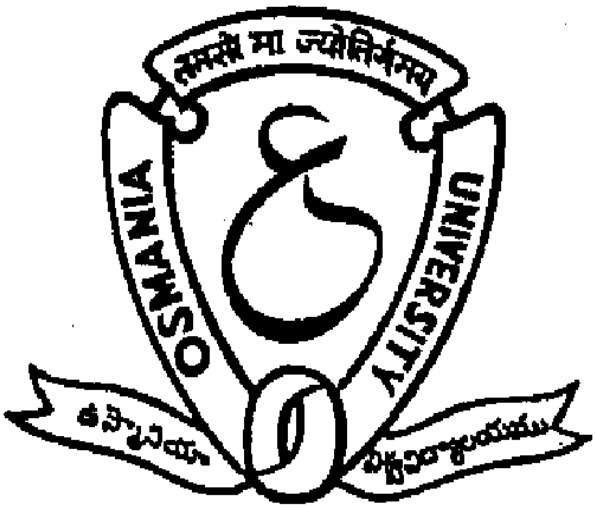
OPERATING SYSTEMS LAB

UNIVERSITY COLLEGE OF ENGINEERING

OSMANIA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that Owais Abdul Haseeb **bearing Roll no:** 1005-21-733067 **studying B.E V semester has successfully completed Operating Systems lab for the academic year 2023-2024.**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

TABLE OF CONTENTS

1. WAP FOR FCFS SCHEDULING ALGORITHM.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 int n,w[15],a[15],t[15],p[15],i,b[15],sumw=0,sumt=0;
 float avgw,avgt;
 printf("enter the no. of processes: ");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  printf("enter each process burst time: ");
  scanf("%d",&b[i]);
  printf("enter each process arrival time: ");
  scanf("%d",&a[i]);
 }
 w[0]=0;
 for(i=1;i<n;i++)
 {
  w[i]=w[i-1]+b[i-1];
 }
 for(i=0;i<n;i++)
 {
  t[i]=b[i]+w[i];
  sumw=sumw+w[i];
  sumt=sumt+t[i];
 }
 avgw=sumw/n;
 avgt=sumt/n;
 printf("the avg waiting time is: %f\n",avgw);
 printf("the avg turnaround time is: %f\n",avgt);
 return 0;
}
```

OUTPUT:

```
enter the no. of processes:
 3
enter each process burst time:
24
enter each process arrival time:
0
enter each process burst time:
3
enter each process arrival time:
0
enter each process burst time:
3
enter each process arrival time:
 0

the avg waiting time is:
17.000000

the avg turnaround time is:
27.000000
```

2. WAP FOR SJF NON-PREEMPTIVE SCHEDULING ALGORITHM.

```
#include<stdio.h>
int main()
{
 int p[15],bt[15],wt[15],i,wait,tat,t,n;
 float awt,atat;
 wait=tat=0;
 int j,temp;
 printf("enter no of processes: ");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  wt[i]=0;
  printf("enter process %d burst time: ",i+1);
  scanf("%d",&bt[i]);
  p[i]=i+1;
 }
 for(i=0;i<n;i++)
 {
  for(j=i+1;j<n;j++)
  {
    if(bt[i]>bt[j])
    {
      temp=bt[i];
      bt[i]=bt[j];
      bt[j]=temp;
      temp=p[i];
      p[i]=p[j];
      p[j]=temp;
    }
  }
 }
 printf("\tProcess\tburst time\twaiting time\tturnaround time\n");
 for(i=0;i<n;i++)
 {
  t=wt[i]+bt[i];
```

```
 printf("\t %d  \t   %d    \t    %d    \t       %d    \n",p[i],bt[i],wt[i],t);
 wt[i+1]=wt[i]+bt[i];
 wait=wait+wt[i];
 tat=tat+wt[i]+bt[i];
 }
awt=wait/(float)n;
atat=tat/(float)n;
printf("average waiting time: %f\n",awt);
printf("average turnaround time: %f\n",atat);
}
```

OUTPUT:

```
enter no of processes:
3
enter process 1 burst time:
 20
enter process 2 burst time:
30
enter process 3 burst time:
 10
      Process burst time     waiting time    turnaround time
       3         10             0                10
       1         20             10               30
       2         30             30               60
average waiting time:
13.333333
average turnaround time:
33.333332
```

3. WAP FOR SJF PREEMPTIVE SCHEDULING ALGORITHM.

```
#include<stdio.h>
int main()
{
  int i,j,n,time,sum_wait=0,sum_turnaround=0,smallest;
  int at[10],bt[10],rt[10],remain; //rt = remaining Time
  printf("Enter no of Processes : ");
  scanf("%d",&n);
  remain=n;
  for(i=0;i<n;i++)
  {
    printf("Enter arrival time, burst time for process p%d :",i+1);
    scanf("%d",&at[i]);
    scanf("%d",&bt[i]);
    rt[i]=bt[i];
  }
  bt[9]=9999;
  printf("\n\nProcess\t| waiting time\t| turnaround time\n");
  for(time=0;remain!=0;time++)
  {
    smallest=9;
    for(i=0;i<n;i++)
    {
      if(at[i]<=time && bt[i]<bt[smallest] && rt[i]>0)
      {
    smallest=i;
      }
    }
    rt[smallest]--;
    if(rt[smallest]==0)
    {
      remain--;
      printf("P[%d]\t|\t%d\t|\t%d\n",smallest+1,time+1-at[smallest]-bt[smallest],time+1-
at[smallest]);
      sum_wait+=time+1-at[smallest]-bt[smallest];
      sum_turnaround+=time+1-at[smallest];
    }
```

```
  }
  printf("\nAvg waiting time = %f\n",sum_wait*1.0/n);
  printf("Avg turnaround time = %f\n",sum_turnaround*1.0/n);
  return 0;
}
```

OUTPUT:

```
Enter no of Processes :
 5
Enter arrival time, burst time for process p1 :
0
10
Enter arrival time, burst time for process p2 :
1
3
Enter arrival time, burst time for process p3 :
2
2
Enter arrival time, burst time for process p4 :
3
4
Enter arrival time, burst time for process p5 :
4
5

Process | waiting time | turnaround time
P[3]    |      0       |        2
P[2]    |      2       |        5
P[4]    |      3       |        7
P[5]    |      6       |       11
P[1]    |      14      |       24

Avg waiting time = 5.000000
Avg turnaround time = 9.800000
```

4. WAP FOR PRIORITY (NON-PREEMPTIVE) SCHEDULING ALGORITHM.

```
#include<stdio.h>
int main()
{
 int n,i,j,w[15],t[15],pr[15],b[15],b1[15],p[15],pn[15],sumw=0,sumt=0,temp;
 float avgw,avgt;
 printf("enter the no. of processes: ");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  printf("enter process %d burst time and priority: ",i+1);
  scanf("%d %d",&b[i],&pr[i]);
  p[i]=pr[i];
 }
 w[0]=0; j=0;
 while(j<n)
 {
  for(i=1;i<=n;i++)
  {
   if(pr[j]==i)
   {
    b1[i-1]=b[j];
    pn[i-1]=j;
    p[i-1]=i;
   }
  }
  j++;
 }
 printf("Process\tburst time\tpriority\twaiting time\tturnaround time\n");
 for(i=0;i<n;i++)
 {
  w[i+1]=w[i]+b1[i];
  t[i]=b1[i]+w[i];
  sumw=sumw+w[i];
  sumt=sumt+t[i];
 printf("  %d    \t    %d    \t   %d   \t      %d    \t         %d      \n",pn[i]+1,b1[i],p[i],w[i],t[i]);
```

```
 }
 avgw=sumw/(float)n;
 avgt=sumt/(float)n;
 printf("the avg waiting time is: %f\n",avgw);
 printf("the avg turnaround time is: %f\n",avgt);
 return 0;
}
```

OUTPUT:

```
enter the no. of processes:
 5
enter process 1 burst time and priority:
10
3
enter process 2 burst time and priority:
 1
1
enter process 3 burst time and priority:
 2
4
enter process 4 burst time and priority:
 1
5
enter process 5 burst time and priority:
5
2
Process burst time      priority        waiting time    turnaround time
  2             1               1               0               1
  5             5               2               1               6
  1             10              3               6               16
  3             2               4               16              18
  4             1               5               18              19
the avg waiting time is: 8.200000
the avg turnaround time is: 12.000000
```

5. WAP FOR PRIORITY (PREEMPTIVE) SCHEDULING ALGORITHM.

```
#include<stdio.h>
int main()
{
  int i,j,n,time,sum_wait=0,sum_turnaround=0,smallest;
  int at[10],bt[10],pt[10],rt[10],remain; //rt = remaining Time
  printf("Enter no of Processes : ");
  scanf("%d",&n);
  remain=n;
  for(i=0;i<n;i++)
  {
    printf("Enter arrival time, burst time and priority for process p%d :",i+1);
    scanf("%d",&at[i]);
    scanf("%d",&bt[i]);
    scanf("%d",&pt[i]);
    rt[i]=bt[i];
  }
  pt[9]=11;
  printf("\n\nProcess\t| waiting time\t| turnaround time\n");
  for(time=0;remain!=0;time++)
  {
    smallest=9;
    for(i=0;i<n;i++)
    {
      if(at[i]<=time && pt[i]<pt[smallest] && rt[i]>0)
      {
    smallest=i;
      }
    }
    rt[smallest]--;
    if(rt[smallest]==0)
    {
      remain--;
      printf("P[%d]\t|\t%d\t|\t%d\n",smallest+1,time+1-at[smallest]-bt[smallest],time+1-
at[smallest]);
      sum_wait+=time+1 - at[smallest] - bt[smallest];
      sum_turnaround+=time+1 - at[smallest];
```

```
   }
  }
  printf("\nAvg waiting time = %f\n",sum_wait*1.0/n);
  printf("Avg turnaround time = %f\n",sum_turnaround*1.0/n);
  return 0;
}
```

OUTPUT:

```
Enter no of Processes :
 5
Enter arrival time, burst time and priority for process p1 :
0
10
3
Enter arrival time, burst time and priority for process p2 :
1
1
1
Enter arrival time, burst time and priority for process p3 :
2
2
4
Enter arrival time, burst time and priority for process p4 :
3
1
5
Enter arrival time, burst time and priority for process p5 :
4
5
2

Process | waiting time  | turnaround time
P[2]   |      0      |      1
P[5]   |      0      |      5
P[1]   |      6      |      16
P[3]   |      14     |      16
P[4]   |      15     |      16

Avg waiting time = 7.000000
Avg turnaround time = 10.800000
```

6. WAP FOR ROUND-ROBIN SCHEDULING ALGORITHM.

```
#include<stdio.h>
int main()
{
  int i,j,n,time,remain,flag=0,ts;
  int sum_wait=0,sum_turnaround=0,at[10],bt[10],rt[10];
  printf("Enter no of Processes : ");
  scanf("%d",&n);
  remain=n;
  for(i=0;i<n;i++)
  {
    printf("Enter arrival time and burst time for Process P%d :",i+1);
    scanf("%d",&at[i]);
    scanf("%d",&bt[i]);
    rt[i]=bt[i];
  }
  printf("Enter time slice :");
  scanf("%d",&ts);
  printf("\n\nProcess\t|Turnaround time|waiting time\n\n");
  for(time=0,i=0;remain!=0;)
  {
    if(rt[i]<=ts && rt[i]>0)
    {
      time+=rt[i];
      rt[i]=0;
      flag=1;
```

```
    }
    else if(rt[i]>0)
    {
      rt[i]-=ts;
      time+=ts;
    }
    if(rt[i]==0 && flag==1)
    {
      remain--;
      printf("P[%d]\t|\t%d\t|\t%d\n",i+1,time-at[i],time-at[i]-bt[i]);
      sum_wait+=time-at[i]-bt[i];
      sum_turnaround+=time-at[i];
      flag=0;
    }
    if(i==n-1)
      i=0;
    else if(at[i+1]<=time)
      i++;
    else
      i=0;
  }
  printf("\nAvg sum_wait = %f\n",sum_wait*1.0/n);
  printf("Avg sum_turnaround = %f\n",sum_turnaround*1.0/n);
  return 0;
}
```

OUTPUT:

```
Enter no of Processes :
 4
Enter arrival time and burst time for Process P1 :
0
6
Enter arrival time and burst time for Process P2 :
0
3
Enter arrival time and burst time for Process P3 :
0
1
Enter arrival time and burst time for Process P4 :
0
7
Enter time slice :
1


Process |Turnaround time|waiting time

P[3]   |      3      |      2
P[2]   |      9      |      6
P[1]   |      15     |      9
P[4]   |      17     |      10

Avg sum_wait = 6.750000
Avg sum_turnaround = 11.000000
```

7. WAP TO DEMONSTRATE FIRST FIT, BEST FIT AND WORST FIT.

```
#include<stdio.h>
int main()
{
 int i,j,temp,b[10],c[10],d[10],arr,n,ch,a;
 printf("\t\t FIRST FIT, BEST FIT, WORST FIT\n");
 printf("Enter the size of no. of blocks:");
 scanf("%d",&n);
 for(i=1;i<=n;i++)
 {
   printf("Enter the size of %d block:",i);
   scanf("%d",&b[i]);
   d[i]=c[i]=b[i];
 }
 printf("\nEnter the size of Arriving block:");
 scanf("%d",&arr);
 printf("\n1.First fit\n2.Best fit\n3.Worst fit\nEnter your choice:");
 scanf("%d",&ch);
 switch(ch)
 {
 case 1:
          for(i=1;i<=n;i++)
           {
             for(j=i+1;j<=n;j++)
             {
               if(d[i]>=d[j])
               {
                  temp=d[i];
                 d[i]=d[j];
                 d[j]=temp;
               }
             }
          }
          if(arr<=d[n]) {
          for(i=1;i<=n;i++)
          {
            if(b[i]>=arr)
```

```
            {
              printf("Arriving block is allocated to block %d.\n",i);
              break;
            }
            else
            continue;
            }}
            else printf("the arriving block cannot be allocated.\n");
            break;
case 2:
          for(i=1;i<=n;i++)
          {
            for(j=i+1;j<=n;j++)
           {
             if(b[i]>=b[j])
             {
               temp=b[i];
               b[i]=b[j];
               b[j]=temp;
             }
            }
          }
          if(arr<=b[n]){
           for(i=1;i<=n;i++)
           {
             if(b[i]>=arr)
             {
               a=b[i];
               break;
             }
             else
             continue;
           }
          for(i=1;i<=n;i++)
            {
              if(c[i]==a)
              {
               printf("Arriving block is allocated to block %d.\n",i);
             }
             }}
```

```
            else printf("the arriving block cannot be allocated.\n");
            break;
case 3:
          for(i=1;i<=n;i++)
            {
             for(j=i+1;j<=n;j++)
             {
              if(b[i]>=b[j])
              {
                 temp=b[i];
                 b[i]=b[j];
                 b[j]=temp;
               }
             }
            }
          if(arr<=b[n]){
            for(i=n;i>=1;i--)
            {
             if(b[i]>=arr)
             {
               a=b[i];
               break;
             }
             else
             continue;
            }
          for(i=1;i<=n;i++)
            {
             if(c[i]==a)
            {
               printf("Arriving block is allocated to block %d.\n",i);
            }
            }}
            else printf("the arriving block cannot be alocated.\n");
            break;
default:
          printf("Enter the valid choice:");
          }
return 0;
}
```

OUTPUT:

```
                FIRST FIT, BEST FIT, WORST FIT
Enter the size of no. of blocks:
5
Enter the size of 1 block:
18
Enter the size of 2 block:
10
Enter the size of 3 block:
56
Enter the size of 4 block:
2
Enter the size of 5 block:
24

Enter the size of Arriving block:
12

1. First fit
2 .Best fit
3. Worst fit
Enter your choice:
1
Arriving block is allocated to block 1.
```

```
            FIRST FIT, BEST FIT, WORST FIT
Enter the size of no. of blocks:
5
Enter the size of 1 block:
18
Enter the size of 2 block:
10
Enter the size of 3 block:
56
Enter the size of 4 block:
2
Enter the size of 5 block:
24

Enter the size of Arriving block:
1
```

1. First fit
2. Best fit
3. Worst fit
Enter your choice:
2
Arriving block is allocated to block 4.

FIRST FIT, BEST FIT, WORST FIT
Enter the size of no. of blocks:
5
Enter the size of 1 block:
18
Enter the size of 2 block:
10
Enter the size of 3 block:
56
Enter the size of 4 block:
2
Enter the size of 5 block:
24

Enter the size of Arriving block:
20

1. First fit
2. Best fit
3. Worst fit
Enter your choice:
3
Arriving block is allocated to block 3.

8. WAP FOR FIFO PAGE REPLACEMENT ALGORITHM.

```
#include<stdio.h>
int main()
{
int i,j,n,ref[50],frame[10],fno,k,avail,count=0;
printf("enter total no of pages in reference string: ");
scanf("%d",&n);
printf("enter the reference string:\n");
for(i=1;i<=n;i++)
  scanf("%d",&ref[i]);
printf("enter no of frames: ");
scanf("%d",&fno);

for(i=0;i<fno;i++)
  frame[i]=-1;
j=0;
printf("ref string | page frames\n");
printf("           | f[1] f[2] f[3]\n");
for(i=1;i<=n;i++)
{
 printf("   %d          ",ref[i]);
 avail=0;
 for(k=0;k<fno;k++)
 {
   if(frame[k]==ref[i])
     avail=1;
 }
 if(avail==0)
 {
   frame[j]=ref[i];
   j=(j+1)%fno;
   count++;
   for(k=0;k<fno;k++)
      printf("%d    ",frame[k]);
 }
 printf("\n");
}
```

```
printf("no of page faults: %d\n",count);
return 0;
}
```

OUTPUT:

```
enter total no of pages in reference string:
 20
enter the reference string:
7  0  1  2  0  3  0  4  2  3  0  3  2  1  2  0  1  7  0  1
enter no of frames:
 3
        ref string |  page frames
                   |  f[1]   f[2]   f[3]
          7            7    -1     -1
          0            7     0     -1
          1            7     0      1
          2            2     0      1
          0
          3            2     3      1
          0            2     3      0
          4            4     3      0
          2            4     2      0
          3            4     2      3
          0            0     2      3
          3
          2
          1            0     1      3
          2            0     1      2
          0
          1
          7            7     1      2
          0            7     0      2
          1            7     0      1

no of page faults: 15
```

9. WAP FOR OPTIMAL PAGE REPLACEMENT ALGORITHM.

```
#include<stdio.h>
int main()
{
int fr[10],p[50],i,j,n;
int max,found=0,lg[10],index,k,l,flag1=0,flag2=0,pf=0,frsize;
printf("total no of pages in reference string: ");
scanf("%d",&n);
printf("enter reference string:\n");
for(i=0;i<n;i++)
 scanf("%d",&p[i]);
printf("no of frames: ");
scanf("%d",&frsize);
for(i=0;i<frsize;i++)
{
fr[i]=-1;
}
for(j=0;j<n;j++)
{
  flag1=0;flag2=0;
  for(i=0;i<frsize;i++)
  {
    if(fr[i]==p[j])
    {
      flag1=1;flag2=1;break;
    }
  }
  if(flag1==0)
  {
    for(i=0;i<frsize;i++)
    {
      if(fr[i]==-1)
      {
        fr[i]=p[j];
        flag2=1;
        pf++;
        break;
```

```
    }
  }
}
if(flag2==0)
{
  for(i=0;i<frsize;i++)
    {lg[i]=0;}
  for(i=0;i<frsize;i++)
  {
    for(k=j+1;k<n;k++)
    {
      if(fr[i]==p[k])
      {
       lg[i]=k-j;
       break;
      }
    }
  }
  found=0;
  for(i=0;i<frsize;i++)
  {
    if(lg[i]==0)
    {
      index=i;found=1;break;
    }
  }
  if(found==0)
  {
    max=lg[0];
    index=0;
    for(i=1;i<frsize;i++)
    {
      if(max<lg[i])
      {
        max=lg[i];index=i;
      }
    }
   }
   fr[index]=p[j];
   pf++;
```

```
   }
  printf("\n");
  for(i=0;i<frsize;i++)
  printf("\t%d",fr[i]);
}
 printf("\nno of page faults:%d\n",pf);
return 0;
}
```

OUTPUT:

```
total no of pages in reference string:
20
enter reference string:
7  0  1  2  0  3  0  4  2  3  0  3  2  1  2  0  1  7  0  1
no of frames:
3

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

          no of page faults:  9
```

10. WAP FOR LRU PAGE REPLACEMENT ALGORITHM.

```
#include<stdio.h>
int main()
{
int fr[10],p[50],i,j,n;
int max,found=0,lg[10],index,k,l,flag1=0,flag2=0,pf=0,frsize;
printf("total no of pages in reference string: ");
scanf("%d",&n);
printf("enter reference string:\n");
for(i=0;i<n;i++)
 scanf("%d",&p[i]);
printf("no of frames: ");
scanf("%d",&frsize);
for(i=0;i<frsize;i++)
{
fr[i]=-1;
}
for(j=0;j<n;j++)
{
  flag1=0;flag2=0;
  for(i=0;i<frsize;i++)
  {
    if(fr[i]==p[j])
    {
      flag1=1;flag2=1;break;
```

```
  }
}
if(flag1==0)
{
  for(i=0;i<frsize;i++)
  {
    if(fr[i]==-1)
    {
      fr[i]=p[j];
      flag2=1;
      pf++;
      break;
    }
  }
}
if(flag2==0)
{
  for(i=0;i<frsize;i++)
    {lg[i]=0;}
  for(i=0;i<frsize;i++)
  {
    for(k=j-1;k>=0;k--)
    {
      if(fr[i]==p[k])
      {
       lg[i]=k-j;
       break;
```

```
    }
   }
  }
  found=0;
  for(i=0;i<frsize;i++)
  {
   if(lg[i]==0)
   {
    index=i;found=1;break;
   }
  }
  if(found==0)
  {
   max=lg[0];
   index=0;
   for(i=1;i<frsize;i++)
   {
    if(max>lg[i])
    {
     max=lg[i];index=i;
    }
   }
  }
  fr[index]=p[j];
   pf++;
 }
printf("\n");
```

```
  for(i=0;i<frsize;i++)
  printf("\t%d",fr[i]);
}
 printf("\nno of page faults:%d\n",pf);
return 0;
}
```

OUTPUT:

total no of pages in reference string:

20

enter reference string:

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

no of frames:

3

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

1	3	2
1	3	2
1	0	2
1	0	2
1	0	7
1	0	7
1	0	7

no of page faults: 12

11. PRODUCER-CONSUMER PROBLEM USING SEMAPHORES.

```
#include<stdio.h>
#include<pthread.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>
#include<semaphore.h>
sem_t empty,full,mutex;
char buf[10];
void *producer(void* arg)
{
  int i;
 //printf("inside producer\n");
  for(i=0;i<10;i++)
 {
  sem_wait(&empty);
  sem_wait(&mutex);
  buf[i]=i;
  printf("item produced is %d\n",buf[i]);
  sem_post(&mutex);
  sem_post(&full);
  sleep(1);
 }
```

```
 pthread_exit("producer\n");
}
void *consumer(void* arg)
{
  int j;
 //printf("inside consumer\n");
  for(j=0;j<10;j++)
  {
    sem_wait(&full);
    sem_wait(&mutex);
    j=buf[j];
    printf("consumed item is:%d\n",buf[j]);
    sem_post(&mutex);
    sem_post(&empty);
    sleep(5);
  }
 pthread_exit("consumer\n");
}

int main()
{
  pthread_t pid1,pid2;
  sem_init(&empty,0,10);
  sem_init(&full,0,0);
  sem_init(&mutex,1,1);
```

```
  void *status;
  pthread_create(&pid1,NULL,producer,NULL);
  pthread_create(&pid2,NULL,consumer,NULL);
  pthread_join(pid1,&status);
  printf("the exited status of 1st is %s\n",(char*)status);
  pthread_join(pid2,&status);
  printf("the exited status %s\n",(char*)status);
  return 0;
}
```

COMPILE:

```
gcc -pthread pc2.c
```

OUTPUT:

```
item produced is 0
consumed item is:0
item produced is 1
item produced is 2
item produced is 3
item produced is 4
consumed item is:1
item produced is 5
item produced is 6
```

item produced is 7

item produced is 8

item produced is 9

consumed item is:2

the exited status of 1st is producer

consumed item is:3

consumed item is:4

consumed item is:5

consumed item is:6

ctrl + z

12. DINNING-PHILOSOPHERS PROBLEM USING SEMAPHORES.

```
#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>
#define N 5
#define THINKING 0
#define HUNGRY 1
#define EATING 2
#define LEFT (ph_num+4)%N
#define RIGHT (ph_num+1)%N
sem_t mutex;
sem_t S[N];
void * philospher(void *num);
void take_fork(int);
void put_fork(int);
void test(int);
int state[N];
int phil_num[N]={0,1,2,3,4};
int main()
{
   int i;
   pthread_t thread_id[N];
   sem_init(&mutex,0,1);
```

```
    for(i=0;i<N;i++)
        sem_init(&S[i],0,0);
    for(i=0;i<N;i++)
    {
        pthread_create(&thread_id[i],NULL,philospher,&phil_num[i]);
        printf("Philosopher %d is thinking\n",i+1);
    }
    for(i=0;i<N;i++)
        pthread_join(thread_id[i],NULL);
}
void *philospher(void *num)
{
    while(1)
    {
        int *i = num;
        sleep(1);
        take_fork(*i);
        sleep(0);
        put_fork(*i);
    }
}
void take_fork(int  ph_num)
{
    sem_wait(&mutex);
    state[ph_num] = HUNGRY;
```

```
    printf("Philosopher %d is Hungry\n",ph_num+1);
    test(ph_num);
    sem_post(&mutex);
    sem_wait(&S[ph_num]);
    sleep(1);
}
void test(int ph_num)
{
    if (state[ph_num] == HUNGRY && state[LEFT] != EATING && state[RIGHT] !=
EATING)
    {
        state[ph_num] = EATING;
        sleep(2);
        printf("Philosopher %d takes fork %d and %d\n",ph_num+1,LEFT+1,ph_num+1);
        printf("Philosopher %d is Eating\n",ph_num+1);
        sem_post(&S[ph_num]);
    }
}
void put_fork(int ph_num)
{
    sem_wait(&mutex);
    state[ph_num] = THINKING;
    printf("Philosopher %d putting fork %d and %d down\n",ph_num+1,LEFT+1,ph_num+1);
    printf("Philosopher %d is thinking\n",ph_num+1);
    test(LEFT);
```

```
    test(RIGHT);
    sem_post(&mutex);
}
```

COMPILE:

```
gcc -pthread dp.c
```

OUTPUT:

```
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 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 4 is Hungry
Philosopher 5 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
```

ctrl + z

13. READER-WRITER'S PROBLEM USING SEMAPHORES.

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>
int data=0,rdcnt=0;
sem_t mutex,writeblock;
void * reader(void * no)
{
   printf("\n\tReader %d is executing ",(int)no);
   sem_wait(&mutex);
   printf("\n\tWait  to mutex by %d reader",(int)no);
   rdcnt++;
   if(rdcnt==1)
   {
      sem_wait(&writeblock);
      printf("\n\tWait  to writerblock by %d reader",(int)no);
   }
   printf("\n\t***Reader %d read data = %d ",(int)no,data);
   if(rdcnt==1)
   {
      sem_post(&writeblock);
      printf("\n\tSignal to writerblock by %d reader",(int)no);
```

```
    }
    sem_post(&mutex);
    printf("\n\tSignal to mutex by %d reader\n",(int)no);
}
void * writer(void * no)
{
    printf("\n\tWriter %d is executing ",(int)no);
    sem_wait(&writeblock);
    printf("\n\tWait  to writerblock by %d writer",(int)no);
    data+=5;
    printf("\n\t***Writer %d write data = %d ",(int)no,data);
    sem_post(&writeblock);
    printf("\n\tSignal to writer by %d writer\n",(int)no);
}
int main()
{
    int no,i,ir=0,iw=0,ch;
    sem_init(&mutex,0,1);
    sem_init(&writeblock,0,1);
    printf("\nEnter no of readers and writers to create : ");
    scanf("%d",&no);
    pthread_t r[no],w[no];
    do
    {
        printf("\n\t1.Reader\n\t2.Writer\n\t3.terminate\n\tYour choice : ");
```

```
        scanf("%d",&ch);
        switch(ch)
        {
           case 1: pthread_create(&r[ir],NULL,reader,(void *)ir);
                   pthread_join(r[ir++],NULL);
                   break;
           case 2: pthread_create(&w[iw],NULL,writer,(void *)iw);
                   pthread_join(w[iw++],NULL);
                   break;
        }
    }while(ch!=3);

    sem_destroy(&mutex);
    sem_destroy(&writeblock);
}
```

COMPILE:

```
gcc -pthread rw1.c
```

OUTPUT :

```
Enter no of readers and writers to create : 2

    1.Reader
    2.Writer
    3.terminate
    Your choice : 1
```

Reader 0 is executing

Wait to mutex by 0 reader

Wait to writerblock by 0 reader

***Reader 0 read data = 0

Signal to writerblock by 0 reader

Signal to mutex by 0 reader

1.Reader

2.Writer

3.terminate

Your choice : 2

Writer 0 is executing

Wait to writerblock by 0 writer

***Writer 0 write data = 5

Signal to writer by 0 writer

1.Reader

2.Writer

3.terminate

Your choice : 3

14. WAP TO DEMONSTRATE ARITHMATIC OPERATORS USING SHELL SCRIPTING

```
#!/bin/bash

# Shell script to demonstrate arithmetic operators

# Take user input for two numbers
echo "Enter the first number: "
read num1

echo "Enter the second number: "
read num2

# Perform arithmetic operations
sum=$((num1 + num2))
difference=$((num1 - num2))
product=$((num1 * num2))
quotient=$((num1 / num2))
remainder=$((num1 % num2))

# Display the results
echo "Sum: $sum"
echo "Difference: $difference"
echo "Product: $product"
echo "Quotient: $quotient"
echo "Remainder: $remainder"
```

OUTPUT :

```
Enter the first number:
10
Enter the second number:
3
Sum: 13
Difference: 7
Product: 30
Quotient: 3
Remainder: 1
```

15. WAP TO DEMONSTRATE DO-WHILE LOOP USING SHELL SCRIPTING

```
#!/bin/bash

# Shell script to demonstrate a do-while loop

# Initialize a counter
counter=1

# Set the maximum number of iterations
max_iterations=5

# Start the do-while loop
while [ $counter -le $max_iterations ]; do
   # Display the current iteration
   echo "Iteration: $counter"

   # Increment the counter
   ((counter++))
done
```

OUTPUT :

```
Iteration: 1

Iteration: 2

Iteration: 3

Iteration: 4

Iteration: 5
```

16. WAP TO DEMONSTRATE IF CONDITION USING SHELL SCRIPTING

```
#!/bin/bash

# Shell script to demonstrate the if condition

# Take user input
echo "Enter a number: "
read number

# Check if the number is greater than 10
if [ $number -gt 10 ]; then
   echo "The number is greater than 10."
elif [ $number -eq 10 ]; then
   echo "The number is equal to 10."
else
   echo "The number is less than 10."
Fi
```

OUTPUT :

```
Enter a number:
15
The number is greater than 10.
```

17. WAP TO DEMONSTRATE CASE CONDITION USING SHELL SCRIPTING

```
#!/bin/bash

# Shell script to demonstrate the case statement

# Take user input

echo "Enter a fruit name: "

read fruit

# Use the case statement to check different fruit names

case $fruit in

   "apple")

      echo "You selected an apple.” ;;

   "banana")

      echo "You selected a banana.";;

   "orange" | "citrus")

      echo "You selected an orange or citrus fruit.";;

   *)

      echo "You selected an unknown fruit.";;

esac
```

OUTPUT :

```
Enter a fruit name:

apple

You selected an apple.
```

17. WAP TO DEMONSTRATE LOGICAL OPERATORS USING SHELL SCRIPTING

```
#!/bin/bash

# Shell script to demonstrate logical operators

# Take user input for two numbers

echo "Enter the first number: "

read num1

echo "Enter the second number: "

read num2

# Check conditions using logical operators

if [ $num1 -gt 0 ] && [ $num2 -gt 0 ]; then

   echo "Both numbers are greater than 0."

else

   echo "At least one of the numbers is not greater than 0."

fi

if [ $num1 -eq 0 ] || [ $num2 -eq 0 ]; then

   echo "At least one of the numbers is equal to 0."

else

   echo "Neither of the numbers is equal to 0."

fi

if [ ! $num1 -eq $num2 ]; then

   echo "The two numbers are not equal."

else

   echo "The two numbers are equal."

Fi
```

OUTPUT :

Enter the first number:

5

Enter the second number:

8

Both numbers are greater than 0.

Neither of the numbers is equal to 0.

The two numbers are not equal.