Assignment 7

Topic: Multithreading

Sl. No.	Question
1.	Write a program to implement multithreading concept where a thread checks for a number to be odd or even and another thread checks whether the number is prime or not.

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
void *odd_even(void *ptr);
void *prime(void *ptr);
main()
{
        int iret1,iret2,*number;
        pthread_t thread1,thread2;
        printf("enter the number\n");
        number = (int*)malloc(4);
        scanf("%d",number);
        iret1=pthread_create(&thread1,NULL,odd_even,(void *) number);
        iret2=pthread_create(&thread2,NULL,prime,(void *) number);
        pthread_join(thread1,NULL);
        pthread_join(thread2,NULL);
        exit(0);
}
void * odd_even(void *ptr)
{
        int *no;
        no=(int *)ptr;
        if(*no%2==0)
                printf("\n%d is an even number\n",*no);
        else
                printf("\n%d is an odd number\n",*no);
        sleep(3);
}

void * prime(void *ptr)
{
        int i=1,n=0,*no;
        no=(int *)ptr;
        while(i<=*no)
        {
                if(*no%i==0)
                        n++;
                i++;
        }
        if(n==2)
        {
                printf("\n%d is a prime number\n",*no);
        }
        else
                printf("\n%d is not a prime number\n",*no);
```

```
        sleep(3);
}
```

2. Create a multi-threaded C program having two threads and a shared variable sum (which is initialized to 0). The first thread should read an integer number. The second thread should find the square of the inputted number, add it to sum and print the sum. The process should go on up to forced exit by the user. Note that second thread should start the work only after first thread finishes for each inputted number.

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

char str[1024];
pthread_mutex_t lock=PTHREAD_MUTEX_INITIALIZER;
int no_read=0,no,sum=0;
pthread_cond_t cond;

void *read()
{
        while(1)
        {
                while(no_read);
                pthread_mutex_lock(&lock);
                printf("\nEnter a no:");
                scanf("%d",&no);
                no_read=1;
                pthread_mutex_unlock(&lock);
                pthread_cond_signal(&cond);
        }
}
void *SquareSum()
{
        while(1)
        {
                pthread_mutex_lock(&lock);
                while(!no_read)
                        pthread_cond_wait(&cond,&lock);
                sum+=no*no;
                printf("\nSum is : %ds\n",sum);
                no_read=0;
                pthread_mutex_unlock(&lock);
        }
}
main()
{
        pthread_t tr,tw;

        pthread_create(&tr,NULL,read,NULL);
        pthread_create(&tw,NULL,SquareSum,NULL);

        pthread_join(tr,NULL);
        pthread_join(tw,NULL);

}
```

3. Write a program to implement multithreading concept by finding the scalar product of two vectors. In the main program first read the two vectors. Create two threads. First thread should multiply first half of the vector and second should multiply the second half. Finally find the sum in main and print.

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
void *block1(void *ptr);
void *block2(void *ptr);
int mat1[]={1,2,3,4,5};
int mat2[]={1,2,3,4,5},res[5];
pthread_mutex_t mutex1=PTHREAD_MUTEX_INITIALIZER;
void sum(int i)
{
        pthread_mutex_lock(&mutex1);
        res[i]=mat1[i]*mat2[i];
        pthread_mutex_unlock(&mutex1);
}

main()
{
        int iret1,iret2,*number,s=0,i;
        pthread_t thread1,thread2;
        /*printf("enter the size of vector\n");
        scanf("%d",&p);
        printf("enter the value of 1 st vector");
        for(i=0;i<p;i++)
                scanf("%d",&mat1[i]);
        printf("enter the value of 2nd vector");
        for(i=0;i<p;i++)
                scanf("%d",&mat2[i]);*/
        number=(int *)malloc(sizeof(int));
        *number=5;
        iret1=pthread_create(&thread1,NULL,block1,(void *) number);
        iret2=pthread_create(&thread2,NULL,block2,(void *) number);
        pthread_join(thread1,NULL);
        pthread_join(thread2,NULL);
        for(i=0;i<*number;i++)
        {
                s=s+res[i];
                printf("%d\t",res[i]);
        }
        printf("\nsum of product of two vectors is %d\n",s);
        exit(0);
}
void * block2(void *ptr)
{
        int *no,i;
        no=(int *)ptr;
        int n=*no;
        for(i=n/2;i<n;i++)
                sum(i);
        sleep(2);
}
void * block1(void *ptr)
{
```

```
        int *no,i;
        no=(int *)ptr;
        int n=*no;
        for(i=0;i<n/2;i++)
                sum(i);
        sleep(2);
}
```

Output:

Enter the size of vector:5
Enter the value of 1st vector : 1 2 3 4 5
Enter the value of 2nd vector : 5 4 3 2 1
Sum of product of two vectors is 35

4. Write a multi-threaded C program in which we have a shared variable CurrentID. This is initialized to 1 at the beginning. Now create 5 threads in your program and assign ID 1, 2, 3, 4, 5 to them respectively. You can pass the ID as a parameter when you create the threads. Each of the threads will try to access the variable "CurrentID". Whenever a thread acquires the variable, it checks whether the CurrentId is equal to its own Id or not. If it is not equal, it will output "Not My Turn!", then print its threadId, and then release the variable. If it is equal, the thread will print "My turn!", then print its threadId, increase the CurrentId by 1, and then release the variable. However, after increasing CurrentID by 1, the thread will check if the value is 5 or not. If it is 5, it will reset it to 1 before releasing the variable.

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

pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;
int CurrentID=1;
void *function(void *n)
{
        char *s=(char *)n;
        int ownID=*((int *)s)+1;
        while(1)
        {
                sleep(1);
                pthread_mutex_lock(&mut);
                printf("\nCurrent ID=%d, OwnID=%d",CurrentID,ownID);
                if(CurrentID==ownID)
                {
                        printf("\nMy Turn. Thread ID=%ld",pthread_self());

                        CurrentID++;
                        if(CurrentID==5)
                                CurrentID=1;
                }
                else
                        printf("\nNot My Turn. Thread ID=%ld",pthread_self());
                pthread_mutex_unlock(&mut);
        }

}

main()
{
        pthread_t tid[5];
```

```
        int i,*p;
        char str[10];
        for(i=0;i<5;i++)
        {
                p=&i;
                pthread_create(&tid[i],NULL,function,(void *)p);
        }
        for(i=0;i<5;i++)
                pthread_join(tid[i],NULL);

}
```