OPERATING SYSTEMS

ASSIGNMENT 8

SEMAPHORES

R SHREJA

COE18B043

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

DINING PHILOSOPHERS PROBLEM.

```c
#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 //the philosopher to the left of phnum
#define RIGHT (phnum + 1) % N //the philosopher to the right of phnum
//state in which the phil is in
int state[N];
//the 5 philosophers
int phil[N] = {0, 1, 2, 3, 4};
//init semaphores
sem_t mutex;
```

```
sem_t S[N];

//if a phil is hungry and the left and the right phils
//arent eating (left and right forks are free)
//change the state from hungry to eating of phnum
void test(int phnum){
   if (state[phnum] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING){
       state[phnum] = EATING;

       sleep(2);//eat fir 2 secs

       printf("Philosopher %d takes fork %d and %d\n",
              phnum + 1, LEFT + 1, phnum + 1);

       printf("Philosopher %d is Eating\n", phnum + 1);
       //send signal that pnum is done
       sem_post(&S[phnum]);
   }
}
void take_fork(int phnum){
   //if hungry enter the loop and put the phil in wait and set state as HUNGRY
   sem_wait(&mutex);

   state[phnum] = HUNGRY;

   printf("Philosopher %d is Hungry\n", phnum + 1);
   //test if left and right forks are free
   test(phnum);
   sem_post(&mutex);
```

```
    // if not free wait to be signalled
    sem_wait(&S[phnum]);

    sleep(1);
}

// put down forks
void put_fork(int phnum){
    //start of cs
    sem_wait(&mutex);

    // change the state to 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);
    //see if the left or the right neigbours want the forks
    test(LEFT);
    test(RIGHT);
    //end of cs
    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);
    // create philosopher processes
    for (i = 0; i < N; i++){
        pthread_create(&thread_id[i], NULL,
                       philospher, &phil[i]);

        printf("Philosopher %d is thinking\n", i + 1);
    }
    for (i = 0; i < N; i++)
        pthread_join(thread_id[i], NULL);
    return 0;
}
```

OUTPUT:

```
shreja@lostinspace:~/Desktop/OS_LAB$ ./dp
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 3 is Hungry
Philosopher 4 is Hungry
Philosopher 2 is Hungry
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 2 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 5 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
^C
```

READER WRITER PROBLEM:

```c
#include<semaphore.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
//init semaphores
sem_t x,y;
//create threads
pthread_t tid;
pthread_t writerthreads[100],readerthreads[100];
//tracking number of readers
int readercount = 0;

//reader thread
void *reader(void* param){

   sem_wait(&x);
   readercount++;

   //if a reader is reading put the writer on wait
   if(readercount==1)
       sem_wait(&y);
   sem_post(&x);

   printf("%d reader is inside\n",readercount);
   usleep(3);
   sem_wait(&x);
   readercount--;
```

```
    if(readercount==0){
        sem_post(&y);
    }
    sem_post(&x);
    printf("%d Reader is leaving\n",readercount+1);
    return NULL;
}

void *writer(void* param){
    printf("Writer is trying to enter\n");
    sem_wait(&y);
    printf("Writer has entered\n");
    sem_post(&y);
    printf("Writer is leaving\n");
    return NULL;
}

int main(){

    int num_readers;
    //user input number of readers
    printf("Enter the number of readers:");
    scanf("%d",&num_readers);
    printf("\n");

    int n1[num_readers]

    sem_init(&x,0,1);
    sem_init(&y,0,1);
```

```
    for(int i=0;i<num_readers;i++){
        pthread_create(&writerthreads[i],NULL,reader,NULL);
        pthread_create(&readerthreads[i],NULL,writer,NULL);
    }

    for(int i=0;i<num_readers;i++){
        pthread_join(writerthreads[i],NULL);
        pthread_join(readerthreads[i],NULL);
    }

}
```

OUTPUT:

```
shreja@lostinspace:~/Desktop/OS_LAB$ gcc reader_writer.c -lpthread
shreja@lostinspace:~/Desktop/OS_LAB$ ./a.out
Enter the number of readers:3

1 reader is inside
Writer is trying to enter
2 reader is inside
2 Reader is leaving
Writer is trying to enter
1 Reader is leaving
Writer has entered
Writer is leaving
Writer has entered
Writer is trying to enter
1 reader is inside
Writer is leaving
1 Reader is leaving
Writer has entered
Writer is leaving
```

(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.

(1)Santa Claus Problem

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

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

//init the number of elves
static const int N_ELfS = 10;
static const int N_REINDEER = 9;

static int elfs;
static int reindeer;
```

```
static sem_t santaSem;
static sem_t reindeerSem;
static sem_t elfTex;
static sem_t mutex;

void *SantaClaus(void *arg){

   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++)
               sem_post(&reindeerSem);
           printf("Santa Claus: make all kids in the world happy\n");

           reindeer = 0;
       }

       else if (elfs == 3)
           printf("Santa Claus: helping elfs\n");

       sem_post(&mutex);
   }
   return arg;
}
```

```
void *Reindeer(void *arg){

   int id = (int)arg;
   printf("This is reindeer %d\n", id);

   while (1){
       sem_wait(&mutex);
       reindeer++;

       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 *Elf(void *arg){

   int id = (int)arg;
   printf("This is elf %d\n", id);
   while (true){

       bool need_help = random() % 100 < 10;
       if (need_help){
```

```
            sem_wait(&elfTex);
            sem_wait(&mutex);

            elfs++;
            if (elfs == 3)
                sem_post(&santaSem);
            else
                sem_post(&elfTex);

            sem_post(&mutex);

            printf("Elf %d will get help from Santa Claus\n", id);
            sleep(10);

            sem_wait(&mutex);

            elfs--;
            if (elfs == 0)
                sem_post(&elfTex);
            sem_post(&mutex);
        }

        // Do some work
        printf("Elf %d at work\n", id);
        sleep(2 + random() % 5);
    }

    return arg;
}
```

```
int main(int ac, char **av){

   elfs = 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++)
       reindeers[r] = CreateThread(Reindeer, (void *)r + 1);

   pthread_t *elfs[N_ELfS];
   for (int e = 0; e < N_ELfS; e++)
       elfs[e] = CreateThread(Elf, (void *)e + 1);

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

OUTPUT:

```
shreja@lostinspace:~/Desktop/LAB_8$ ./santa_claus
This is reindeer 1
This is reindeer 4
This is reindeer 3
This is reindeer 2
This is reindeer 5
This is reindeer 6
This is reindeer 7
This is reindeer 8
This is elf 1
Elf 1 at work
This is elf 2
Elf 2 at work
This is reindeer 9
Santa Claus: preparing sleigh
This is elf 4
Elf 4 at work
Reindeer 3 getting hitched
Reindeer 7 getting hitched
Santa Claus: make all kids in the world happy
Reindeer 2 getting hitched
This is elf 7
Elf 7 at work
Reindeer 1 getting hitched
Reindeer 4 getting hitched
This is elf 8
Elf 8 at work
Reindeer 6 getting hitched
This is elf 6
Elf 6 at work
This is elf 3
Elf 3 at work
Reindeer 8 getting hitched
Reindeer 9 getting hitched
This is elf 10
Elf 10 at work
```

EXPLANATION:

The solution consists mainly of an outer class SantaClaus, that sets up all needed synchronisation variables and controls the program termination, and two inner classes Elf and Reindeer, that are instantiated on a separate thread for each individual elf and reindeer behaviour. The source code is also available as syntax coloured HTML; an earlier version without the harnessing part can be found here.

The barrier that manages the grouping of the elves is protected by a Semaphore ElfTex with three permits. This implements the requirement “any other elves wishing to visit Santa must wait for those elves to return” in a rather defensive manner: there is some virtual waiting room for the elves to wait before waking Santa that has room for only three elves

(2) H2O Problem

```c
#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;

//create 3 therads one each for h h and o
pthread_t o_thread,h_thread1,h_thread_2;


//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.
void barrier_wait(){

   sem_wait(&b_mutex);
   bcount++;
   sem_post(&b_mutex);

   if(bcount==3)
       sem_post(&sbarrier);

   sem_wait(&sbarrier);
   sem_post(&sbarrier);
```

```
}

//if 2 h's and one o are avaialable they are bonded
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 H20 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:

```
shreja@lostinspace:~/Desktop/LAB_8$ gcc h2o.c -o h2o -pthread
shreja@lostinspace:~/Desktop/LAB_8$ ./h2o
 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  1 Hydrogen molecule ready  one Oxygen is 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  1 Hydrogen molecule ready  one Oxygen is 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  1 Hydrogen molecule ready  one Oxygen is ready
 Water mol # 9 created
  1 Hydrogen molecule ready  1 Hydrogen molecule ready  one Oxygen is ready
 Water mol # 10 created
  1 Hydrogen molecule ready  1 Hydrogen molecule ready  one Oxygen is ready
 Water mol # 11 created
  1 Hydrogen molecule ready  1 Hydrogen molecule ready  one Oxygen is ready
 Water mol # 12 created
  1 Hydrogen molecule ready  1 Hydrogen molecule ready  one Oxygen is ready
 Water mol # 13 created
  1 Hydrogen molecule ready  1 Hydrogen molecule ready  one Oxygen is ready
 Water mol # 14 created
^C
```