Swami Vivekanand College of Engineering Indore
Department of Information Technology

Session: July-Dec 2023

LAB MANUAL

Semester: vth

Operating System

IT-501

Submitted To

Ms.Aarti khare

(Assistant Professor)

Submitted By

INDEX

s.no	Name of Experiment	Date of Experiment	Date of submission	signature	Remark
1.	Program to implement FCFS CPU scheduling algorithm				
2.	Program to implement SJF CPU scheduling algorithm				
3.	Program to implement priority CPU scheduling				
4.	Program to implement Round Robin CPU scheduling algorithm.				
5.	Program to implement classical inter process communication problem(Producer consumer).				
6.	Program to implement classical inter process communication problem(Reader Writers).				
7.	Program to implement classical inter process communication problem(Dining Philosophers).				
8.	Program to implement FIFO page replacement algorithm.				
9.	Program to implement LRU page replacement algorithm.				

Experiment-1

Q.1 Program to implement FSFC CPU scheduling algorithm.

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:

```
/* A program to simulate the FCFS CPU scheduling algorithm */

#include<stdio.h>
int main()
{
char pn[10][10];

int arr[10],bur[10],star[10],finish[10],tat[10],wt[10],i,n;
int totwt=0,tottat=0;

printf("Enter the number of processes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the Process Name, Arrival Time & Burst Time:");
scanf("%s%d%d",&pn[i],&arr[i],&bur[i]);
}
for(i=0;i<n;i++)
{
if(i==0)
{
star[i]=arr[i];
wt[i]=star[i]-arr[i];
finish[i]=star[i]+bur[i];
tat[i]=finish[i]-arr[i];
}
else
{
star[i]=finish[i-1];
wt[i]=star[i]-arr[i];
finish[i]=star[i]+bur[i];
tat[i]=finish[i]-arr[i];
}
}
printf("\nPName Arrtime Burtime Start TAT Finish");
```

```
for(i=0;i<n;i++)
{
printf("\n%s\t%6d\t\t%6d\t%6d\t%6d\t%6d",pn[i],arr[i],bur[i],star[i],tat[i],finish[i]);
totwt+=wt[i];
tottat+=tat[i];
}
printf("\nAverage Waiting time:%f", (float)totwt);
printf("\nAverage Turn Around Time:%f", (float)tottat);
}
```

OUTPUT:

```
$ vi fcfs.c
$ cc fcfs.c
$ ./a.out
Enter the number of processes: 3
Enter the Process Name, Arrival Time & Burst Time: 1 2 3
Enter the Process Name, Arrival Time & Burst Time: 2 5 6
Enter the Process Name, Arrival Time & Burst Time: 3 6 7
PName Arrtime Burtime Start TAT Finish
 1        2        3      2     3    5
 2        5        6      5     6    11
 3        6        7      11    12   18
Average Waiting time: 1.666667
Average Turn Around Time: 7.000000
```

Experiment-2

Q.2 Program to implement SJF CPU scheduling algorithm

ALGORITHM:

Step 1: Start the program.
Step 2: Get the number of process.
Step 3: Get the id and service time for each process.
Step 4: Initially the waiting time of first short process as 0 and total time of first short is process the
service time of that process.
Step 5: Calculate the total time and waiting time of remaining process.
Step 6: Waiting time of one process is the total time of the previous process.
Step 7: Total time of process is calculated by adding the waiting time and service time of each process.
Step 8: Total waiting time calculated by adding the waiting time of each process.
Step 9: Total turn around time calculated by adding all total time of each process.
Step 10: Calculate average waiting time by dividing the total waiting time by total number of process.
Step 11: Calculate average turn around time by dividing the total waiting time by total number of
process.
Step 12: Display the result.
Step 13: Stop the program.

Program:

```
/* A program to simulate the SJF CPU scheduling algorithm */
#include<stdio.h>
#include<string.h>
main()
{
int i=0,pno[10],bt[10],n,wt[10],temp=0,j,tt[10];
float sum,at;
printf("\n Enter the no of process ");
scanf("\n %d",&n);
printf("\n Enter the burst time of each process");
for(i=0;i<n;i++)
{
printf("\n p%d",i);
scanf("%d",&bt[i]);
}

for(i=0;i<n-1;i++)
{
for(j=i+1;j<n;j++)
{
if(bt[i]>bt[j])
{
temp=bt[i];
bt[i]=bt[j];
bt[j]=temp;
temp=pno[i];
pno[i]=pno[j];
pno[j]=temp;
}
```

```
}
}
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=bt[i-1]+wt[i-1];
sum=sum+wt[i];
}
printf("\n process no \t burst time\t waiting time \t turn around time\n");
for(i=0;i<n;i++)
{
tt[i]=bt[i]+wt[i];
at+=tt[i];
printf("\n p%d\t\t%d\t\t%d\t\t%d",i,bt[i],wt[i],tt[i]);
}
printf("\n\n\t Average waiting time%f\n\t Average turn around time%f", sum, at);
}
```

OUTPUT:

```
$ vi sjf.c
$ cc sjf.c
$ ./a.out
Enter the no of process 5
Enter the burst time of each process
p0 1
p1 5
p2 2
p3 3
p4 4
process no burst time waiting time turn around time
p0 1 0 1
p1 2 1 3
p2 3 3 6
p3 4 6 10
p4 5 10 15
Average waiting time 4.000000
Average turn around time 7.000000
```

Experiment-3

Q.3 Write a program to implemen priority scheduling

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>
#include<conio.h>
 #include<iostream.h>
void main()
{
 clrscr();
  int x,n,p[10],pp[10],pt[10],w[10],t[10],awt,atat,i;
printf("Enter the number of process : ");
scanf("%d",&n);
printf("\n Enter process : time priorities \n");
for(i=0;i<n;i++){
printf("\nProcess no %d : ",i+1);
scanf("%d %d",&pt[i],&pp[i]);
p[i]=i+1;
}
for(i=0;i<n-1;i++)
{
for(int j=i+1;j<n;j++)
{
if(pp[i]<pp[j])
{
x=pp[i]; pp[i]=pp[j];
pp[j]=x;
x=pt[i];
pt[i]=pt[j];
pt[j]=x;
x=p[i];
p[i]=p[j];
p[j]=x;
}
}
}

w[0]=0;
awt=0;
t[0]=pt[0];
atat=t[0];
for(i=1;i<n;i++)
```

```
{
w[i]=t[i-1];
awt+=w[i];
t[i]=w[i]+pt[i];
atat+=t[i];
}
printf("\n\n Job \t Burst Time \t Wait Time \t Turn Around TimePriority \n");
for(i=0;i<n;i++){
printf("\n %d \t\t %d  \t\t %d \t\t %d \t\t %d
\n",p[i],pt[i],w[i],t[i],pp[i]);
awt/=n;
atat/=n;
printf("\n Average Wait Time : %d \n",awt);
printf("\n Average Turn Around Time : %d \n",atat);
getch();
}
```

OUTPUT:-

```
Enter process:p1
Enter cpu burst:10
Enter priroty:3

Enter process:p2
Enter cpu burst:1
 Enter priroty:1
 Enter process:p3
Enter cpu burst:2
Enter priroty:4
Enter process:p4
Enter cpu burst:1
Enter priroty:5

Enter process:p5
Enter cpu burst:5
Enter priroty:2
Process gaint chart is below::
0-p2-1-p5 -6- p1 -16- p3 -18- p4-19
TP:0.26 TAT:12  RT:12  WT:6.2
```

Experiment-4

Q.4 To write a C program to implement Round Robin CPU scheduling algorithm.

ALGORITHM:

Step 1: Start the program.
Step 2: Initialize all the structure elements.
Step 3: Receive inputs from the user to fill process id, burst time and arrival time.
Step 4: Calculate the waiting time for all the process id.
i. The waiting time for first instance of a process is calculated as:
a[i].waittime=count +a[i].arrivt.
ii. The waiting time for the rest of the instances of the process is calculated as:
a) If the time quantum is greater than the remaining burst time then waiting time is calculated as: a[i].waittime=count + tq.
b) Else if the time quantum is greater than the remaining burst time then waiting time is
calculated as: a[i].waittime=count - remaining burst time
Step 5: Calculate the average waiting time and average turnaround time
Step 6: Print the results of the step 4.
Step 7: Stop the program

Program :

```
#include<stdio.h>
struct process
{
int burst,wait,comp,f;
}p[20]={0,0};
int main()
{
int n,i,j,totalwait=0,totalturn=0,quantum,flag=1,time=0;
printf("\nEnter The No Of Process :");
scanf("%d",&n);
printf("\nEnter The Quantum time (in ms) :");
scanf("%d",&quantum);
for(i=0;i<n;i++)
{
printf("Enter The Burst Time (in ms) For Process #%2d :",i+1);
scanf("%d",&p[i].burst);
p[i].f=1;
}
printf("\nOrder Of Execution \n");
printf("\nProcess Starting Ending Remaining");
printf("\n\t\tTime \tTime \t Time");
while(flag==1)
{
flag=0;
for(i=0;i<n;i++)
{
if(p[i].f==1)
{
flag=1;
j=quantum;
if((p[i].burst-p[i].comp)>quantum)
```

```
{
p[i].comp+=quantum;
}
else
{
p[i].wait=time-p[i].comp;
j=p[i].burst-p[i].comp;
p[i].comp=p[i].burst;
p[i].f=0;
}
printf("\nprocess # %-3d %-10d %-10d %-10d", i+1, time, time+j,
p[i].burst-p[i].comp);
time+=j;
}
}
}
printf("\n\n----------");
printf("\nProcess \t Waiting Time TurnAround Time ");
for(i=0;i<n;i++)
{
printf("\nProcess # %-12d%-15d%-15d",i+1,p[i].wait,p[i].wait+p[i].burst);
totalwait=totalwait+p[i].wait;
totalturn=totalturn+p[i].wait+p[i].burst;
}
printf("\n\nAverage\n---------- ");
printf("\nWaiting Time: %fms",totalwait/(float)n);
printf("\nTurnAround Time : %fms\n\n",totalturn/(float)n);
return 0;
}
```

OUTPUT:

```
/tmp/N1tvTs4VL4.o
Enter The No Of Process :3
Enter The Quantum time (in ms) :5
Enter The Burst Time (in ms) For Process # 1 :25
Enter The Burst Time (in ms) For Process # 2 :30
Enter The Burst Time (in ms) For Process # 3 :54
Order Of Execution

Process Starting Ending Remaining
                Time    Time    Time
process # 1   0       5       20
process # 2   5       10      25
process # 3   10      15      49
process # 1   15      20      15
process # 2   20      25      20
process # 3   25      30      44
process # 1   30      35      10
process # 2   35      40      15
process # 3   40      45      39
process # 1   45      50      5
process # 2   50      55      10
process # 3   55      60      34
```

```
process # 1   60        65        0
process # 2   65        70        5
process # 3   70        75        29
process # 2   75        80        0
process # 3   80        85        24
process # 3   85        90        19
process # 3   90        95        14
process # 3   95        100       9
process # 3   100       105       4
process # 3   105       109       0

------------------
Process            Waiting Time TurnAround Time
Process # 1          40           65
Process # 2          50           80
Process # 3          55           109
Average
Waiting Time: 48.333332ms
TurnAround Time : 84.666664ms
```

Experiment -5

Q.5 To write C programs to simulate Producer – Consumer Problem.

ALGORITHM:
Step 1: Start the program.
Step 2: Declare and initialize the necessary variables.
Step 3: Create a Producer.
Step 4: Producer (Child Process) performs a down operation and writes a message.
Step 5: Producer performs an up operation for the consumer to consume.
Step 6: Consumer (Parent Process) performs a down operation and reads or consumes the data
(message).
Step 7: Consumer then performs an up operation.
Step 8: Stop the program

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

OUTPUT:

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

Experiment -6

Q.6 To write C programs to simulate Readers – Writers Problem.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare and initialize the necessary variables.
Step 3: Create a Reader.
Step 4: Reader performs a down operation and read a message in the database.
Step 5: Reader performs an up operation for the Writer to access a message from the database.
Step 6: Write performs a down operation and writes or access the data (message) from the database.
Step 7: Writer then performs an up operation.
Step 8: Stop the program.

Program:

```
//File: mesg.h
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>
#include<stdlib.h>
#define MKEY1 5543L
#define MKEY2 4354L
#define PERMS 0666
typedef struct
{
long mtype;
char mdata[50];
mesg;
//File: sender1.c
#include "mesg.h"
mesg msg;
int main()
{
int mq_id;
int n;
if((mq_id=msgget(MKEY1,PERMS|IPC_CREAT))<0)
{
printf("Sender: Error creating message");
exit(1);}
msg.mtype=1111L;
n=read(0,msg.mdata,50);
msg.mdata[n]='\0';
msgsnd(mq_id,&msg,50,0);
}
//File: receiver1.c
#include "mesg.h"
mesg msg;
main()
{
int mq_id;
```

```
int n;
if( ( mq_id=msgget(MKEY1, PERMS|IPC_CREAT ) ) < 0)
{
printf("receiver: Error opening message");
exit(1);
}
msgrcv(mq_id,&msg,50,1111L,0);
write(1,msg.mdata,50);
msgctl(mq_id,IPC_RMID,NULL);
}
```

OUTPUT:

```
$ vi mesg.h
$ vi sender1.c
$ vi receiver1.c
$ cc sender1.c
$ ./a.out
mcet
$ cc receiver1.c
$ ./a.out
mcet
```

Experiment-7

Q.7 To write C programs to simulate solutions to Dining Philosophers Problem**.**

ALGORITHM:

Step 1: Start the program.
Step 2: Define the number of philosophers.
Step 3: Declare one thread per philosopher.
Step 4: Declare one chopsticks per philosopher.
Step 5: When a philosopher is hungry.
i. See if chopsticks on both sides are free.
ii. Acquire both chopsticks or.
iii. Eat.
iv. restore the chopsticks.
v. If chopsticks aren't free.
Step 6: Wait till they are available.
Step 7: Stop the program.

Program:

```
int tph, philname[20], status[20], howhung, hu[20], cho;
int 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)
{
printf("\nAll are hungry..\nDead lock stage will occur");
printf("\nExiting..");
}
else
{
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..");
}
}
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++)
{
printf("\nP %d is granted to eat", philname[hu[pos]]);
for(x=pos;x<howhung;x++)
printf("\nP %d is waiting", philname[hu[x]]);
}
}
two()
{
nt 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]]);
}
}
}
}
```

OUTPUT:

```
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
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
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
P 3 and P 5 are granted to eat
P 0 is waiting
P 5 is waiting
combination
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
```

Experiment-8

Q.8 Programs to implement FIFO Page Replacement Algorithms:

ALGORITHM:
Step 1: Start the program
Step 2: Read the number of frames
Step 3: Read the number of pages
Step 4: Read the page numbers
Step 5: Initialize the values in frames to -1
Step 6: Allocate the pages in to frames in First in first out order.
Step 7: Display the number of page faults.
Step 8: Stop the program

Program:

```
/* A program to simulate FIFO Page Replacement Algorithm */
#include<stdio.h>
main()
{
int a[5],b[20],n,p=0,q=0,m=0,h,k,i,q1=1;
char f='F'
printf("Enter the Number of Pages:");
scanf("%d",&n);
printf("Enter %d Page Numbers:",n);
for(i=0;i<n;i++)
scanf("%d",&b[i]);
for(i=0;i<n;i++)
{
if(p==0)
{
if(q>=3)
q=0;
a[q]=b[i];
q++;
if(q1<3)
{
q1=q;
}
}
printf("\n%d",b[i]);
printf("\t");
for(h=0;h<q1;h++)
printf("%d",a[h]);
if((p==0)&&(q<=3))
{
printf("-->%c",f);
m++;
}
p=0;
for(k=0;k<q1;k++)
{
if(b[i+1]==a[k])
p=1;
}
}
```

```
printf("\nNo of faults:%d",m);
```

OUTPUT:

```
$ vi fifo.c
$ cc fifo.c
$ ./a.out
Enter the Number of Pages: 12
Enter 12 Page Numbers:
2  3  2  1  5  2  4  5  3  2  5  2
2  2-->F
3  23-->F
2  23
1  231-->F
5  531-->F
2  521-->F
4  524-->F
5  524
3  324-->F
2  324
5  354-->F
2  352-->F
No of faults: 9
```

Experiment-9

Q.9 Program to implement LRU page replacement algorithms.

ALGORITHM:

Step 1: Start the program.
Step 2: Read the number of frames.
Step 3: Read the number of pages.
Step 4: Read the page numbers.
Step 5: Initialize the values in frames to -1.
Step 6: Allocate the pages in to frames by selecting the page that has not been used for the longest
period of time.
Step 7: Display the number of page faults.
Step 8: Stop the program.

Program :

```
* A program to simulate LRU Page Replacement Algorithm */
#include<stdio.h>
main(){
int a[5],b[20],p=0,q=0,m=0,h,k,i,q1=1,j,u,n;
char f='F';
printf("Enter the number of pages:");
scanf("%d",&n);
printf("Enter %d Page Numbers:",n);
for(i=0;i<n;i++)
scanf("%d",&b[i]);
for(i=0;i<n;i++)
{
if(p==0)
{
if(q>=3)
q=0;
a[q]=b[i];
q++;
if(q1<3){
q1=q;}
}
printf("\n%d",b[i]);
printf("\t");
for(h=0;h<q1;h++)
printf("%d",a[h]);
if((p==0)&&(q<=3)){
printf("-->%c",f);
m++;
}
p=0;
if(q1==3)
{
for(k=0;k<q1;k++)
{
if(b[i+1]==a[k])
p=1;
}
```

```
for(j=0;j<q1;j++)
{
u=0;
k=i;
while(k>=(i-1)&&(k>=0))
{
if(b[k]==a[j])
u++;
k--;}
if(u==0)
q=j;
}
}
else{
for(k=0;k<q;k++){
if(b[i+1]==a[k])
p=1;}
}
}
printf("\nNo of faults:%d",m);
```

OUTPUT:

```
$ vi lru.c
$ cc lru.c
$ ./a.out
Enter the number of pages: 12
Enter 12 Page Numbers:
2 3 2 1 5 2 4 5 3 2 5 2
2 2-->F
3 23-->F
2 23
1 231-->F
5 251-->F
2 251
4 254-->F
5 254
3 354-->F
2 352-->F
5 352
2 352
No of faults: 7
```