Ex.No:5	CPU SCHEDULING ALGORITHMS
	PRIORITY

AIM:

To write a C program for implementation of Priority scheduling algorithms.

ALGORITHM:

Step 1: Inside the structure declare the variables.
Step 2: Declare the variable i,j as integer, totwtime and totttime is equal to zero.
Step 3: Get the value of ‘n’ assign p and allocate the memory.
Step 4: Inside the for loop get the value of burst time and priority.
Step 5: Assign wtime as zero .
Step 6: Check p[i].pri is greater than p[j].pri .
Step 7: Calculate the total of burst time and waiting time and assign as turnaround time.
Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<stdio.h>
#include<stdlib.h>
typedef struct
{
int pno;
int pri;
int pri;
int btime;
int wtime;
}sp;
int main()
{
int i,j,n;
int tbm=0,totwtime=0,totttime=0;
sp *p,t;
printf("\n PRIORITY SCHEDULING.\n");
printf("\n enter the no of process....\n");
scanf("%d",&n);
p=(sp*)malloc(sizeof(sp));
printf("enter the burst time and priority:\n");
for(i=0;i<n;i++)
{
printf("process%d:",i+1);
scanf("%d%d",&p[i].btime,&p[i].pri);
p[i].pno=i+1;
```



```
p[i].wtime=0;
}
for(i=0;i<n-1;i++)
for(j=i+1;j<n;j++)
{
if(p[i].pri>p[j].pri)
{
t=p[i];
p[i]=p[j];
p[j]=t;
}
}
printf("\n process\tbursttime\twaiting time\tturnaround time\n");
for(i=0;i<n;i++)
{
totwtime+=p[i].wtime=tbm;
tbm+=p[i].btime;
printf("\n%d\t\t%d",p[i].pno,p[i].btime);
printf("\t%d\t\t%d",p[i].wtime,p[i].wtime+p[i].btime);
}
totttime=tbm+totwtime;
printf("\n total waiting time:%d",totwtime);
printf("\n average waiting time:%f",(float)totwtime/n);
printf("\n total turnaround time:%d",totttime);
printf("\n avg turnaround time:%f",(float)totttime/n);
}
```

OUTPUT:

Ex.No:5.b	CPU SCHEDULING ALGORITHMS
	ROUND ROBIN SCHEDULING

AIM:

To write a C program for implementation of Round Robin scheduling algorithms.

ALGORITHM:

Step 1: Inside the structure declare the variables.
Step 2: Declare the variable i,j as integer, totwtime and totttime is equal to zero.
Step 3: Get the value of ‘n’ assign p and allocate the memory.
Step 4: Inside the for loop get the value of burst time and priority and read the time quantum.
Step 5: Assign wtime as zero.
Step 6: Check p[i].pri is greater than p[j].pri .
Step 7: Calculate the total of burst time and waiting time and assign as turnaround time.
Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
struct rr
{
int pno,btime,sbtime,wtime,lst;
}p[10];
int main()
{
int pp=-1,ts,flag,count,ptm=0,i,n,twt=0,totttime=0;
printf("\n round robin scheduling.....");
printf("enter no of processes:");
scanf("%d",&n);
printf("enter the time slice:");
scanf("%d",&ts);
printf("enter the burst time");
for(i=0;i<n;i++)
{
printf("\n process%d\t",i+1);
scanf("%d",&p[i].btime);
p[i].wtime=p[i].lst=0;
p[i].pno=i+1;
p[i].sbtime=p[i].btime;
}
```



```
printf("scheduling....\n");
do
{
flag=0;
for(i=0;i<n;i++)
{
count=p[i].btime;
if(count>0)
{
flag=-1;
count=(count>=ts)?ts:count;
printf("\n process %d",p[i].pno);
printf("from%d",ptm);
ptm+=count;
printf("to%d",ptm);
p[i].btime-=count;
if(pp!=i)
{
pp=i;
p[i].wtime+=ptm-p[i].lst-count;
p[i].lst=ptm;
}
}
```

OUTPUT:

Ex.No:5.c	CPU SCHEDULING ALGORITHMS
	FCFS

AIM:

To write a C program for implementation of FCFS and SJF scheduling algorithms.

ALGORITHM:

Step 1: Inside the structure declare the variables.
Step 2: Declare the variable i,j as integer,totwtime and totttime is equal to zero.
Step 3: Get the value of ‘n’ assign pid as I and get the value of p[i].btime.
Step 4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time and turnaround time.
Step 5: Calculate total wait time and total turnaround time by dividing by total number of process.
Step 6: Print total wait time and total turnaround time.
Step 7: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
 struct fcfs
{
int pid;
int btime;
int wtime;
int ttime;
}
p[10];
int main()
{
int i,n;
int towtwtime=0,totttime=0;
printf("\n fcfs scheduling...\n");
printf("enter the no of process");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p[i].pid=1;
printf("\n burst time of the process");
scanf("%d",&p[i].btime);
}
```



```
p[0].wtime=0;
p[0].ttime=p[0].btime;
totttime+=p[i].ttime;
for(i=0;i<n;i++)
{
p[i].wtime=p[i-1].wtime+p[i-1].btim
p[i].ttime=p[i].wtime+p[i].btime;
totttime+=p[i].ttime;
towtwtime+=p[i].wtime;
}
for(i=0;i<n;i++)
{{
printf("\n waiting time for process");
printf("\n turn around time for process");
printf("\n");
}}
printf("\n total waiting time :%d", totwtime );
printf("\n average waiting time :%f",(float)totwtime/n);
printf("\n total turn around time :%d",totttime);
printf("\n average turn around time: :%f",(float)totttime/n);
}
```

OUTPUT:

Ex.No:5.d	CPU SCHEDULING ALGORITHMS
	SJF SCHEDULING

AIM:

To write a C program for implementation of SJF scheduling algorithms.

ALGORITHM:

Step 1: Inside the structure declare the variables.
Step 2: Declare the variable i,j as integer,totwtime and totttime is equal to zero.
Step 3: Get the value of ' n' assign pid as I and get the value of p[i].btime.
Step 4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time and turnaround time.
Step 5: Calculate total wait time and total turnaround time by dividing by total number of process.
Step 6: Print total wait time and total turnaround time.
Step 7: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct
{
int pid;
int btime;
int wtime;
}
sp;
int main()
{
int i,j,n,tbm=0,towtwtime=0,totttime
sp*p,t;
printf("\n sjf schaduling ..\n");
printf("enter the no of processor");
scanf("%d",&n);
p=(sp*)malloc(sizeof(sp));
printf("\n enter the burst time");
for(i=0;i<n;i++)
{
printf("\n process %d\t",i+1);
scanf("%d",&p[i].btime);
p[i].pid=i+1;
p[i].wtime=0;
```



```
}
for(i=0;i<n;i++)
for(j=j+1,j<n;j++)
{
if(p[i].btime>p[j].btime)
{
t=p[i];
p[i]=p[j];
p[j]=t;
} }
printf("\n process scheduling\n");
printf("\n process \tburst time \t w
for(i=0;i<n;i++)
{
towtwtime+=p[i].wtime=tbm;
tbm+=p[i].btime;
printf("\n%d\t\t%d",p[i].pid,p[i].bt
printf("\t\t%d\t\t%d",p[i].wtime,p[i
}
totttime=tbm+towtwtime;
printf("\n total waiting time :%d", totwtime );
printf("\n average waiting time :%f",(float)totwtime/n);
printf("\n total turn around time :%d",totttime);
printf("\n average turn around time: :%f",(float)totttime/n);
}
```

OUTPUT:

RESULT:

Ex.No:6	PRODUCER CONSUMER PROBLEM USING SEMAPHORES

AIM:

To write a C-program to implement the producer – consumer problem using semaphores.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the required variables.
Step 3: Initialize the buffer size and get maximum item you want to produce.
Step 4: Get the option, which you want to do either producer, consumer or exit from the operation.
Step 5: If you select the producer, check the buffer size if it is full the producer should not produce the item or otherwise produce the item and increase the value buffer size.
Step 6: If you select the consumer, check the buffer size if it is empty the consumer should not consume the item or otherwise consume the item and decrease the value of buffer size.
Step 7: If you select exit come out of the program.
Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>
int mutex=1,full=0,empty=3,x=0;
main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");
while(1) {
printf("\nENTER YOUR CHOICE\n");
scanf("%d",&n);
switch(n)
{ case 1:
if((mutex==1)&&(empty!=0))
producer();
else
printf("BUFFER IS FULL");
break;
```



```
case 2:
if((mutex==1)&&(full!=0))
consumer();
else
printf("BUFFER IS EMPTY");
break;
case 3:
exit(0);
break;
}
}
}
int wait(int s) {
return(--s); }
int signal(int s) {
return(++s); }
void producer() {
mutex=wait(mutex);
full=signal(full);
empty=wait(empty);
x++;
printf("\nproducer produces the item%d",x);
mutex=signal(mutex); }
void consumer() {
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\n consumer consumes item%d",x);
x--;
mutex=signal(mutex); }
```


Ex.No:10	THREADING & SYNCHRONIZATION APPLICATIONS

AIM:

To write a c program to implement Threading and Synchronization Applications.

ALGORITHM:

Step 1: Start the process
Step 2: Declare process thread, thread-id.
Step 3: Read the process thread and thread state.
Step 4: Check the process thread equals to thread-id by using if condition.
Step 5: Check the error state of the thread.
Step 6: Display the completed thread process.
Step 7: Stop the process

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
void* doSomeThing(void *arg)
{
   unsigned long i = 0;
   pthread_t id = pthread_self();

   if(pthread_equal(id,tid[0]))
   {
      printf("\n First thread processing\n");
   }
   else
   {
      printf("\n Second thread processing\n");
   }

   for(i=0; i<(0xFFFFFFFF);i++);
   return NULL;
}
int main(void)
{
   int i = 0;
```



```
   int err;

   while(i < 2)
   {
      err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);
      if (err != 0)
         printf("\ncan't create thread :[%s]", strerror(err));
      else
         printf("\n Thread created successfully\n");

      i++;
   }

   sleep(5);
   return 0;
}
```

SAMPLE OUTPUT:

RESULT:

Ex.No:13.a	PAGE REPLACEMENT ALGORITHMS
	FIFO

AIM:

To write a C program for implementation of FIFO page replacement algorithm.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the necessary variables.
Step 3: Enter the number of frames.
Step 4: Enter the reference string end with zero.
Step 5: FIFO page replacement selects the page that has been in memory the longest time and when the page must be replaced the oldest page is chosen.
Step 6: When a page is brought into memory, it is inserted at the tail of the queue.
Step 7: Initially all the three frames are empty.
Step 8: The page fault range increases as the no of allocated frames also increases.
Step 9: Print the total number of page faults.
Step 10: Stop the program.

PROGRAM:

```
#include<stdio.h>
int main()
{
int i=0,j=0,k=0,i1=0,m,n,rs[30],flag=1,p[30];
system("clear");
printf("FIFO page replacement algorithm....\\n");
printf("enter the no. of frames:");
scanf("%d",&n);
printf("enter the reference string:");
while(1)
{
scanf("%d",&rs[i]);
if(rs[i]==0)
break;
i++;
}
m=i;
for(j=0;j<n;j++)
p[j]=0;
for(i=0;i<m;i++)
```



```
{
flag=1;
for(j=0;j<n;j++)
if(p[j]==rs[i])
{
printf("data already in page....\n");
flag=0;
break;
}
if(flag==1)
{
p[i1]=rs[i];
i1++;
k++;
if(i1==n)
i1=0;
for(j=0;j<n;j++)
{
printf("\n page %d:%d",j+1,p[j]);
if(p[j]==rs[i])
printf("*");
}
printf("\n\n");
}
}
printf("total no page faults=%d",k);
}
```

OUTPUT: