OS LAB Exp:5

```c
//5a.Write a program to achieve synchronization between multiple threads. Both
    threads make use of semaphore variables to update a shared variable so that
    only one of the threads is executing in its critical section.(use semaphore
    functions)
#include<pthread.h>
#include<stdio.h>
#include<semaphore.h>
#include<unistd.h>
void *fun1();
 void *fun2();
 int shared=1; //shared variable
 sem_t s; //semaphore variable
 int main()
 {
 sem_init(&s,0,1); //initialize semaphore variable - 1st argument is address of
     variable, 2nd is number of processes sharing semaphore, 3rd argument is the
     initial value of semaphore variable
 pthread_t thread1, thread2;
 pthread_create(&thread1, NULL, fun1, NULL);
 pthread_create(&thread2, NULL, fun2, NULL);
 pthread_join(thread1, NULL);
 pthread_join(thread2,NULL);
 printf("Final value of shared is %d\n",shared); //prints the last updated value
     of shared variable
 }
 void *fun1()
{
    int x;
    sem_wait(&s); //executes wait operation on s
    x=shared;//thread1 reads value of shared variable
    printf("Thread1 reads the value as %d\n",x);
    x++; //thread1 increments its value
    printf("Local updation by Thread1: %d\n",x);
    sleep(1); //thread1 is preempted by thread 2
    shared=x; //thread one updates the value of shared variable
    printf("Value of shared variable updated by Thread1 is: %d\n",shared);
    sem_post(&s);
}
void *fun2()
{
    int y;
    sem_wait(&s);
    y=shared;//thread2 reads value of shared variable
    printf("Thread2 reads the value as %d\n",y);
    y--; //thread2 increments its value
    printf("Local updation by Thread2: %d\n",y);
    sleep(1); //thread2 is preempted by thread 1
    shared=y; //thread2 updates the value of shared variable
    printf("Value of shared variable updated by Thread2 is: %d\n",shared);
    sem_post(&s);
}
```

Output

```
Thread1 reads the value as 1
Local updation by Thread1: 2
Value of shared variable updated by Thread1 is: 2
Thread2 reads the value as 2
Local updation by Thread2: 1
Value of shared variable updated by Thread2 is: 1
Final value of shared is 1
```

```c
// 5. b) Simulate Bounded-buffer (or Producer-Consumer) problem solution using
    Semaphore
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
#include<stdlib.h>
sem_t empty,full;
pthread_mutex_t mutex;
int buffer[5];
int count=0;
void*producer(void*arg)
{
    long int num=(long int)arg;
    sem_wait(&empty);
    pthread_mutex_lock(&mutex);
    buffer[count]=rand()%10;
    printf("\n Producer %ld Produced: %d",num+1,buffer[count]);
    count++;
    sleep(1);
    pthread_mutex_unlock(&mutex);
    sem_post(&full);
    pthread_exit(NULL);
}
void*consumer(void*arg)
{
    long int num=(long int)arg;
    sem_wait(&full);
    pthread_mutex_lock(&mutex);
   count--;
   printf("\n consumer %ld consumed: %d",num+1,buffer[count]);
   sleep(1);
   pthread_mutex_unlock(&mutex);
   sem_post(&empty);
   pthread_exit(NULL);
}
int main()
{
   int np,nc;
   pthread_t p[10],c[10];
   unsigned long int i,j,k,l;
   printf("Enter number of producers and number of consumers");
   scanf("%d %d",&np,&nc);
   sem_init(&empty,0,5);
   sem_init(&full,0,0);
   pthread_mutex_init(&mutex,NULL);
   for(i=0;i<np;i++)
       pthread_create(&p[i],NULL,producer,(void*)i);
   for(j=0;j<nc;j++)
       pthread_create(&c[j],NULL,consumer,(void*)j);
   for(k=0;k<np;k++)
       pthread_join(p[k],NULL);
   for(l=0;l<nc;l++)
       pthread_join(c[l],NULL);
}
```

Output

```
Enter number of producers and number of consumers4 3

Producer 1 Produced: 3
Producer 2 Produced: 6
Producer 3 Produced: 7
Producer 4 Produced: 5
consumer 1 consumed: 5
consumer 3 consumed: 7
consumer 2 consumed: 6
```

```c
//5c.Simulation of reader's writer's problem using semaphore
#include<pthread.h>
#include<semaphore.h>
#include<stdio.h>
sem_t wrt;
pthread_mutex_t mutex;
int cnt=1;
int numreader=0;
void *reader(void *rno)
{
 pthread_mutex_lock(&mutex);
 numreader++;
 if(numreader==1)
 {
  sem_wait(&wrt);
 }
 void *writer(void *wno)
{
 sem_wait(&wrt);
 cnt=cnt*2;
 printf("Writer %d modified cnt to %d\n",(*((int*)wno)),cnt);
 sem_post(&wrt);
}
 pthread_mutex_unlock(&mutex);
 printf("Reader %d: read cnt as %d\n",*((int *)rno),cnt);
 pthread_mutex_lock(&mutex);
 numreader--;
 if(numreader==0)
 {
  sem_post(&wrt);
 }
 pthread_mutex_unlock(&mutex);
}
int main()
{
 pthread_t read[10],write[5];
 pthread_mutex_init(&mutex,NULL);
 sem_init(&wrt,0,1);
 int a[10]={1,2,3,4,5,6,7,8,9,10};
 for(int i=0;i<10;i++)
 {
  pthread_create(&read[i],NULL,(void *)reader,(void *)&a[i]);
 }
 for(int i=0;i<5;i++)
 {
  pthread_create(&write[i],NULL,(void *)writer,(void *)&a[i]);
 }
 for(int i=0;i<10;i++)
 {
  pthread_join(read[i],NULL);
 }
 for(int i=0;i<5;i++)
 {
  pthread_join(write[i],NULL);
 }
 pthread_mutex_destroy(&mutex);
}
```

Output

```
Reader 1: read cnt as 1
Reader 2: read cnt as 1
Reader 3: read cnt as 1
Reader 4: read cnt as 1
Reader 5: read cnt as 1
Reader 6: read cnt as 1
Reader 7: read cnt as 1
Reader 8: read cnt as 1
Reader 9: read cnt as 1
Writer 3 modified cnt to 2
Reader 10: read cnt as 2
Writer 1 modified cnt to 4
Writer 4 modified cnt to 8
Writer 2 modified cnt to 16
Writer 5 modified cnt to 32
```

```c
//5d.Dining-Philosophers problem solution using semaphore
#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>
#include<unistd.h>
sem_t chop[5];
sem_t mutex;
char status[6]={'-','-','-','-','-'};
void printstatus()
{
 int i;
 printf("\n");
 for(i=0;i<5;i++)
  printf(" %c ",status[i]);
}
void *Philosopher(void *arg)
{
 long int num=(long int) arg;
 status[num]='H';
 printstatus();
 sem_wait(&mutex);
 sem_wait(&chop[num]);
 sleep(1);
 sem_wait(&chop[(num+1)%5]);
 status[num]='E';
 printstatus();
 sleep(1);
 sem_post(&chop[(num+1)%5]);
 sem_post(&chop[num]);
 sem_post(&mutex);
 status[num]='T';
 printstatus();
 pthread_exit(NULL);
}
int main()
{
 pthread_t phil[5];
 long int i,j;
 for(i=0;i<5;i++)
  sem_init(&chop[i],0,1);
 sem_init(&mutex,0,1);
 printf("\n");
 for(i=0;i<5;i++)
  printf(" P[%ld] ",i);
 for(i=0;i<5;i++)
  pthread_create(&phil[i],NULL,Philosopher,(void *)i);
 for(j=0;j<5;j++)
  pthread_join(phil[j],NULL);
}
```

Output

```
P[0]  P[1]  P[2]  P[3]  P[4]
 H  -  -  -  -
 H  H  -  -  -
 H  H  H  -  -
 H  H  H  H  -
H  H  H  H  H
E  H  H  H  H
T  H  H  H  H
T  E  H  H  H
T  T  H  H  H
T  T  E  H  H
T  T  T  H  H
T  T  T  E  H
T  T  T  T  H
T  T  T  T  E
T  T  T  T  T
```