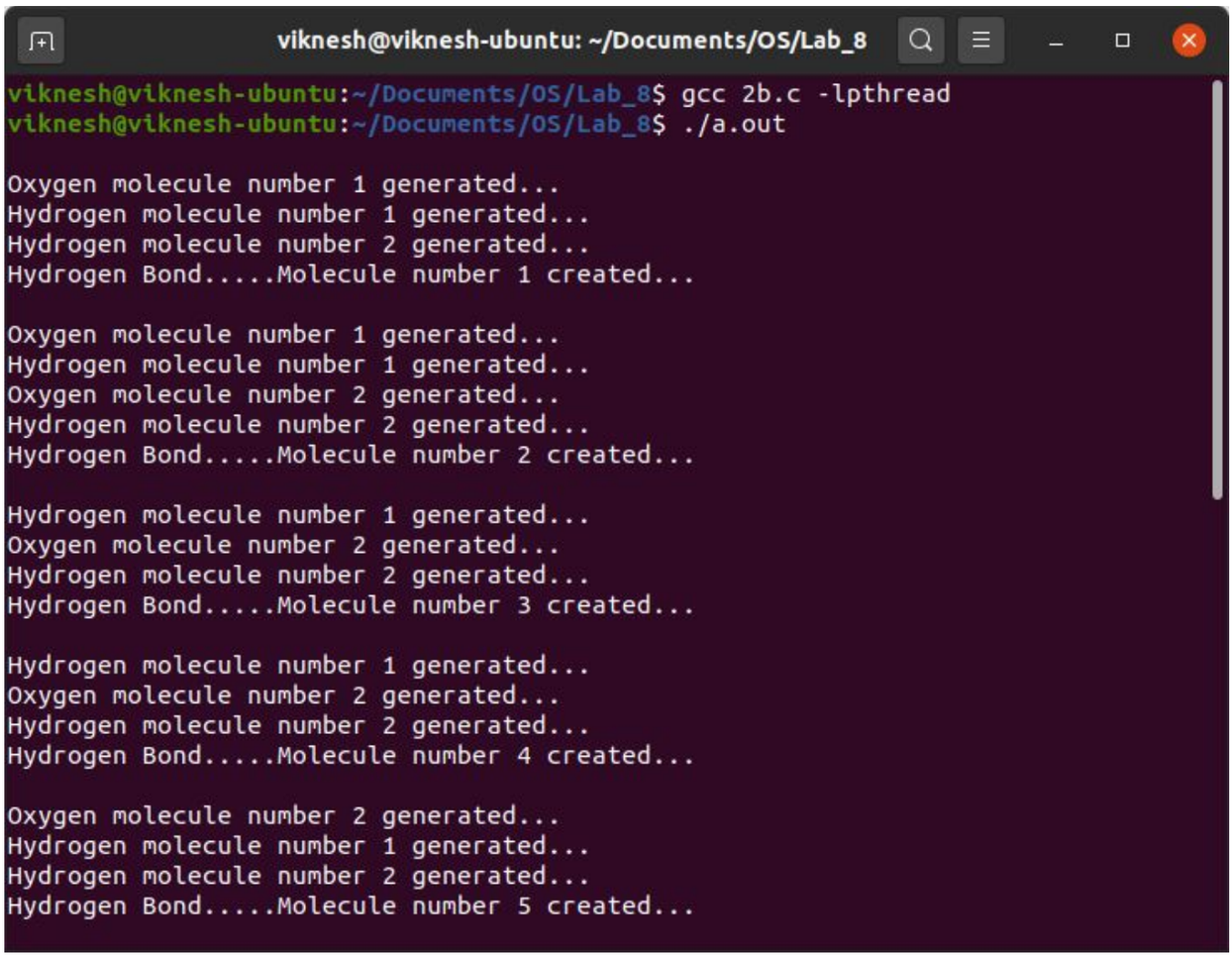
- There are two kinds of threads - Oxygen and Hydrogen.
- We **create a barrier** that makes **each thread wait until a complete molecule is formed**.
- Once the **thread crosses the barrier**, it should **create the water molecule**.
- If an oxygen thread arrives at the barrier when no hydrogen thread is present, it has to wait for two hydrogen threads.
- If a hydrogen thread arrives at the barrier when no other threads are present, it has to wait for one oxygen thread and another hydrogen thread.
- **Oxygen** and **hydrogen** are **counters protected by mutex**.
- **OxyQueue** is the **semaphore oxygen threads wait on**.
- **HydroQueue** is the **semaphore hydrogen threads wait on**.

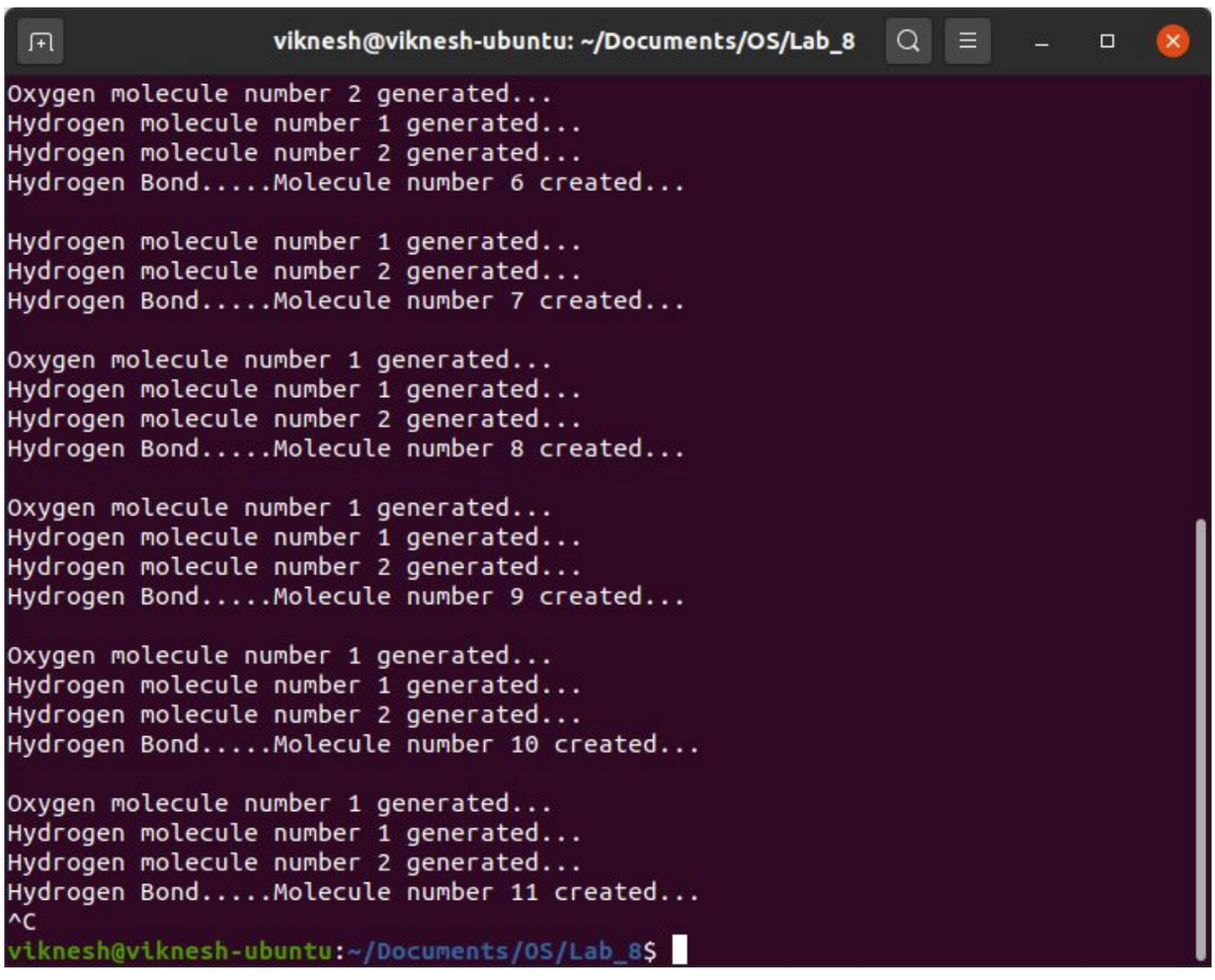
C CODE:

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

int hydrogen=0,oxygen=0;
int count=0;
sem_t mutex,hydroQueue,oxyQueue;

void *Hydrogen()
{
    while(1)
    {
            sem_wait(&mutex);
            hydrogen+=1;
            printf("\nHydrogen molecule number %d generated...",hydrogen);
            if((hydrogen>=2) &&(oxygen>=1))
            {
                    ++count;
                    printf("\nHydrogen Bond.....Molecule number %d created...\n",count);
                    sem_post(&hydroQueue);
                    sem_post(&hydroQueue);
                    hydrogen-=2;
                    sem_post(&oxyQueue);
                    oxygen-=1;
            }
            else
            {
                    sem_post(&mutex);
            }
            sem_wait(&hydroQueue);
            sleep(random()%5);
    }
}
```

```
void *Oxygen()
{
    while(1)
    {
        sem_wait(&mutex);
        oxygen+=1;
        printf("\nOxygen molecule number %d generated...",oxygen);
        if(hydrogen>=2)
        {
            ++count;
            printf("\nOxygen Bond.....Molecule number %d created...\n",count);
            sem_post(&hydroQueue);
            sem_post(&hydroQueue);
            hydrogen-=2;
            sem_post(&oxyQueue);
            oxygen-=1;
        }
        else
        {
            sem_post(&mutex);
        }
        sem_wait(&oxyQueue);
        sleep(random()%5);
        sem_post(&mutex);
    }
}

int main()
{
    sem_init(&mutex,0,1);
    sem_init(&oxyQueue,0,1);
    sem_init(&hydroQueue,0,1);
    pthread_t oxy,hydro1,hydro2;
    pthread_create(&oxy,NULL,Oxygen,NULL);
    pthread_create(&hydro1,NULL,Hydrogen,NULL);
    pthread_create(&hydro2,NULL,Hydrogen,NULL);
    pthread_join(oxy,NULL);
    pthread_join(hydro1,NULL);
    pthread_join(hydro2,NULL);
    return 0;
}
```

OUTPUT:


viknesh@viknesh-ubuntu: ~/Documents/OS/Lab_8
viknesh@viknesh-ubuntu:~/Documents/OS/Lab_8$ gcc 2b.c -lpthread
viknesh@viknesh-ubuntu:~/Documents/OS/Lab_8$ ./a.out
Oxygen molecule number 1 generated...
Hydrogen molecule number 1 generated...
Hydrogen molecule number 2 generated...
Hydrogen Bond.....Molecule number 1 created...
Oxygen molecule number 1 generated...
Hydrogen molecule number 1 generated...
Oxygen molecule number 2 generated...
Hydrogen molecule number 2 generated...
Hydrogen Bond.....Molecule number 2 created...
Hydrogen molecule number 1 generated...
Oxygen molecule number 2 generated...
Hydrogen molecule number 2 generated...
Hydrogen Bond.....Molecule number 3 created...
Hydrogen molecule number 1 generated...
Oxygen molecule number 2 generated...
Hydrogen molecule number 2 generated...
Hydrogen Bond.....Molecule number 4 created...
Oxygen molecule number 2 generated...
Hydrogen molecule number 1 generated...
Hydrogen molecule number 2 generated...
Hydrogen Bond.....Molecule number 5 created...


viknesh@viknesh-ubuntu: ~/Documents/OS/Lab_8
Oxygen molecule number 2 generated...
Hydrogen molecule number 1 generated...
Hydrogen molecule number 2 generated...
Hydrogen Bond.....Molecule number 6 created...
Hydrogen molecule number 1 generated...
Hydrogen molecule number 2 generated...
Hydrogen Bond.....Molecule number 7 created...
Oxygen molecule number 1 generated...
Hydrogen molecule number 1 generated...
Hydrogen molecule number 2 generated...
Hydrogen Bond.....Molecule number 8 created...
Oxygen molecule number 1 generated...
Hydrogen molecule number 1 generated...
Hydrogen molecule number 2 generated...
Hydrogen Bond.....Molecule number 9 created...
Oxygen molecule number 1 generated...
Hydrogen molecule number 1 generated...
Hydrogen molecule number 2 generated...
Hydrogen Bond.....Molecule number 10 created...
Oxygen molecule number 1 generated...
Hydrogen molecule number 1 generated...
Hydrogen molecule number 2 generated...
Hydrogen Bond.....Molecule number 11 created...
^C
viknesh@viknesh-ubuntu:~/Documents/OS/Lab_8$