

---

## Operating System Assignment-8

---

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

**Dining Philosophers solution:**

---

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

#define N 5
#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)
{
    if (state[phnum] == HUNGRY
        && state[LEFT] != EATING
        && state[RIGHT] != EATING) {
        // state that eating
        state[phnum] = EATING;

        sleep(2);

        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]) has no effect
        // during takefork
        // used to wake up hungry philosophers
        // during putfork
        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* philospher(void* num)
{
    while (1) {
        int* i = num;

        sleep(1);

        take_fork(*i);
    }
}

```

```

        sleep(0);
        put_fork(*i);
    }
}

int main()
{
    int i;
    pthread_t thread_id[N];

    // initialize the semaphores
    sem_init(&mutex, 0, 1);

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

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

```

**Output:**

---

```

harish@harish-ubuntu:~/Desktop/os/asst8$ gcc dining_phil.c -lpthread
harish@harish-ubuntu:~/Desktop/os/asst8$ ./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 4 is Hungry
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 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 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 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
^C
harish@harish-ubuntu:~/Desktop/os/asst8$

```

### Reader-Writer Solution:

---

```

#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
#include<unistd.h>
sem_t mutex, writeblock;
int data = 0, rcount = 0;
void *reader(void *arg);
void *writer(void *arg);

int main()
{
    int i, b;
    pthread_t rtid[5], wtid[5];
    sem_init(&mutex, 0, 1);
    sem_init(&writeblock, 0, 1);
    for(i=0; i<=2; i++)
    {
        pthread_create(&wtid[i], NULL, writer, &i);

```

```

        pthread_create(&rtid[i],NULL,reader, &i);
    }
    for(i=0;i<=2;i++)
    {
        pthread_join(wtid[i],NULL);
        pthread_join(rtid[i],NULL);
    }
    return 0;
}

void *reader(void *arg)
{
    int *t;
    int f;
    t= (int *) arg;
    f=*t;
    sem_wait(&mutex);
    rcount = rcount + 1;
    if(rcount==1)
        sem_wait(&writeblock);
    sem_post(&mutex);

    printf("Data read by the reader%d is %d\n",f,data);
    sleep(1);
    sem_wait(&mutex);
    rcount = rcount - 1;
    if(rcount==0)
        sem_post(&writeblock);

    sem_post(&mutex);
}

void *writer(void *arg)
{
    int *t;
    int f;
    t= (int *) arg;
    f=*t;
    sem_wait(&writeblock);
    data++;
    printf("Data writen by the writer%d is %d\n",f,data);
    sleep(1);
    sem_post(&writeblock);
}

```

**Output:**

---

```

harish@harish-ubuntu:~/Desktop/os/asst8$ gcc reader_writer.c -lpthread
harish@harish-ubuntu:~/Desktop/os/asst8$ ./a.out
Data written by the writer1 is 1
Data read by the reader1 is 1
Data read by the reader2 is 1
Data read by the reader0 is 1
Data written by the writer1 is 2
Data written by the writer2 is 3

```

**(B) 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.**

---

**H2O problem:**

---

**Approach:**

The size of hydroqueue is 2 and that of oxyqueue is 1. As the first two hydrogen threads arrive, they get pushed to the hydrogen queue, and as the first oxygen queue arrives, it gets pushed to the oxygen queue. Once the requirement to bond is achieved (2 x hydrogen and 1 x oxygen), the threads call where they bond after which they are moved to the barrier to make sure that the bonded hydrogen and oxygen are accounted for.

The forthcoming hydrogen and oxygen threads are made to wait until the their respective queues can accomodate them.

---

**Code:**

```

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

int hydrogen=0,oxygen=0,bcount=0;
sem_t mutex,hydroqueue,oxyqueue,b_mutex,sbarrier;

pthread_t o_thread,h_thread1,h_thread_2;

void bond()
{
    static int i=0;
    i++;
    if(i%3==0)
    {
        printf(" Water mol # %d created\n ",i/3);
    }
    sleep(2);
}

```

```

void * o_fn(void *arg)
{
    while(1)
    {
        sem_wait(&mutex);
        oxygen+=1;
        if(hydrogen>=2)
        {
            sem_post(&hydroqueue);
            sem_post(&hydroqueue); //increase by 2 so twice --> allows 2
H molecules
            hydrogen-=2;
            sem_post(&oxyqueue);
            oxygen-=1;
        }
        else
        {
            sem_post(&mutex);
        }

        sem_wait(&oxyqueue);
        printf(" one Oxygen is ready\n");
        bond();

        //barrier_wait();
        sem_post(&mutex); //release the lock acquired (this lock can be
acquired by any thread and there has to be one unlock thats it) after 1
H2O molecule is done
    }
}

void *h_fn(void *arg)
{
    while(1)
    {
        sem_wait(&mutex);
        hydrogen+=1;

        if(hydrogen>=2 && oxygen>=1)
        {
            sem_post(&hydroqueue);
            sem_post(&hydroqueue);
            hydrogen-=2;
            sem_post(&oxyqueue);
            oxygen-=1;
        }
        else
        {
            sem_post(&mutex);
        }

        sem_wait(&hydroqueue);
        printf(" 1 Hydrogen molecule ready ");
    }
}

```

```

        bond();

        //barrier_wait();
    }
}

int main()
{
    sem_init(&b_mutex,0,1);
    sem_init(&sbarrier,0,0);
    sem_init(&mutex,0,1);
    sem_init(&oxyqueue,0,0);
    sem_init(&hydroqueue,0,0);

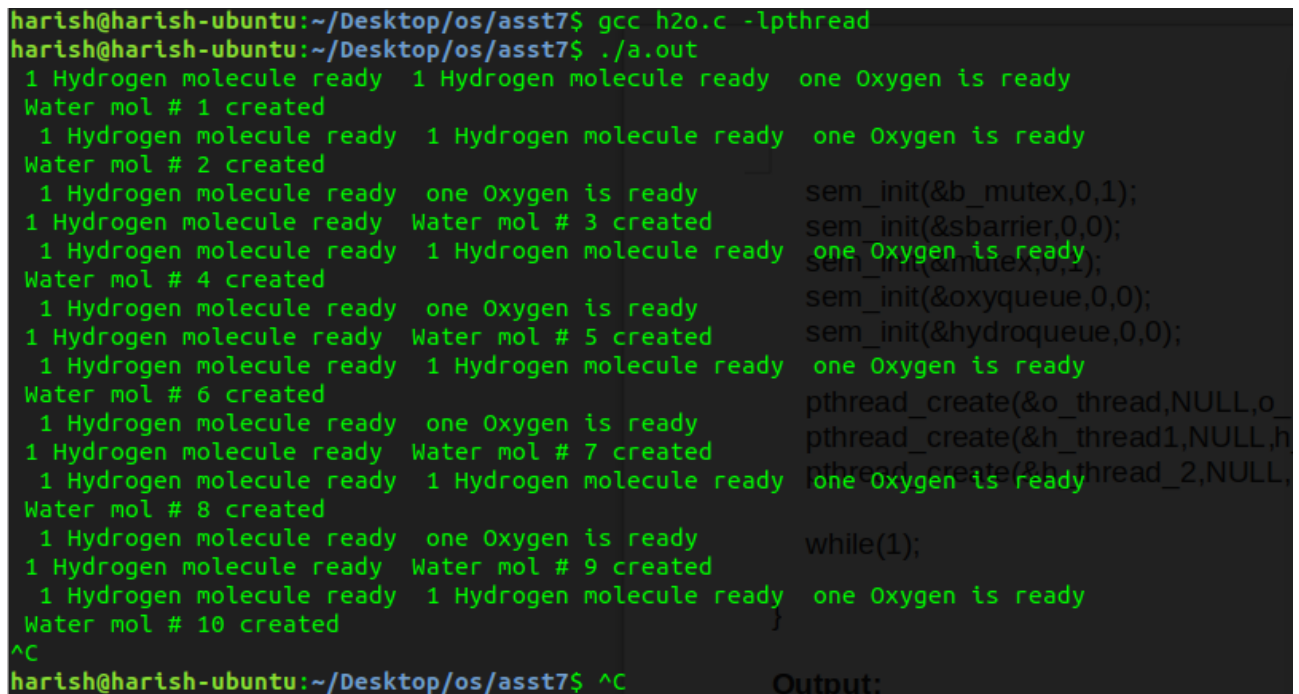
    pthread_create(&o_thread,NULL,o_fn,NULL);
    pthread_create(&h_thread1,NULL,h_fn,NULL);
    pthread_create(&h_thread_2,NULL,h_fn,NULL);

    while(1);
}

```

## Output:

---



```

harish@harish-ubuntu:~/Desktop/os/asst7$ gcc h2o.c -lpthread
harish@harish-ubuntu:~/Desktop/os/asst7$ ./a.out
1 Hydrogen molecule ready  1 Hydrogen molecule ready  one Oxygen is ready
Water mol # 1 created
1 Hydrogen molecule ready  1 Hydrogen molecule ready  one Oxygen is ready
Water mol # 2 created
1 Hydrogen molecule ready  one Oxygen is ready
1 Hydrogen molecule ready  Water mol # 3 created
1 Hydrogen molecule ready  1 Hydrogen molecule ready  one Oxygen is ready
Water mol # 4 created
1 Hydrogen molecule ready  one Oxygen is ready
1 Hydrogen molecule ready  Water mol # 5 created
1 Hydrogen molecule ready  1 Hydrogen molecule ready  one Oxygen is ready
Water mol # 6 created
1 Hydrogen molecule ready  one Oxygen is ready
1 Hydrogen molecule ready  Water mol # 7 created
1 Hydrogen molecule ready  1 Hydrogen molecule ready  one Oxygen is ready
Water mol # 8 created
1 Hydrogen molecule ready  one Oxygen is ready
1 Hydrogen molecule ready  Water mol # 9 created
1 Hydrogen molecule ready  1 Hydrogen molecule ready  one Oxygen is ready
Water mol # 10 created
^C
harish@harish-ubuntu:~/Desktop/os/asst7$ ^C

```

## Santa Claus Solution:

---

### Approach:

---

elves and reindeer are counters, both protected by mutex. Elves and reindeer get mutex to modify the counters; Santa gets it to check them. Santa waits on santasem until either an



elf or a reindeer signals him. The reindeer wait on reindeerSem until Santa signals them to enter the paddock and get hitched. The elves use elfsem to prevent additional elves from entering while three elves are being helped.

When Santa wakes up, he checks which of the two conditions holds and either deals with the reindeer or the waiting elves. If there are nine reindeer waiting, Santa prepares Sleigh, then signals reindeerSem nine times, allowing the reindeer to invoke getHitched. If there are elves waiting, Santa just helps Elves.

### Code:

---

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

int elves=0, reindeer=0;
sem_t mutex;
sem_t santasem, reindeersem, elfsem;

pthread_t *CreateThread(void *(*f)(void *), void *a)
{
    pthread_t *t = malloc(sizeof(pthread_t));

    int ret = pthread_create(t, NULL, f, a);

    return t;
}

void* Santa(void* param)
{
    printf("inside santa!\n");
    while(1)
    {
        sem_wait(&santasem);
        sem_wait(&mutex);
        if(reindeer>=9)
        {
            printf("Sleigh prepared! Happy Christmas!\n");
            for(int i=0; i<9; ++i)
                sem_post(&reindeersem);
            reindeer=0;
        }
        else if(elves==3)
            printf("Santa is helping elves\n");
        sem_post(&mutex);
    }
    pthread_exit(0);
}

void* Reindeer(void* param)
{

```

```

int* id=(int *) param;
int rid=*id;
printf("[R]:reindeer number:%d came\n",rid);
while(1)
{
    sem_wait(&mutex);
    reindeer+=1;
    //printf("reindeer count:%d\n",reindeer);
    if(reindeer==9)
        sem_post(&santasem);
    sem_post(&mutex);
    sem_wait(&reindeersem);
    printf("[R] Reindeer %d getting hitched!\n",rid);
    sleep(20);
}
pthread_exit(0);
}

void* Elves(void *param)
{
    int* id=(int *) param;
    int rid=*id;
    //printf("[E]:elf number:%d came\n",rid);
    while(1)
    {
        sem_wait(&elfsem);
        sem_wait(&mutex);
        elves+=1;
        //printf("elves count:%d\n",elves);
        if(elves==3)
            sem_post(&santasem);
        else
            sem_post(&elfsem);
        sem_post(&mutex);
        printf("[E]:elf %d need help!\n",rid);
        sleep(10);

        sem_wait(&mutex);
        elves-=1;
        if(elves==0)
            sem_post(&elfsem);
        sem_post(&mutex);
        sleep(10);
    }
    pthread_exit(0);
}

int main()
{
    sem_init(&mutex, 0, 1);
    sem_init(&elfsem, 0, 1);
    sem_init(&santasem, 0, 0);
    sem_init(&reindeersem, 0, 0);
    int i=0;

```

```

pthread_t tid,eid[9],rid[9];
pthread_create(&tid,NULL,Santa,NULL);
//pthread_t *santa_claus=CreateThread(Santa,0);
int r[]={0,1,2,3,4,5,6,7,8};
while(i<9)
{
    pthread_create(&rid[i],NULL,Reindeer,&r[i]);
    pthread_create(&eid[i],NULL,Elves,&r[i]);
    i++;
}

pthread_join(tid,NULL);
for(int i=0;i<9;++i)
{
    pthread_join(rid[i],NULL);
    pthread_join(eid[i],NULL);
}
return 0;
}

```

## Output:

```

harish@harish-ubuntu:~/Desktop/os/asst8$ gcc santa_claus.c -lpthread
harish@harish-ubuntu:~/Desktop/os/asst8$ ./a.out
inside santa!
[R]:reindeer number:0 came
[E]:elf 0 need help!
[R]:reindeer number:2 came
[E]:elf 2 need help!
[E]:elf 1 need help!
[R]:reindeer number:3 came
Santa is helping elves
[R]:reindeer number:1 came
[R]:reindeer number:4 came
[R]:reindeer number:5 came
[R]:reindeer number:6 came
[R]:reindeer number:7 came
[R]:reindeer number:8 came
Sleigh prepared! Happy Christmas!
[R] Reindeer 0 getting hitched!
[R] Reindeer 1 getting hitched!
[R] Reindeer 6 getting hitched!
[R] Reindeer 7 getting hitched!
[R] Reindeer 3 getting hitched!
[R] Reindeer 2 getting hitched!
[R] Reindeer 8 getting hitched!
[R] Reindeer 4 getting hitched!
[R] Reindeer 5 getting hitched!
[E]:elf 3 need help!
[E]:elf 5 need help!
[E]:elf 4 need help!
Santa is helping elves
^C
harish@harish-ubuntu:~/Desktop/os/asst8$

```