VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering

Assessment - III, FEBRUARY 2020

B.Tech, Winter-2019-2020

NAME	PRIYAL BHARDWAJ
REG. NO.	18BIT0272
COURSE CODE	ITE2002
COURSE NAME	OPERATING SYSTEMS
SLOT	L-37+L-38
FACULTY	Prof. SUDHA S.

(a) Implement the solution for reader – writer's problem.

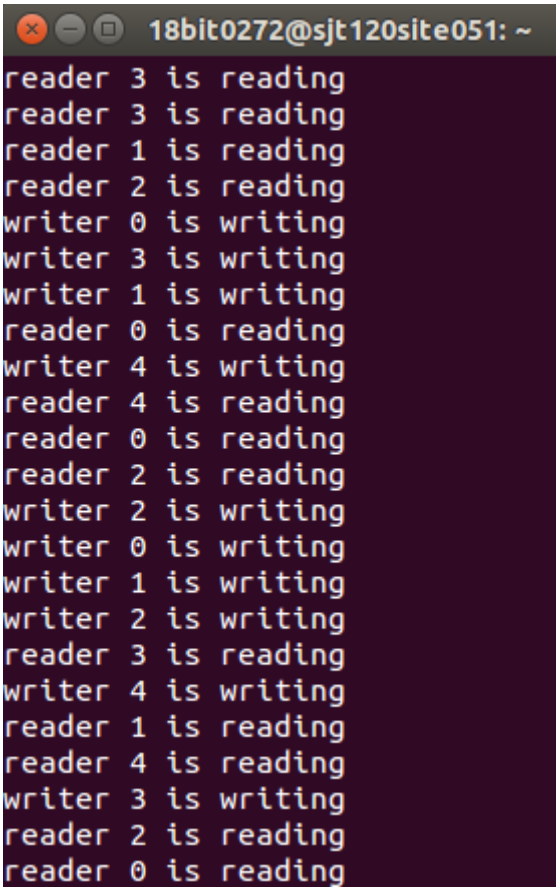
CODE/OUTPUT:

```
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;

class monitor {
private:
    // no. of readers
    int rcnt;

    // no. of writers
    int wcnt;

    // no. of readers waiting
    int waitr;

    // no. of writers waiting
    int waitw;

    // condition variable to check whether reader can read
    pthread_cond_t canread;

    // condition variable to check whether writer can write
    pthread_cond_t canwrite;

    // mutex for synchronisation
    pthread_mutex_t condlock;

public:
    monitor()
    {
        rcnt = 0;
        wcnt = 0;
        waitr = 0;
        waitw = 0;

        pthread_cond_init(&canread, NULL);
        pthread_cond_init(&canwrite, NULL);
        pthread_mutex_init(&condlock, NULL);
    }

    // mutex provide synchronisation so that no other thread
    // can change the value of data
    void beginread(int i)
    {
        pthread_mutex_lock(&condlock);

        // if there are active or waiting writers
        if (wcnt == 1 || waitw > 0) {
            // incrementing waiting readers
```

```
            waitr++;

            // reader suspended
            pthread_cond_wait(&canread, &condlock);
            waitr--;
        }

        // else reader reads the resource
        rcnt++;
        cout << "reader " << i << " is reading\n";
        pthread_mutex_unlock(&condlock);
        pthread_cond_broadcast(&canread);
    }

    void endread(int i)
    {

        // if there are no readers left then writer enters monitor
        pthread_mutex_lock(&condlock);

        if (--rcnt == 0)
            pthread_cond_signal(&canwrite);

        pthread_mutex_unlock(&condlock);
    }

    void beginwrite(int i)
    {
        pthread_mutex_lock(&condlock);

        // a writer can enter when there are no active
        // or waiting readers or other writer
        if (wcnt == 1 || rcnt > 0) {
            ++waitw;
            pthread_cond_wait(&canwrite, &condlock);
            --waitw;
        }
        wcnt = 1;
        cout << "writer " << i << " is writing\n";
        pthread_mutex_unlock(&condlock);
    }

    void endwrite(int i)
    {
        pthread_mutex_lock(&condlock);
        wcnt = 0;

        // if any readers are waiting, threads are unblocked
        if (waitr > 0)
            pthread_cond_signal(&canread);
        else
            pthread_cond_signal(&canwrite);
        pthread_mutex_unlock(&condlock);
    }

}
```

```
// global object of monitor class
M;

void* reader(void* id)
{
    int c = 0;
    int i = *(int*)id;

    // each reader attempts to read 5 times
    while (c < 5) {
        usleep(1);
        M.beginread(i);
        M.endread(i);
        c++;
    }
}

void* writer(void* id)
{
    int c = 0;
    int i = *(int*)id;

    // each writer attempts to write 5 times
    while (c < 5) {
        usleep(1);
        M.beginwrite(i);
        M.endwrite(i);
        c++;
    }
}

int main()
{
    pthread_t r[5], w[5];
    int id[5];
    for (int i = 0; i < 5; i++) {
        id[i] = i;

        // creating threads which execute reader function
        pthread_create(&r[i], NULL, &reader, &id[i]);

        // creating threads which execute writer function
        pthread_create(&w[i], NULL, &writer, &id[i]);
    }

    for (int i = 0; i < 5; i++) {
        pthread_join(r[i], NULL);
    }
    for (int i = 0; i < 5; i++) {
        pthread_join(w[i], NULL);

       }
```


18bit0272@sjt120site051: ~
18bit0272@sjt120site051:~$ g++ 1.cpp -lpthread
18bit0272@sjt120site051:~$ ./a.out
writer 2 is writing
reader 0 is reading
reader 2 is reading
reader 1 is reading
reader 2 is reading
writer 0 is writing
writer 4 is writing
writer 1 is writing
reader 4 is reading
reader 0 is reading
reader 1 is reading
writer 4 is writing
reader 3 is reading
writer 2 is writing
reader 4 is reading
writer 2 is writing
writer 3 is writing
reader 3 is reading
writer 1 is writing
writer 4 is writing
reader 4 is reading


18bit0272@sjt120site051: ~
reader 3 is reading
reader 3 is reading
reader 1 is reading
reader 2 is reading
writer 0 is writing
writer 3 is writing
writer 1 is writing
reader 0 is reading
writer 4 is writing
reader 4 is reading
reader 0 is reading
reader 2 is reading
writer 2 is writing
writer 0 is writing
writer 1 is writing
writer 2 is writing
reader 3 is reading
writer 4 is writing
reader 1 is reading
reader 4 is reading
writer 3 is writing
reader 2 is reading
reader 0 is reading

(b) Implement the solution for dining philosopher's problem.

CODE/OUTPUT:

```
#include<stdio.h>

#define n 4

int compltedPhilo = 0,i;

struct fork{

int taken;

}ForkAvil[n];

struct philosp{

int left;

int right;

}Philostatus[n];

void goForDinner(int philID){ //same like threads concept here
cases implemented

if(Philostatus[philID].left==10 &&
Philostatus[philID].right==10)

        printf("Philosopher %d completed his
dinner\n",philID+1);

//if already completed dinner

else if(Philostatus[philID].left==1 &&
Philostatus[philID].right==1){

         //if just taken two forks

         printf("Philosopher %d completed his
dinner\n",philID+1);

         philostatus[philID].left = Philostatus[philID].right =
10; //remembering that he completed dinner by assigning value
10

         int otherFork = philID-1;

         if(otherFork== -1)

              otherFork=(n-1);

         ForkAvil[philID].taken = ForkAvil[otherFork].taken =
0; //releasing forks

         printf("Philosopher %d released fork %d and fork
%d\n",philID+1,philID+1,otherFork+1);
```

```
          compltedPhilo++;

     }

     else if(Philostatus[philID].left==1 &&
Philostatus[philID].right==0){ //left already taken, trying for
right fork

               if(philID==(n-1)){

                    if(ForkAvil[philID].taken==0){ //KEY POINT
OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse
DIRECTION

                         ForkAvil[philID].taken =
Philostatus[philID].right = 1;

                         printf("Fork %d taken by philosopher
%d\n",philID+1,philID+1);

                         }else{

                         printf("Philosopher %d is waiting for
fork %d\n",philID+1,philID+1);

                         }

                         }else{ //except last philosopher case

                         int dupphilID = philID;

                    philID-=1;

                    if(philID== -1)

                         philID=(n-1);

                    if(ForkAvil[philID].taken == 0){

               ForkAvil[philID].taken =
Philostatus[dupphilID].right = 1;

                         printf("Fork %d taken by Philosopher
%d\n",philID+1,dupphilID+1);

          }else{

               printf("Philosopher %d is waiting for Fork
%d\n",dupphilID+1,philID+1);

                    }

               }

          }

          else if(Philostatus[philID].left==0){ //nothing taken
yet

                    if(philID==(n-1)){
```

```
                    if(ForkAvil[philID-1].taken==0){ //KEY POINT
OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse
DIRECTION

                         ForkAvil[philID-1].taken =
Philostatus[philID].left = 1;

                         printf("Fork %d taken by philosopher
%d\n",philID,philID+1);

               }else{

                    printf("Philosopher %d is waiting for fork
%d\n",philID+1,philID);

               }

          }else{ //except last philosopher case

               if(ForkAvil[philID].taken == 0){

                    ForkAvil[philID].taken =
Philostatus[philID].left = 1;

                    printf("Fork %d taken by Philosopher
%d\n",philID+1,philID+1);

                         }else{

                         printf("Philosopher %d is waiting for
Fork %d\n",philID+1,philID+1);

                    }

               }

          }else{}

}

int main(){

for(i=0;i<n;i++)

  ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;


while(compltedPhilo<n){

/* Observe here carefully, while loop will run until all
philosophers complete dinner

Actually problem of deadlock occur only thy try to take at same
time

This for loop will say that they are trying at same time. And
remaining status will print by go for dinner function

*/
```

```
for(i=0;i<n;i++)

          goForDinner(i);

printf("\nTill now num of philosophers completed dinner are
%d\n\n",compltedPhilo);

}

return 0;

}
```

```
Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 2 released fork 2 and fork 1
Fork 2 taken by Philosopher 3
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 2

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 3 released fork 3 and fork 2
Fork 3 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Fork 4 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 4 completed his dinner
Philosopher 4 released fork 4 and fork 3

Till now num of philosophers completed dinner are 4
```

```
Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 2 released fork 2 and fork 1
Fork 2 taken by Philosopher 3
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 2

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 3 released fork 3 and fork 2
Fork 3 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Fork 4 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 4 completed his dinner
Philosopher 4 released fork 4 and fork 3

Till now num of philosophers completed dinner are 4
```

(c) A pair of processes involved in exchanging a sequence of integers. The number of integers that can be produced and consumed at a time is limited to 100. Write a Program to implement the producer and consumer problem using POSIX semaphore for the above scenario.

CODE/OUTPUT:

```
#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>
#include<stdlib.h>
#define buffersize 100
pthread_mutex_t mutex;
pthread_t tidP[100],tidC[100];
sem_t full,empty;
int counter;
int buffer[buffersize];
```

```
void initialize()
{
     pthread_mutex_init(&mutex,NULL);
     sem_init(&full,1,0);
     sem_init(&empty,1,buffersize);
     counter=0;
}
void write(int item)
{
     buffer[counter++]=item;
}
int read()
{
     return(buffer[--counter]);
}
void * producer (void * param)
{
     int waittime,item,i;
     item=rand()%5;
     waittime=rand()%5;
     sem_wait(&empty);pthread_mutex_lock(&mutex);
     printf("\nProducer has produced item: %d\n",item);
     write(item);
     pthread_mutex_unlock(&mutex);
     sem_post(&full);
}
void * consumer (void * param)
{
     int waittime,item;
     waittime=rand()%5;
     sem_wait(&full);
     pthread_mutex_lock(&mutex);
     item=read();
     printf("\nConsumer has consumed item: %d\n",item);
     pthread_mutex_unlock(&mutex);
     sem_post(&empty);
}
int main()
{
     int n1,n2,i;
     initialize();
     printf("\nEnter the no of producers: ");
     scanf("%d",&n1);
     printf("\nEnter the no of consumers: ");
     scanf("%d",&n2);
     for(i=0;i<n1;i++)
          pthread_create(&tidP[i],NULL,producer,NULL);
```

```
    for(i=0;i<n2;i++)
         pthread_create(&tidC[i],NULL,consumer,NULL);
    for(i=0;i<n1;i++)
         pthread_join(tidP[i],NULL);
    for(i=0;i<n2;i++)
         pthread_join(tidC[i],NULL);
    //sleep(5);
    exit(0);

}
```

(d) Write a Program to implement banker's algorithm for Deadlock avoidance.

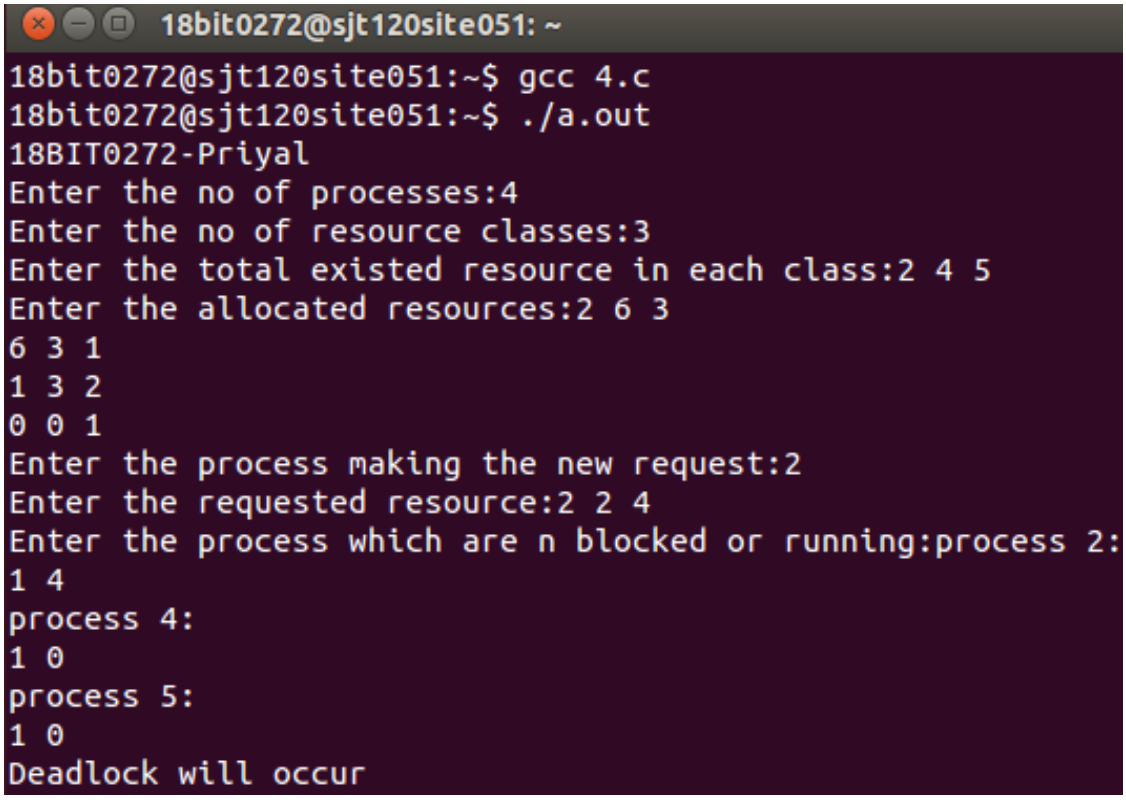
CODE/OUTPUT:

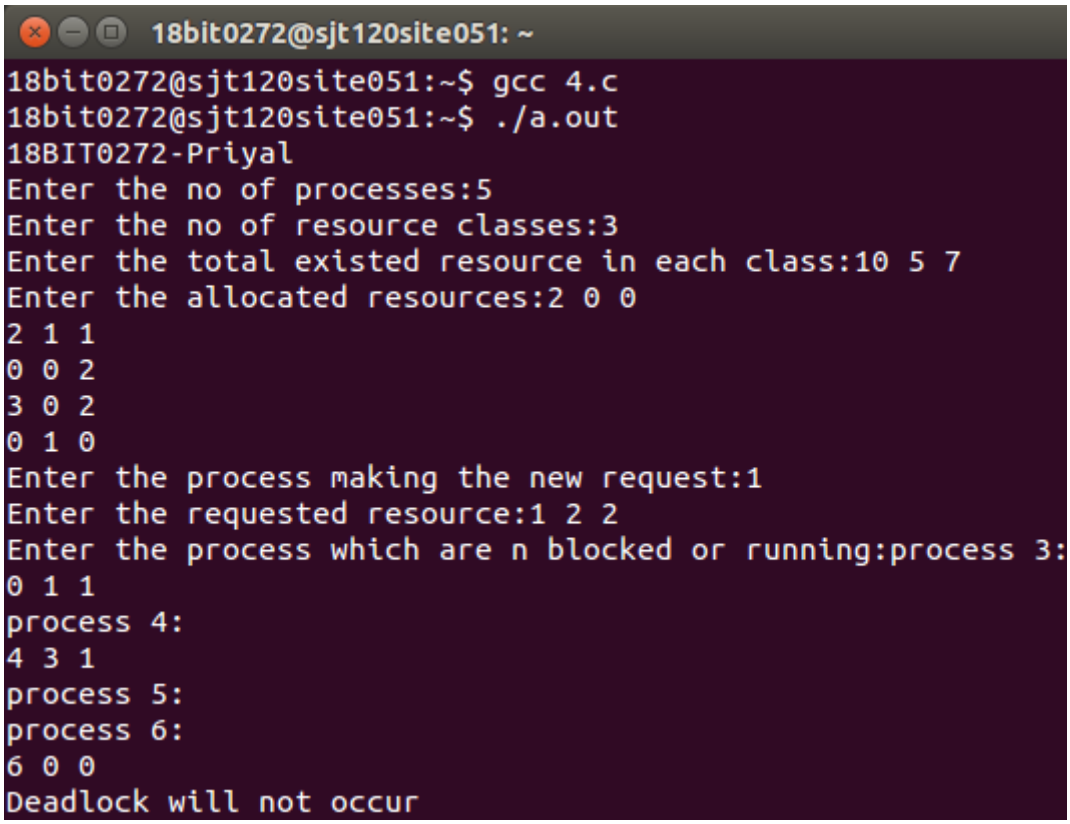
```
#include<stdio.h>
#include<stdlib.h>
void main()
{
printf("18BIT0272-Priyal\n");
int n,r,i,j,k,p,u=0,s=0,m;
int block[10],run[10],active[10],newreq[10];
int max[10][10],resalloc[10][10],resreq[10][10];
int totalloc[10],totext[10],simalloc[10];
//clrscr();
printf("Enter the no of processes:");
scanf("%d",&n);
printf("Enter the no of resource classes:");
scanf("%d",&r);
printf("Enter the total existed resource in each class:");
for(k=1; k<=r; k++)
scanf("%d",&totext[k]);
printf("Enter the allocated resources:");
for(i=1; i<=n; i++)
for(k=1; k<=r; k++)
scanf("%d",&resalloc[i][k]);
printf("Enter the process making the new request:");
scanf("%d",&p);
printf("Enter the requested resource:");
for(k=1; k<=r; k++)
scanf("%d",&newreq[k]);
printf("Enter the process which are n blocked or
running:");
for(i=1; i<=n; i++)
{
```

```
if(i!=p)
{
printf("process %d:\n",i+1);
scanf("%d%d",&block[i],&run[i]);
}
}
block[p]=0;
run[p]=0;
for(k=1; k<=r; k++)
{
j=0;
for(i=1; i<=n; i++)
{
totalloc[k]=j+resalloc[i][k];
j=totalloc[k];
}
}
for(i=1; i<=n; i++)
{
if(block[i]==1||run[i]==1)
active[i]=1;
else
active[i]=0;
}
for(k=1; k<=r; k++)
{
resalloc[p][k]+=newreq[k];
totalloc[k]+=newreq[k];
}
for(k=1; k<=r; k++)
```

```
{
if(totext[k]-totalloc[k]<0)
{
u=1;
break;
}
}
if(u==0)
{
for(k=1; k<=r; k++)
simalloc[k]=totalloc[k];
for(s=1; s<=n; s++)
for(i=1; i<=n; i++)
{
if(active[i]==1)
{
j=0;
for(k=1; k<=r; k++)
{
if((totext[k]-simalloc[k])<(max[i][k]-resalloc[i][k]))
{
j=1;
break;
}
}
}
if(j==0)
{
active[i]=0;
for(k=1; k<=r; k++)
```

```
simalloc[k]=resalloc[i][k];
}
}
m=0;
for(k=1; k<=r; k++)
resreq[p][k]=newreq[k];
printf("Deadlock will not occur\n\t");
}
else
{
for(k=1; k<=r; k++)
{
resalloc[p][k]=newreq[k];
totalloc[k]=newreq[k];
}
printf("Deadlock will occur\n\t");
}
}
```




18bit0272@sjt120site051: ~
18bit0272@sjt120site051:~$ gcc 4.c
18bit0272@sjt120site051:~$ ./a.out
18BIT0272-Priyal
Enter the no of processes:5
Enter the no of resource classes:3
Enter the total existed resource in each class:10 5 7
Enter the allocated resources:2 0 0
2 1 1
0 0 2
3 0 2
0 1 0
Enter the process making the new request:1
Enter the requested resource:1 2 2
Enter the process which are n blocked or running:process 3:
0 1 1
process 4:
4 3 1
process 5:
process 6:
6 0 0
Deadlock will not occur
