INSTITUTE OF TECHNOLOGY
& MANAGEMENT
GWALIOR • MP • INDIA

Laboratory Manual

Operating System

(IT-501)

For

Third Year Student
Department: Information Technology

Department of Information Technology

Vision Of IT Department

The Department of Information Technology envisions preparing technically competent problem solvers, researchers, innovators, entrepreneurs, and skilled IT professionals for the development of rural and backward areas of the country for the modern computing challenges.

Mission Of IT Department

- To offer valuable education through an effective pedagogical teaching-learning process.
- To shape technologically strong students for industry, research & higher studies.
- To stimulate the young brain entrenched with ethical values and professional behaviors for the progress of society.

Program Educational Objectives

- Our graduates will show management skills and teamwork to attain employers' objectives in their careers.
- Our graduates will explore the opportunities to succeed in research and/or higher studies.
- Our graduates will apply technical knowledge of Information Technology for innovation and entrepreneurship.
- Our graduates will evolve ethical and professional practices for the betterment of society.

Program Outcomes (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Outcomes

Operating System (IT 501)

Course Outcomes(Cos)		Bloom Level
CO1	Gain knowledge of history of operating systems. Understand design issues associated with operating systems	L2,L3
CO2	Gain knowledge of various process management concepts including scheduling, synchronization, deadlocks	L2
CO3	Understand concepts of memory management including virtual memory	L2,L3
CO4	Understand issues related to file system interface and implementation,disk management	L3,L4
CO5	Be familiar with protection and security mechanisms and Be familiar with various types of operating systems including Unix	L2

Course	Course Outcomes	P01	P02	P03	P04	P05	P06	P07	P08	P09	P010	P011	P012	PSO1	PSO2	PSO3
IT 501 .1	Gain knowledge of history of operating systems. Understand design issues associated with operating systems	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**
IT 501 .2	Gain knowledge of various process management concepts including scheduling,synchronization,deadlocks	**2**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**
IT 501 .3	Understand concepts of memory management including virtual memory	**2**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
IT 501 .4	Understand issues related to file system interface and implementation,diskmanagement	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
IT 501 .5	Be familiar with protection and security mechanisms and Be familiar with various types of operating systems including Unix	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**

List Of Programs

S.No	Lab Experiment	CO	BL	Page No
1	Program to implement FCFS CPU scheduling algorithm.	CO2	L3	1-4
2	Program to implement SJF CPU scheduling algorithm.	CO2	L3	5-7
3	Program to implement Priority CPU Scheduling algorithm.	CO2	L3	8-10
4	Program to implement Round Robin CPU scheduling algorithm.	CO2	L3	11-13
5	Program to implement classical inter process communication problem(producer consumer).	CO2	L3,L4	14-16
6	Program to implement classical inter process communication problem(Reader Writers).	CO2	L3	17-19
7	Program to implement classical inter process communication problem(Dining Philosophers).	CO2	L3	20-22
8	Program to implement FIFO page replacement algorithm.	CO2	L3	23-24
9	Program to implement LRU page replacement algorithm	CO3	L3,L4	25-27

1. Program to implement FCFS CPU scheduling algorithm.

Explanation/ Algorithm:

1. Start the process

2. Get the number of processes to be inserted

3. Get the value for burst time of each process from the user

4. Having allocated the burst time(bt) for individual processes , Start with the first process from it's initial position let other process to be in queue

5. Calculate the waiting time(wt) and turnaround time(tat) as

Wt(pi) = wt(pi-1) + tat(pi-1) (i.e wt of current process = wt of previous process + tat of previous process)

tat(pi) = wt(pi) + bt(pi) (i.e tat of current process = wt of current process + bt of current process)

6. Calculate the total and average waiting time and turnaround time

7. Display the values

8. Stop the process

Program: Without Arrival Time

```
#include<stdio.h>

void main()
{
   int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
   printf("Enter total number of processes");
   scanf("%d",&n);

   printf("\nEnter Process Burst Time\n");
   for(i=0;i<n;i++)
   {
      printf("P[%d]:",i+1);
      scanf("%d",&bt[i]);

   wt[0]=0;    //waiting time for first process is 0

   //calculating waiting time
   for(i=1;i<n;i++)
   {
```

```
    wt[i]=0;
    for(j=0;j<i;j++)
       wt[i]=wt[i]+bt[j];
}

printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

//calculating turnaround time
for(i=0;i<n;i++)
{
   tat[i]=bt[i]+wt[i];
   avwt+=wt[i];
   avtat+=tat[i];
   printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
}

avwt/=i;
avtat/=i;
printf("\n\nAverage Waiting Time:%d",avwt);
printf("\nAverage Turnaround Time:%d",avtat);
 getch();
}
```

Output

```
Enter total number of processes3
Enter Process Burst Time
P[1]:3
P[2]:4
P[3]:2
```

Process	Burst Time	Waiting Time	Turnaround Time
P[1]	3	0	3
P[2]	4	3	7
P[3]	2	7	9

```
Average Waiting Time:3
Average Turnaround Time:6
```

With Arrival Time

```
#include<stdio.h>

 void main(){
    int bt[10],at[10],tat[10],wt[10],ct[10];
    int n,i,j,k,sum=0;
    float ATAT=0,AWT=0;
    printf("Enter number of processes");
    scanf("%d",&n);
```

```
    printf("Enter arrival time and burst time for each process\n\n");
    for( i=0;i<n;i++)    {
                printf("Arrival time of process");
                scanf("%d",&at[i]);

                printf("Burst time of process");
                scanf("%d",&bt[i]);
                printf("\n");
    }
    for(j=0;j<n;j++)     {
                sum=sum+bt[j];
                ct[j]=+ct[j]+sum;
    }
    for( k=0;k<n;k++)
    {
                tat[k]=ct[k]-at[k];
                ATAT+=tat[k];
    }

                for(k=0;k<n;k++)
    {
                wt[k]=tat[k]-bt[k];
                AWT+=wt[k];
    }

    for( i=0;i<n;i++)
    {
                printf("P%d\t %d\t  %d\t %d\t %d\n",i+1,at[i],bt[i],tat[i],wt[i]);
    }

    printf("\n\nAverage Turnaround Time = %f\n",ATAT/n);
    printf("Average WT = %f\n\n",AWT/n);
    getch();
}
```

Output

Enter number of processes3

Enter arrival time and burst time for each process

Arrival time of process1

Burst time of process2

Arrival time of process2

Burst time of process3

Arrival time of process3

Burst time of process4

P1	1	2	833	831
P2	2	3	835	832
P3	3	4	838	834

2. Write a Program to implement SJF CPU scheduling algorithm.

Explanation/ Algorithm:

1. Start the process

2. Get the number of processes to be inserted

3. Sort the processes according to the burst time and allocate the one with shortest burst to execute first

4. If two processes have same burst length then FCFS scheduling algorithm is used

5. Calculate the total and average waiting time and turnaround time

6. Display the values

7. Stop the process

Program:

```
#include<stdio.h>
 int main()
{
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
   float avg_wt,avg_tat;
   clrscr();
   printf("Enter number of process:");
   scanf("%d",&n);

   printf("nEnter Burst Time:");
   for(i=0;i<n;i++)
   {
      printf("p%d:",i+1);
      scanf("%d",&bt[i]);
      p[i]=i+1;
   }

  //sorting of burst times
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(bt[j]<bt[pos])
```

```
            pos=j;
        }

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }

    wt[0]=0;

      for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }

    avg_wt=(float)total/n;
    total=0;

    printf("\n Process\t\ Burst Time\t Waiting Time\t ");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
     printf("\nP[%d]\t %d\t  %d\n",i+1,bt[i],wt[i],tat[i]);

  }
    avg_tat=(float)total/n;
    printf("\n Average Waiting Time=%f",avg_wt);
    printf("\n Average Turnaround Time=%f",avg_tat);
   getch();
}
```

Output

```
Enter number of process:4
Enter Burst Time:p1:2

p2:3
p3:4
p4:3
 Process  Burst Time      Waiting Time
P[1]        2          0
P[2]        3          2
```

P[3] 3 5
P[4] 4 8

Average Waiting Time=3.750000
Average Turnaround Time=6.750000

3. Program to implement Priority CPU Scheduling algorithm.

Explanation/ Algorithm:

1. Start the process

2. Get the number of processes to be inserted

3. Get the corresponding priority of processes

4. Sort the processes according to the priority and allocate the one with highest priority to execute first

5. If two process have same priority then FCFS scheduling algorithm is used

6. Calculate the total and average waiting time and turnaround time

7. Display the values

8. Stop the process

Program:

```
#include<stdio.h>
 void main()
  {
    int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
    clrscr();
    printf("Enter Total Number of Process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
      printf("\nP[%d]\n",i+1);
      printf("Burst Time:");
      scanf("%d",&bt[i]);
      printf("Priority:");
      scanf("%d",&pr[i]);
      p[i]=i+1;           //contains process number

    //sorting burst time, priority and process number in ascending order using selection
sort
    for(i=0;i<n;i++)
    {
```

```
  pos=i;
  for(j=i+1;j<n;j++)
  {
     if(pr[j]<pr[pos])
              pos=j;
  }

  temp=pr[i];
  pr[i]=pr[pos];
  pr[pos]=temp;

  temp=bt[i];
  bt[i]=bt[pos];
  bt[pos]=temp;

  temp=p[i];
  p[i]=p[pos];
  p[pos]=temp;
}

wt[0]=0;   //waiting time for first process is zero

//calculate waiting time
for(i=1;i<n;i++)
{
  wt[i]=0;
  for(j=0;j<i;j++)
     wt[i]+=bt[j];

  total+=wt[i];
}

avg_wt=total/n;      //average waiting time
total=0;

printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
  tat[i]=bt[i]+wt[i];     //calculate turnaround time
  total+=tat[i];
  printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

avg_tat=total/n;     //average turnaround time
printf("\n\nAverage Waiting Time=%d",avg_wt);
printf("\nAverage Turnaround Time=%d\n",avg_tat);

   getch();
   }
```

Output

Enter Total Number of Process:3

Enter Burst Time and Priority
P[1]
Burst Time:6
Priority:2

P[2]
Burst Time:8
Priority:1

P[3]
Burst Time:3
Priority:3

Process	Burst Time	Waiting Time	Turnaround Time
P[2]	8	0	8
P[1]	6	8	14
P[3]	3	14	17

Average Waiting Time=7
Average Turnaround Time=13

4. Program to implement Round Robin CPU scheduling algorithm.

Explanation/ Algorithm:

1. Start the process

2. Get the number of elements to be inserted

3. Get the value for burst time for individual processes

4. Get the value for time quantum

5. Make the CPU scheduler go around the ready queue allocating CPU to each process for the time interval specified

6. Make the CPU scheduler pick the first process and set time to interrupt after quantum. And after it's expiry dispatch the process

7. If the process has burst time less than the time quantum then the process is released by the CPU

8. If the process has burst time greater than time quantum then it is interrupted by the OS and the process is put to the tail of ready queue and the schedule selects next process from head of the queue

9. Calculate the total and average waiting time and turnaround time

10. Display the results

11. Stop the process

Program:

```
#include<stdio.h>

void main()
{

  int count,j,n,time,remain,flag=0,tq;
  int wt=0,tt=0,at[10],bt[10],rt[10];
  printf("Enter Total Process:\t ");
  scanf("%d",&n);
  remain=n;
  for(count=0;count<n;count++)
  {
    printf("Enter Arrival Time and Burst Time for Process Process Number %d
:",count+1);
    scanf("%d",&at[count]);
    scanf("%d",&bt[count]);
```

```
  rt[count]=bt[count];
 }
 printf("Enter Time Quantum:\t");
 scanf("%d",&tq);
 printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
 for(time=0,count=0;remain!=0;)
 {
  if(rt[count]<=tq && rt[count]>0)
  {
   time+=rt[count];
   rt[count]=0;
   flag=1;
  }
  else if(rt[count]>0)
  {
   rt[count]-=tq;
   time+=tq;
  }
  if(rt[count]==0 && flag==1)
  {
   remain--;
   printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
   wt+=time-at[count]-bt[count];
   tt+=time-at[count];
   flag=0;
  }
  if(count==n-1)
   count=0;
  else if(at[count+1]<=time)
   count++;
  else
   count=0;
 }
 printf("\nAverage Waiting Time= %f\n",wt*1.0/n);
 printf("Avg Turnaround Time = %f",tt*1.0/n);

getch();
 }
```

Output

```
Enter Total Process: 3
Enter Arrival Time and Burst Time for Process Process Number 1 : 0 5
Enter Arrival Time and Burst Time for Process Process Number 2 : 1 3
Enter Arrival Time and Burst Time for Process Process Number 3 : 2 8
Enter Time Quantum: 2
Process | Turnaround Time|Waiting Time

P[1]   |  5      |  0
P[2]   |  7      |  3
P[3]   |  15     |  7
```

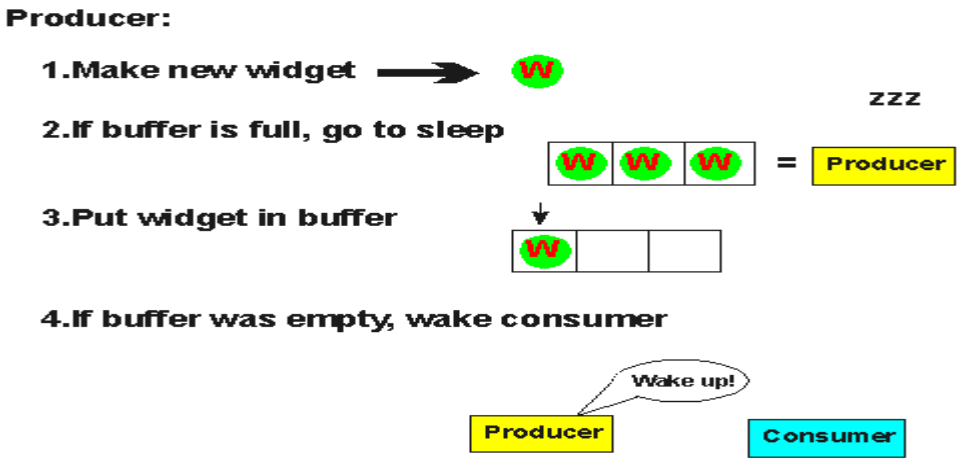
Average Waiting Time= 3.333333
Avg Turnaround Time = 9.000000

5. Program to implement classical inter process communication problem (producer consumer)

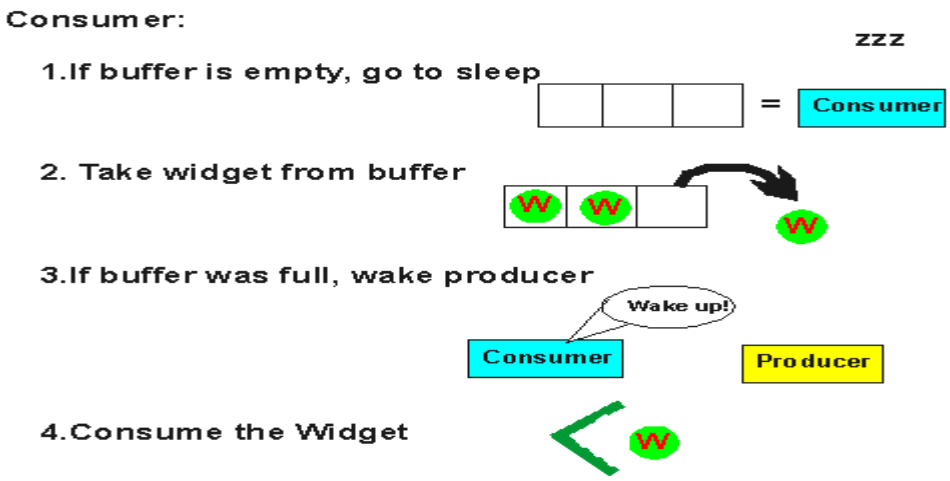
***Explanation*:**

The producer-consumer problem illustrates the need for synchronization in systems where many processes share a resource. In the problem, two processes share a fixed-size buffer. One process produces information and puts it in the buffer, while the other process consumes information from the buffer. These processes do not take turns accessing the buffer, they both work concurrently. Here in lies the problem. What happens if the producer tries to put an item into a full buffer? What happens if the consumer tries to take an item from an empty buffer?

In order to synchronize these processes, we will block the producer when the buffer is full, and we will block the consumer when the buffer is empty. So the two processes, Producer and Consumer, should work as follows:



(1) The producer must first create a new widget.
(2) Then, it checks to see if the buffer is full. If it is, the producer will put itself to sleep until the consumer wakes it up. A "wakeup" will come if the consumer finds the buffer empty.
(3) Next, the producer puts the new widget in the buffer. If the producer goes to sleep in step (2), it will not wake up until the buffer is empty, so the buffer will never overflow.
(4) Then, the producer checks to see if the buffer is empty. If it is, the producer assumes that the consumer is sleeping, an so it will wake the consumer. Keep in mind that between any of these steps, an interrupt might occur, allowing the consumer to run.



(1) The consumer checks to see if the buffer is empty. If so, the consumer will put itself to sleep until the producer wakes it up. A "wakeup" will occur if the producer finds the buffer empty after it puts an item into the buffer.
(2) Then, the consumer will remove a widget from the buffer. The consumer will never try to remove a widget from an empty buffer because it will not wake up until the buffer is full.
(3) If the buffer was full before it removed the widget, the consumer will wake the producer.
(4) Finally, the consumer will consume the widget. As was the case with the producer, an interrupt could occur between any of these steps, allowing the producer to run.

Program:

```
#include<stdio.h>
void main()
{
int buffer[10], bufsize, in, out, produce, consume, choice=0;
in = 0;
out = 0;
bufsize = 10;
while(choice !=3)
{
printf("\n1. Produce \t 2. Consume \t3. Exit");
printf("\nEnter your choice: ");
scanf("%d",&choice);
switch(choice) {
case 1: if((in+1)%bufsize==out)
printf("\nBuffer is Full");
else
{
}
Break;
printf("\nEnter the value: ");
scanf("%d", &produce);
buffer[in] = produce;
```

```
in = (in+1)%bufsize;
case 2: if(in == out)
printf("\nBuffer is Empty");
} } }
else
{
}
break;
consume = buffer[out];
printf("\nThe consumed value is %d", consume);
out = (out+1)%bufsize;
```

OUTPUT

```
1. Produce 2. Consume 3. Exit
Enter your choice: 2
Buffer is Empty
1. Produce 2. Consume 3. Exit
Enter your choice: 1
Enter the value: 100
1. Produce 2. Consume 3. Exit
Enter your choice: 2
The consumed value is 100
1. Produce 2. Consume 3. Exit
Enter your choice: 3
```

6. Program to implement classical inter process communication problem (Reader Writers).

Explanation:- The classical inter process communication problem known as the Reader-Writer problem involves multiple processes (or threads) accessing a shared resource concurrently. The scenario typically involves two types of processes: readers and writers.

Readers: These processes only read the shared resource and do not modify it.

Writers: These processes write to the shared resource.

The problem arises in scenarios where multiple readers can access the resource simultaneously, but only one writer can access the resource at a time, and no reader should be allowed to access the resource when a writer is writing to it.

To implement a solution to this problem, various synchronization mechanisms such as semaphores, mutexes, or monitors can be used to ensure mutual exclusion and coordination between readers and writers.

Program:

```
#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

sem_t mutex;

sem_t db;

int readercount=0;

pthread_t reader1,reader2,writer1,writer2;

void *reader(void *);

void *writer(void *);

main()

{

sem_init(&mutex,0,1);

sem_init(&db,0,1);
```

```
while(1)
{
pthread_create(&reader1,NULL,reader,"1");
pthread_create(&reader2,NULL,reader,"2");
pthread_create(&writer1,NULL,writer,"1");
pthread_create(&writer2,NULL,writer,"2");
}
}
void *reader(void *p)
{
printf("prevoius value %dn",mutex);
sem_wait(&mutex);
printf("Mutex acquired by reader %dn",mutex);
readercount++;
if(readercount==1) sem_wait(&db);
sem_post(&mutex);
printf("Mutex returned by reader %dn",mutex);
printf("Reader %s is Readingn",p);
//sleep(3);
sem_wait(&mutex);
printf("Reader %s Completed Readingn",p);
readercount--;
if(readercount==0) sem_post(&db);
sem_post(&mutex);
}
void *writer(void *p)
{
```

```
printf("Writer is Waiting n");

sem_wait(&db);

printf("Writer %s is writingn ",p);

sem_post(&db);

//sleep(2);

}
```

Output

```
previous value 1

Mutex acquired by reader 1
Mutex returned by reader 1
Reader 1 is Reading
previous value 0
Mutex acquired by reader 2
Mutex returned by reader 2
Reader 2 is Reading
previous value 0
Mutex acquired by writer 1
Writer 1 is writing
previous value 1
Mutex acquired by writer 2
Writer 2 is writing
```

7. Program to implement classical inter process communication problem (Dining Philosophers).

Explanation :- The Dining Philosophers problem is a classic synchronization problem that illustrates the challenges of avoiding deadlock and resource contention in a concurrent system. The problem involves a group of philosophers sitting at a table with a bowl of spaghetti in front of each. There are five philosophers and five chopsticks placed between them. Philosophers alternate between thinking and eating. To eat, a philosopher must pick up the two chopsticks adjacent to them. The challenge is to design a solution where each philosopher can eat without causing deadlock (wherein all philosophers are waiting for chopsticks held by others and cannot proceed) or starvation (wherein a philosopher never gets to eat).

Program:

```
int tph, philname[20], status[20], howhung, hu[20], cho;
main()
{
int i;
clrscr();
printf("\n\nDINING PHILOSOPHER PROBLEM");
printf("\nEnter the total no. of philosophers: ");
scanf("%d",&tph);
for(i=0;i<tph;i++)
{
philname[i] = (i+1);
status[i]=1;
}
printf("How many are hungry : ");
scanf("%d", &howhung);
if(howhung==tph)
{
}
else
{
printf("\nAll are hungry..\nDead lock stage will occur");
printf("\nExiting..");
for(i=0;i<howhung;i++)
{
printf("Enter philosopher %d position: ",(i+1));
scanf("%d", &hu[i]);
status[hu[i]]=2;
}
do
{
printf("1.One can eat at a time\t2.Two can eat at a time\t3.Exit\nEnter your choice:");
scanf("%d", &cho);
switch(cho)
```

```
{
case 1: one();
break;
case 2: two();
break;
case 3: exit(0);
default: printf("\nInvalid option..");
}
46
}while(1);
}
}
one()
{
int pos=0, x, i;
printf("\nAllow one philosopher to eat at any time\n");
for(i=0;i<howhung; i++, pos++)
{
}
}
two()
{
printf("\nP %d is granted to eat", philname[hu[pos]]);
for(x=pos;x<howhung;x++)
printf("\nP %d is waiting", philname[hu[x]]);
int i, j, s=0, t, r, x;
printf("\n Allow two philosophers to eat at same time\n");
for(i=0;i<howhung;i++)
{
for(j=i+1;j<howhung;j++)
{
if(abs(hu[i]-hu[j])>=1&& abs(hu[i]-hu[j])!=4)
{
printf("\n\ncombination %d \n", (s+1));
t=hu[i];
r=hu[j];
s++;
printf("\nP %d and P %d are granted to eat", philname[hu[i]],
philname[hu[j]]);
for(x=0;x<howhung;x++)
{
if((hu[x]!=t)&&(hu[x]!=r))
printf("\nP %d is waiting", philname[hu[x]]);
}
}
}
}
}
```

INPUT

DINING PHILOSOPHER PROBLEM

Enter the total no. of philosophers: 5
How many are hungry : 3
Enter philosopher 1 position: 2
Enter philosopher 2 position: 4
Enter philosopher 3 position: 5

OUTPUT

1.One can eat at a time 2.Two can eat at a time 3.Exit
Enter your choice: 1
Allow one philosopher to eat at any time
P 3 is granted to eat
P 3 is waiting
P 5 is waiting
P 0 is waiting
P 5 is granted to eat
P 5 is waiting
P 0 is waiting
P 0 is granted to eat
P 0 is waiting
47
1.One can eat at a time 2.Two can eat at a time 3.Exit
Enter your choice: 2
Allow two philosophers to eat at same time
combination 1
P 3 and P 5 are granted to eat
P 0 is waiting
combination 2
P 3 and P 0 are granted to eat
P 5 is waiting
combination 3
P 5 and P 0 are granted to eat
P 3 is waiting
1.One can eat at a time 2.Two can eat at a time 3.Exit
Enter your choice: 3

8. Program to implement FIFO page replacement algorithm.

Explanation / Algorithm

1. Start the process

2. Declare the size with respect to page length

3. Check the need of replacement from the page to memory

4. Check the need of replacement from old page to new page in memory

5. Forma queue to hold all pages

6. Insert the page require memory into the queue

7. Check for bad replacement and page fault

8. Get the number of processes to be inserted

9. Display the values

10. Stop the process

Program:

```
#include<stdio.h>
void main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
printf("\n enter the length of the Reference string:\n");
 scanf("%d",&n);
 printf("\n enter the reference string:\n");
 for(i=1;i<=n;i++)
  scanf("%d",&a[i]);
  printf("\n enter the number of Frames:");
  scanf("%d",&no);
  for(i=0;i<no;i++)
  frame[i]= -1;
   j=0;
  printf("\tref string\t page frames\n");
   for(i=1;i<=n;i++) {
   printf("%d\t\t",a[i]);
    avail=0; for(k=0;k<no;k++)
    if(frame[k]==a[i])
    avail=1;
    if (avail==0){
    frame[j]=a[i];
    j=(j+1)%no;
   count++;
```

```
for(k=0;k<no;k++)
printf("%d\t",frame[k]);
}
 printf("\n\n");
}
 printf("Page Fault Is %d",count);
getch();
}
```

Output

```
Enter the length of the Reference string:
10
Enter the reference string:
1 2 3 4 1 5 6 7 1 2
Enter the number of Frames: 3
ref string     page frames
1              1       -1      -1

2              1       2       -1

3              1       2       3

4              4       2       3

1              4       1       3

5              4       1       5

6              6       1       5

7              6       7       5

1              6       7       1

2              2       7       1
Page Fault Is 7
```

9. Program to implement LRU page replacement algorithm.

Explanation/ Algorithm

1. Start the process

2. Declare the size

3. Get the number of pages to be inserted

4. Get the value

5. Declare counter and stack

6. Select the least recently used page by counter value

7. Stack them according the selection.

8. Display the values

9. Stop the process

Program:

```
#include<stdio.h>
void main()
{
 int frames[10], temp[10], pages[10];
 int total_pages, m, n, position, k, l, total_frames;
 int a = 0, b = 0, page_fault = 0;
 clrscr();
 printf("\nEnter Total Number of Frames:\t");
 scanf("%d", &total_frames);
 for(m = 0; m < total_frames; m++)
 {
        frames[m] = -1;
 }
 printf("Enter Total Number of Pages:\t");
 scanf("%d", &total_pages);
 printf("Enter Values for Reference String:\n");
 for(m = 0; m < total_pages; m++)
 {
        printf("Value No.[%d]:\t", m + 1);
      scanf("%d", &pages[m]);
 }
 for(n = 0; n < total_pages; n++)
 {
      a = 0, b = 0;
```

```
for(m = 0; m < total_frames; m++)
{
 if(frames[m] == pages[n])
 {
  a = 1;
  b = 1;
  break;
  }
  }
  if(a == 0)
  {
   for(m = 0; m < total_frames; m++)
   {
    if(frames[m] == -1)
    {
   frames[m] = pages[n];
   b = 1;
   break;
    }
    }
    }
   if(b == 0)
  {
   for(m = 0; m < total_frames; m++)
   {
   temp[m] = 0;
   }
   for(k = n - 1, l = 1; l <= total_frames - 1; l++, k--)
   {
    for(m = 0; m < total_frames; m++)
    {
     if(frames[m] == pages[k])
     {
      temp[m] = 1;
      }
      }
      }
      for(m = 0; m < total_frames; m++{
      if(temp[m] == 0)
      position = m;
      }
      frames[position] = pages[n];
      page_fault++;
      }
      printf("\n");
      for(m = 0; m < total_frames; m++)
      {
      printf("%d\t", frames[m]);
        }
        }
```

```
printf("\nTotal Number of Page Faults:\t%d\n", page_fault);
getch();
}
```

```
Enter Total Number of Frames: 3
Enter Total Number of Pages: 12
Enter Values for Reference String:
Value No.[1]: 7
Value No.[2]: 0
Value No.[3]: 1
Value No.[4]: 2
Value No.[5]: 0
Value No.[6]: 3
Value No.[7]: 0
Value No.[8]: 4
Value No.[9]: 2
Value No.[10]: 3
Value No.[11]: 0
Value No.[12]: 3

 7   -1  -1
 7   0   -1
 7   0   1
 2   0   1
 2   0   1
 2   3   1
 2   3   0
 4   3   0
 4   2   0
 4   2   3
 0   2   3
 0   2   3

 Total Number of Page Faults: 7
```