**OPERATING SYSTEMS**

**LAB MANUAL**



SUBJECT CODE: IT221

REGULATION: R19

CLASS: II Year II Semester

**DEPARTMENT OF INFORMATION TECHNOLOGY**

***ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY & SCIENCES***

(UGC AUTONOMOUS)

(Affiliated to Andhra University, Approved by AICTE & Accredited by NBA) Sangivalasa, Bheemunipatnam Mandal, Visakhapatnam Dt. 531162.

Phone : 08933-225083/84/87 Fax:226395

Website:[www.anits.edu.in](http://www.anits.edu.in/) [email:principal@anits.edu.in](mailto:principal@anits.edu.in)

***ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY & SCIENCES***

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**Faculty Signature Student Signature**

**Operating Systems Lab Record**

Submitted By:

M.Kovid

IT-B

319126511103

Submitted to:

1. Durga Praveen Kumar

Assistant Professor

Department Of Information Technology

INDEX

|  |  |  |
| --- | --- | --- |
| **S.NO** | **NAME OF THE EXPERIMENT** | **CO** |
| 1. | Implement example programs on Shell Programming & AWKscripts. | 2 |
| 2. | Write programs using the following system calls of LINUX operating system: Fork, exec, getpid, exit, wait, close, stat, opendir,readdir | 1 |
| 3. | Write programs using the I/O system calls of LINUX operating system (open, read, write, etc) and error reporting using errno | 1 |
| 4. | Write C programs to simulate UNIX commands like ls, grep,etc. | 1 |
| 5. | CPU SCHEDULING ALGORITHMS  a)Round Robin b)SJF c)FCFS d)Priority | 1 |
| 6. | Implement the Producer – Consumer problem using semaphores (using LINUX system calls). | 1 |
| 7. | Programs using pipes | 2 |
| 8. | Implement Banker’s algorithm for handlingdeadlock | 1 |
| 9. | Implement free space management strategies such as First fit, Best fit and Worstfit | 2 |
| 10. | Implement page replacement algorithms such as FIFO,LRU | 2 |
| 11. | FILE ALLOCATION STRATEGIES  a)sequential b)Indexed c)Linked | 3 |
| 12. | Implement disk arm scheduling algorithms such as FCFS,SSTF | 3 |

**Prerequisite:**

Operating System Concepts.

**Course Objectives:**

1. Analyze the working of an operating system, its programming interface and file system.

2. Develop algorithms for process scheduling, memory management, page replacement algorithms and disk scheduling.

**Course Outcomes:**

|  |  |
| --- | --- |
| After completion of this course, a student will be able to : | |
| 1. | Implement scheduling algorithms, deadlock management. |
| 2. | Implement free space management and page replacement strategies. |
| 3. | Implement file allocation methods and disk scheduling algorithms. |

**Mapping of course outcomes with program outcomes:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **PO** | | | | | | | | | | | | **PSO** | |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **1** | **2** |
| **CO** | **1** | 3 | 3 | 3 |  | 3 | 2 | 3 |  |  | 3 |  |  | 3 | 3 |
| **2** | 3 | 3 | 3 |  | 3 |  | 2 | 2 |  | 3 |  | 3 | 3 | 3 |
| **3** | 3 | 3 | 3 |  | 3 |  | 2 | 2 |  | 3 |  | 3 | 3 | 3 |

### **PROGRAM EDUCATIONAL OBJECTIVES (PEOs)**

|  |  |
| --- | --- |
| **1** | To enable IT graduates to excel in professional career contributing towards the need of the industry and society. |
| **2** | To impart knowledge of theory, practice, and design in the areas of Information Technology like Data Science and Computer Communications and training the students to analyze and interpret the data for IT applications. |
| **3** | Exhibit leadership, managerial and ethical qualities in their profession and adapt to global environment by engaging in lifelong learning. |

### **PROGRAM SPECIFIC OUTCOMES (PSOs)**

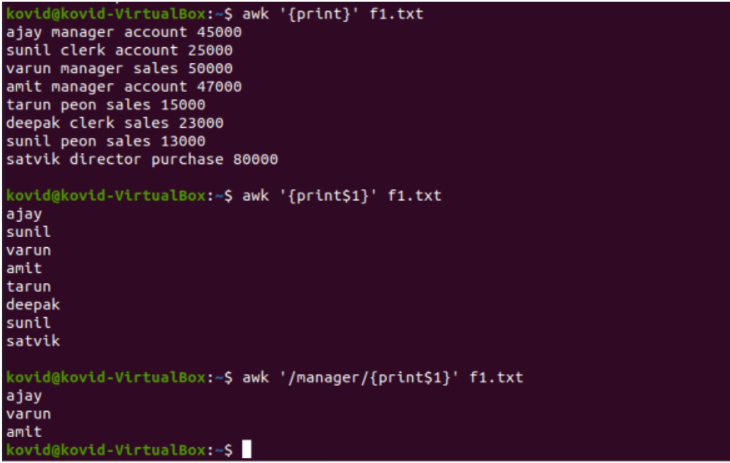
|  |  |
| --- | --- |
| **1** | The ability to analyze, design and develop computer based information systems leveraging the concepts of computing techniques, data analytics, software engineering and networking. |
| **2** | The ability to apply the knowledge of computing skills in building the Software Systems that meet the requirements of Industry and Society. |

### **PROGRAM OUTCOMES (POs)**

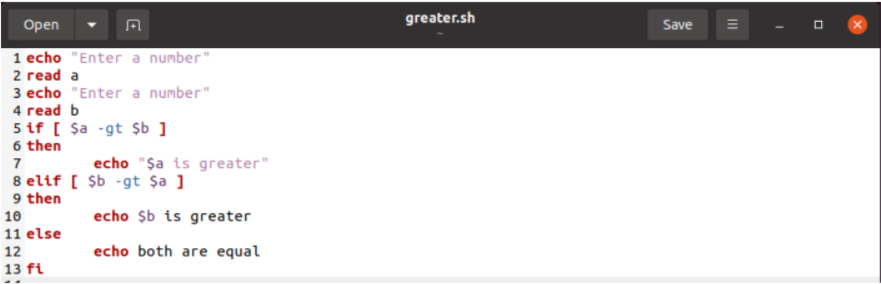
|  |  |
| --- | --- |
| **1** | Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. |
| **2** | Problem analysis: Identify, formulate, review research literature, and analyze complex 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. |

1. **Implement example programs on Shell Programming & AWK scripts.**

a)AWK Script



b) Find greater no. program in shell



Output:



c) Prime number



d) Factorial



**e)Fibonacci**

N=6

a=0

b=1

echo "The Fibonacci series is : "

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

do

echo -n " $a "

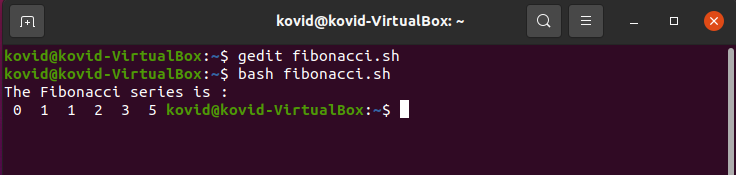
fn=$((a + b))

a=$b

b=$fn

done

**Output:**

****

**2. Write programs using the following system calls of LINUX operating system: Fork, exec, getpid, exit, wait, close, stat, opendir,readdir.**

a)fork

#include<stdio.h>

#include<unistd.h>

void main()

{

int n,i,sum=1;

pid\_t p;

printf("\nThis is M.Kovid - 319126511103 Output : ");

printf("Enter n value:");

scanf("%d",&n);

p=fork();

if(p==0)

{

for(i=n;i!=1;i--)

sum\*=i;

printf("%d!=%d\n",n,sum);

}

else

{

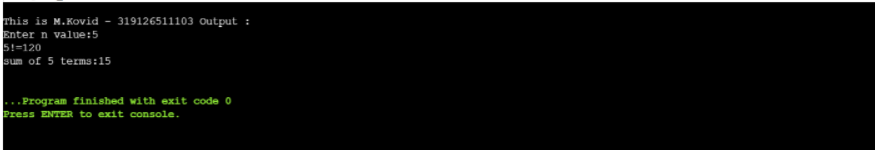
sleep(5);

printf("sum of %d terms:%d\n",n,(n\*(n+1))/2);

}

}

Output:



b) GETPID

#include<stdio.h>

#include<unistd.h>

main()

{

pid\_t pid;

pid=fork();

if(fork()==0)

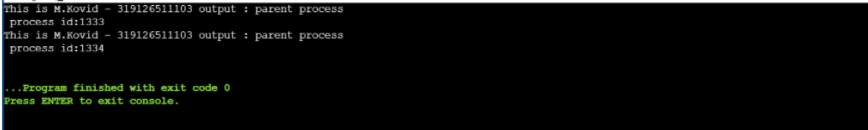
printf("child process\n process id:%d\n",getpid());

else

printf("parent process\n process id:%d\n",getppid());

}

Output:



# c) Open Directory

#include<stdio.h>

#include<dirent.h>

void main()

{

char dirname[10];

DIR \*p;

struct dirent \*d;

printf("\nThis is M.Kovid - 319126511103 Output : ");

printf("\nEnter directory name\n");

scanf("%s",dirname);

p=opendir(dirname);

if(p==NULL)

{

perror("Cannot find directory");

exit(0);

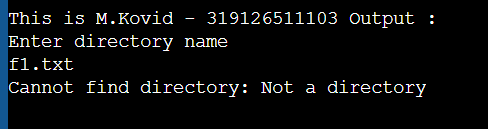
}

while(d=readdir(p))

printf("%s\n",d->d\_name);

}

Output:



**3.Write programs using the I/O system calls of LINUX operating system (open, read, write, etc) and error reporting using errno**

# Copy content from one file to another file

#include<stdio.h>

#include<unistd.h>

int main()

{

FILE \*f1,\*f2;

int fd[2];

char n1[20],n2[20],buf[20];

printf("\nThis is M.Kovid - 319126511103 Output : ");

printf("enter first file name");

scanf("%s",n1);

printf("enter second file name");

scanf("%s",n2);

f2=fopen(n2,"w");f1=fopen(n1,"r");

while(fread(buf,1,1,f1)!=0)

{

fwrite(buf,1,1,f2);

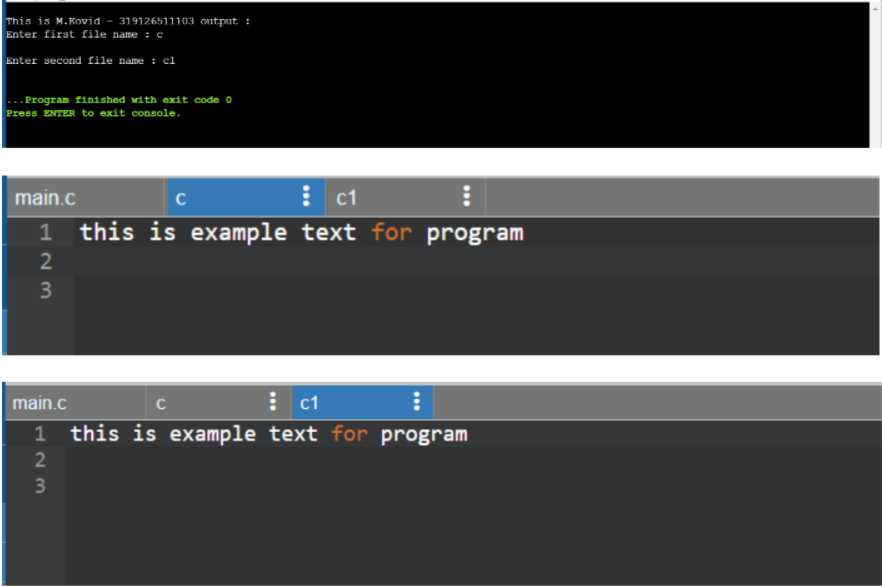
}

fclose(f1);

fclose(f2);

}

Output:



**4. Write a C programs to simulate UNIX commands like ls, grep,etc.**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include<string.h>

#include <fcntl.h>

void match\_pattern(char \*argv[])

{

int fd,r,j=0;

char temp,line[100];

if((fd=open(argv[2],O\_RDONLY)) != -1)

{

while((r=read(fd,&temp,sizeof(char)))!= 0)

{

if(temp!='\n')

{

line[j]=temp;

j++;

}

else

{

if(strstr(line,argv[1])!=NULL)

printf("%s\n",line);

memset(line,0,sizeof(line));

j=0;

}

}

}

}

void main(int argc,char \*argv[])

{

struct stat stt;

if(argc==3)

{

if(stat(argv[2],&stt)==0)

match\_pattern(argv);

else

{

perror("stat()");

exit(1);

}

}

}

**5.Given the list of processes, their CPU burst times and arrival times, display/print the Gantt chart for scheduling algorithms FCFS,SJF, PRIORITY & RR. For each of the scheduling policies, compute and print the average waiting time, average turnaround time and Gantt chart**

**//FCFS Algorithm**

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival Time** | **Burst Time** |
| **P1** | **1** | **3** |
| **P2** | **0** | **3** |
| **P3** | **2** | **4** |

#include<stdio.h>

struct process

{

int pid;

float at,bt,wt,ct;

};

void main()

{

int n,i,j;

float sum=0;

printf(“\nThis is M.Kovid - 319126511103 Output : ”);

printf("enter no. of processes:");

scanf("%d",&n);

struct process p[1000],t;

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

{

printf("enter %d process pid:\nenter pid,at,bt:\n",i+1);

scanf("%d%f%f",&p[i].pid,&p[i].at,&p[i].bt);

}

for(i=1;i<n;i++)

{

t=p[i];//t=p[1]=p2

for(j=i-1;j<i&&j>=0;j--)

{

if(t.at<p[j].at)//p2.at<p1.at

{

p[j+1]=p[j];//p[1]=p[0]=p1

p[j]=t;//p[0]=p2

}

}

}

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

{

if(i==0)

{

p[i].wt=0;

p[i].ct=p[i].wt+p[i].bt;

}

else

{

p[i].wt=p[i-1].wt+p[i-1].bt;

p[i].ct=p[i].wt+p[i].bt;

sum+=p[i].wt;

}

}

printf("S.No\tProcess id\tArrival time\tBurst time\tWaiting time\tCompletion time\n");

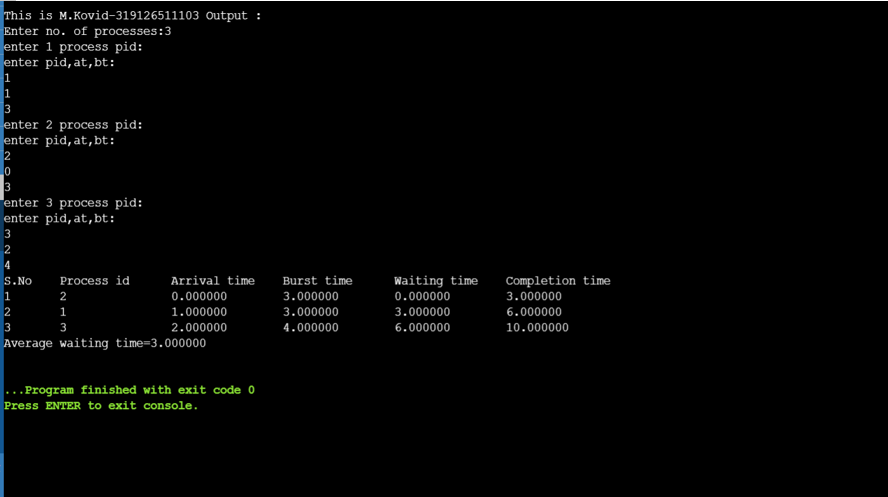
for(i=0;i<n;i++)

printf("%d\t%d\t\t%f\t%f\t%f\t%f\n",i+1,p[i].pid,p[i].at,p[i].bt,p[i].wt,p[i].ct);

printf("Average waiting time=%f\n",sum/n);

}

Output:



**//SJF Algorithm**

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival Time** | **Burst Time** |
| **P1** | **1** | **3** |
| **P2** | **0** | **2** |
| **P3** | **2** | **4** |

#include<stdio.h>

struct process

{

int pid;

float at,bt,wt,ct;

};

void main()

{

int n,i,j;

float sum=0;

printf(“\nThis is M.Kovid - 319126511103 Output : ”);

printf("enter no. of processes:");

scanf("%d",&n);

struct process p[1000],t;

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

{

printf("enter %d process pid:\nenter pid,at,bt:\n",i+1);

scanf("%d%f%f",&p[i].pid,&p[i].at,&p[i].bt);

}

for(i=1;i<n;i++)

{

t=p[i];//t=p[1]=p2

for(j=i-1;j<i&&j>=0;j--)

{

if(t.bt<p[j].bt)//p2.at<p1.at

{

p[j+1]=p[j];//p[1]=p[0]=p1

p[j]=t;//p[0]=p2

}

}

}

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

{

if(i==0)

{

p[i].wt=0;

p[i].ct=p[i].wt+p[i].bt;

}

else

{

p[i].wt=p[i-1].wt+p[i-1].bt;

p[i].ct=p[i].wt+p[i].bt;

sum+=p[i].wt;

}

}

printf("S.No\tProcess id\tArrival time\tBurst time\tWaiting time\tCompletion time\n");

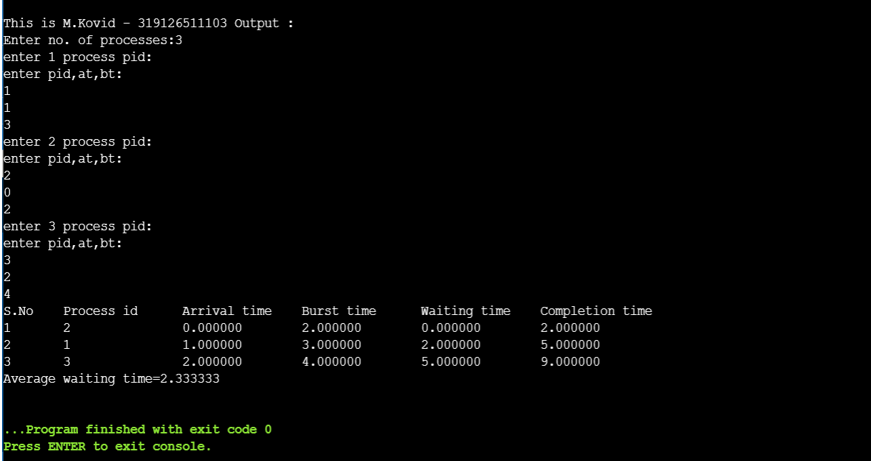
for(i=0;i<n;i++)

printf("%d\t%d\t\t%f\t%f\t%f\t%f\n",i+1,p[i].pid,p[i].at,p[i].bt,p[i].wt,p[i].ct);

printf("Average waiting time=%f\n",sum/n);

}

Output:



**//Priority Algorithm**

|  |  |  |  |
| --- | --- | --- | --- |
| **Process** | **Arrival Time** | **Burst Time** | **Priority** |
| **1** | **0** | **3** | **2** |
| **2** | **1** | **4** | **1** |
| **3** | **2** | **5** | **3** |

#include<stdio.h>

struct process

{

int pid;

float at,bt,wt,ct,pt;

};

void main()

{

int n,i,j;

float sum=0;

printf("\nThis is M.Kovid - 319126511103 Output : ");

printf("\nEnter no. of processes:");

scanf("%d",&n);

struct process p[1000],t;

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

{

printf("enter %d process pid:\nenter pid,at,bt,pt:\n",i+1);

scanf("%d%f%f%f",&p[i].pid,&p[i].at,&p[i].bt,&p[i].pt);

}

for(i=1;i<n;i++)

{

t=p[i];//t=p[1]=p2

for(j=i-1;j<i&&j>=0;j--)

{

if(t.pt<p[j].pt)//p2.at<p1.at

{

p[j+1]=p[j];//p[1]=p[0]=p1

p[j]=t;//p[0]=p2

}

}

}

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

{

if(i==0)

{

p[i].wt=0;

p[i].ct=p[i].wt+p[i].bt;

}

else

{

p[i].wt=p[i-1].wt+p[i-1].bt;

p[i].ct=p[i].wt+p[i].bt;

sum+=p[i].wt;

}

}

printf("S.No\tProcess id\tArrival time\tBurst time\tWaiting time\tCompletion time\tPriority\n");

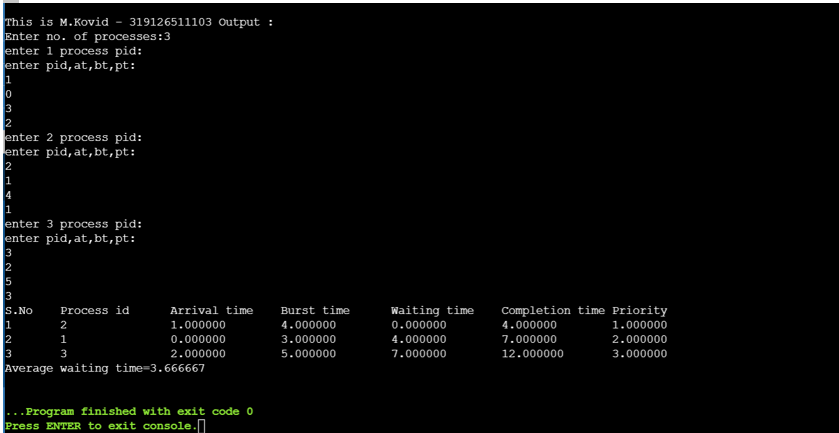
for(i=0;i<n;i++)

printf("%d\t%d\t\t%f\t%f\t%f\t%f\t%f\n",i+1,p[i].pid,p[i].at,p[i].bt,p[i].wt,p[i].ct,p[i].pt);

printf("Average waiting time=%f\n",sum/n);

}

Output:



**Round Robin**

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival Time** | **Burst Time** |
| **P1** | **0** | **24** |
| **P2** | **1** | **3** |
| **P3** | **2** | **3** |

**Program :**

#include<stdio.h>

struct process

{

int pid;

float at,bt,rt;

};

int n,i,j,ts;

struct process p[1000],t;

void sort()

{

for(i=1;i<n;i++)

{

t=p[i];

for(j=i-1;j<i&& j>=0;j--)

{

if(t.at<p[j].at)

{

p[j+1]=p[j];

p[j]=t;

}

}

}

}

void main()

{

float tt=0;

int k=0;

printf("\nThis is M.Kovid - 319126511103 Output : ");

printf("\nEnter no. of processes : ");

scanf("%d",&n);

printf("\nEnter time slice : ");

scanf("%d",&ts);

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

{

printf("\nEnter %d process pid,at,bt : ",k+1);

scanf("%d%f%f",&p[k].pid,&p[k].at,&p[k].bt);

p[k].rt=p[k].bt;

printf("%f\t",p[k].rt);

}

printf("\nGantt chart : \n");

printf("\nTime slice : %d",ts);

printf("\nS.No\tProcess id\tArrival time\tBurst time\tRemaining Time\n");

do

{

sort();

tt=0;

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

{

if(p[i].rt>0)

{

if(p[i].rt<ts)

{

p[i].rt=0;

tt=tt+p[i].rt;

}

else

{

p[i].rt=p[i].rt-ts;

tt=tt+p[i].rt;

printf("%f\n",tt);

}

printf("%d\t%d\t\t%f\t%f\t%f\n",i+1,p[i].pid,p[i].at,p[i].bt,p[i].rt);

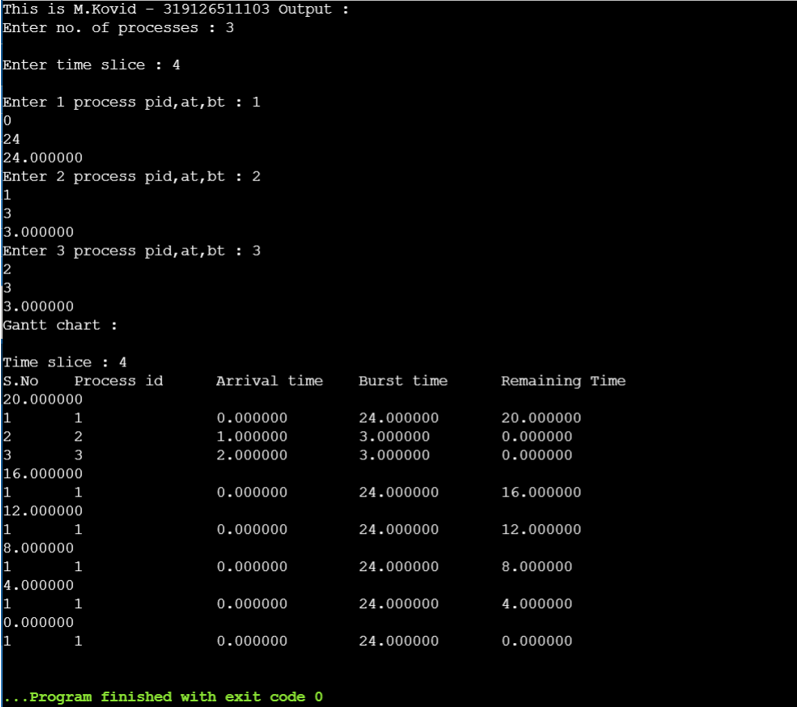
}

}

}while(tt>0);

}

**Output :**



# STRF

#include<stdio.h>

struct process

{

int pid;

float at,bt,rt;

};

int n,i,j,ts;

struct process p[1000],t;

void sort()

{

for(i=1;i<n;i++)

{

t=p[i];

for(j=i-1;j<i&& j>=0;j--)

{

if(t.rt<p[j].rt)

{

p[j+1]=p[j];

p[j]=t;

}

}

}

}

Void main()

{

float tt=0;

int k=0;

printf("\nThis is M.Kovid - 319126511103 Output : ");

printf("enter no. of processes:");

scanf("%d",&n);

printf("enter time slice");

scanf("%d",&ts);

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

{

printf("enter %d process pid ,at,bt",k+1);

scanf("%d%f%f",&p[k].pid,&p[k].at,&p[k].bt);

p[k].rt=p[k].bt;

printf("%f\t",p[k].rt);

}

printf("gantt chart:\n");

printf("Time slice:%d",ts);

printf("\nS.No\tProcess id\tArrival time\tBurst time\tRemaining Time\n");

do

{

sort();

tt=0;

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

{

if(p[i].rt>0)

{

if(p[i].rt<ts)

{

p[i].rt=0;

tt=tt+p[i].rt;

}

else

{

p[i].rt=p[i].rt-ts;

tt=tt+p[i].rt;

}

printf("%d\t%d\t\t%f\t%f\t%f\n",i+1,p[i].pid,p[i].at,p[i].bt,p[i].rt);

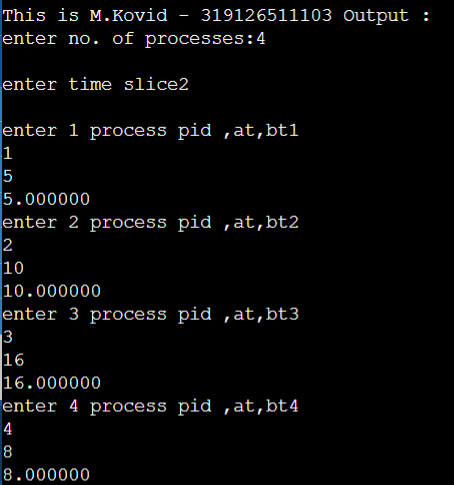
}

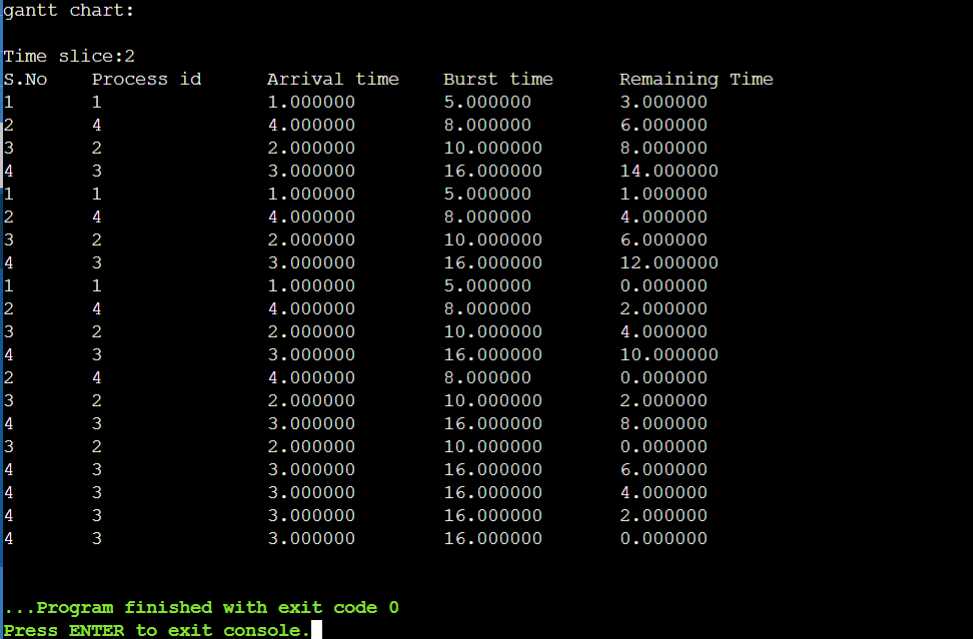
}

}while(tt>0);

}

**Output:**

****

****

**6.Implement the Producer – Consumer problem using semaphores (using LINUX system calls).**

#include<stdio.h>

#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()

{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

printf("This is M.Kovid - 319126511103 Output : ");

printf("\n1.Producer\n2.Consumer\n3.Exit");

while(1)

{

printf("\nEnter your choice:");

scanf("%d",&n);

switch(n)

{

case 1: if((mutex==1)&&(empty!=0))

producer();

else

printf("\nBuffer is full");

break;

case 2: if((mutex==1)&&(full!=0))

consumer();

else

printf("\nBuffer is empty");

break;

case 3:

exit(0);

break;

}

}

return 0;

}

int wait(int s)

{

return (--s);

}

int signal(int s)

{

return(++s);

}

void producer()

{

mutex=wait(mutex);//0

full=signal(full);//1

empty=wait(empty);//2

x++;

printf("\nProducer produces the item %d",x);

mutex=signal(mutex);//1

}

void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

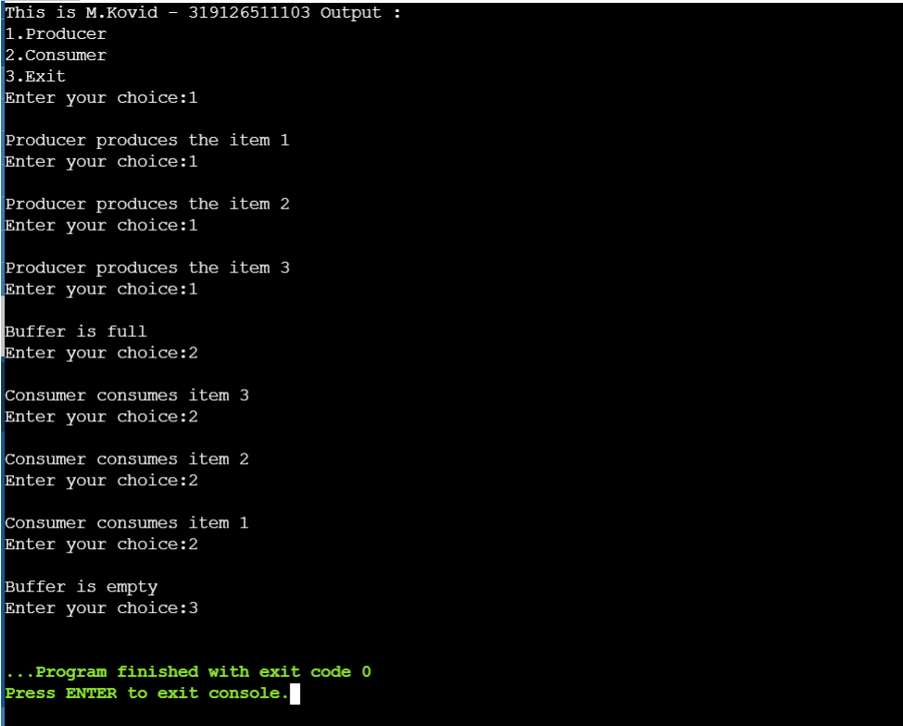
printf("\nConsumer consumes item %d",x);

x--;

mutex=signal(mutex);

}

**Output :**

****

#include<stdio.h>

void main()

{

int buffer[10], bufsize,in,out,produce,consume,choice=0;

in = 0;

out = 0;

bufsize = 10;

printf("\nThis is M.Kovid - 319126511103 Output : ");

while(choice !=3)

{

printf("\n 1. Produce \t 2. Consume \t3. Exit");

printf("\n Enter your choice: ");

scanf("%d", &choice);

switch(choice) {

case 1: if((in+1)%bufsize==out)

printf("\n Buffer is Full");

else

{

printf("\nEnter the item no: ");

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);

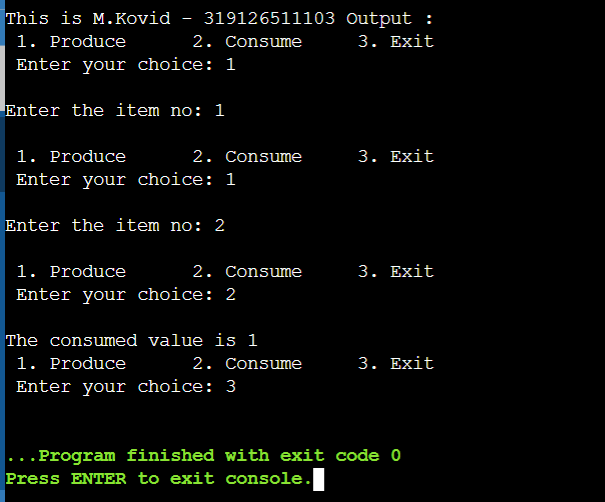
out = (out+1)%bufsize;

}

break;

} } }

**Output:**

****

7. **Programs using pipes**

Write and read using fork and pipe

#include<stdio.h>

#include<unistd.h>

int main()

{

int fd[2];

char buf[20];

pipe(fd);

printf("\nThis is M.Kovid - 319126511103 Output : ");

if(fork()==0)

{

write(fd[1],"hello",5);

}

else

{

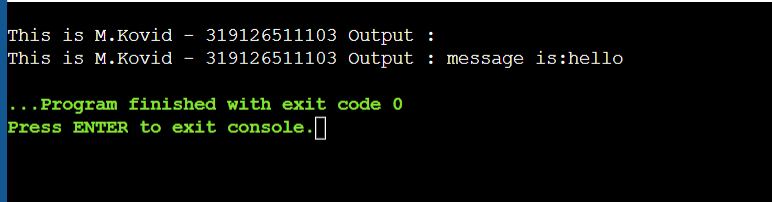
read(fd[0],&buf,5);

printf("\nmessage is:%s",buf);

}

}

Output:



**Two way pipes:**

#include<stdio.h>

#include<unistd.h>

int main() {

int pipefds1[2], pipefds2[2];

int returnstatus1, returnstatus2;

int pid;

char pipe1writemessage[20] = "Hi";

char pipe2writemessage[20] = "Hello";

char readmessage[20];

returnstatus1 = pipe(pipefds1);

printf("\nThis is M.Kovid - 319126511103 Output : ");

if (returnstatus1 == -1) {

printf("Unable to create pipe 1 \n");

return 1;

}

returnstatus2 = pipe(pipefds2);

if (returnstatus2 == -1) {

printf("Unable to create pipe 2 \n");

return 1;

}

pid = fork();

if (pid != 0) {

close(pipefds1[0]); // Close the unwanted pipe1 read side

close(pipefds2[1]); // Close the unwanted pipe2 write side

printf("In Parent: Writing to pipe 1 – Message is %s\n", pipe1writemessage);

write(pipefds1[1], pipe1writemessage, sizeof(pipe1writemessage));

read(pipefds2[0], readmessage, sizeof(readmessage));

printf("In Parent: Reading from pipe 2 – Message is %s\n", readmessage);

} else { //child process

close(pipefds1[1]); // Close the unwanted pipe1 write side

close(pipefds2[0]); // Close the unwanted pipe2 read side

read(pipefds1[0], readmessage, sizeof(readmessage));

printf("In Child: Reading from pipe 1 – Message is %s\n", readmessage);

printf("In Child: Writing to pipe 2 – Message is %s\n", pipe2writemessage);

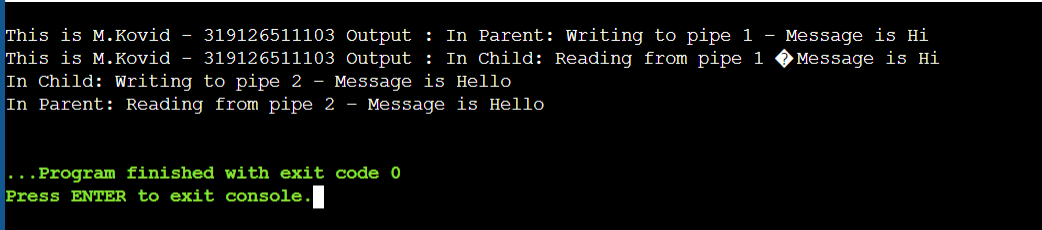
write(pipefds2[1], pipe2writemessage, sizeof(pipe2writemessage));

}

return 0;

}

Output:



**8. Implement Banker’s algorithm for handling deadlock**

#include<stdio.h>

void main()

{

int n,r,all[20][20],max[20][20],need[20][20],avl[20],i,j,k=0,l=0,p[20],t=0;;

printf("\nThis is M.Kovid - 319126511103 Output : ");

printf("enter no. of process:");

scanf("%d",&n);

printf("enter no. of resources:");

scanf("%d",&r);

printf("enter allocation instances:\n");

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

{

printf("enter p%d allocation resources instances:\n",i);

for(j=0;j<r;j++)

{

printf("enter %d resource instances:",j);

scanf("%d",&all[i][j]);

}

}

printf("enter max resources instances:\n");

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

{

printf("enter p%d max resources instances:\n",i);

for(j=0;j<r;j++)

{

printf("enter %d resource instances:",j);

scanf("%d",&max[i][j]);

}

}

printf("need matrix:\n");

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

{

for(j=0;j<r;j++)

{

need[i][j]=max[i][j]-all[i][j];

printf("%d\t",need[i][j]);

}

printf("\n");

}l=0,p[20],t=0;

printf("enter available resources:\n");

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

scanf("%d",&avl[i]);

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

p[i]=0;

printf("safe sequence\n");

do

{

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

{

k=0;

for(j=0;j<r;j++)

{if(need[i][j]<=avl[j])

k++;}

if(k==j)

{

for(l=0,j=0;j<r;j++,l++)

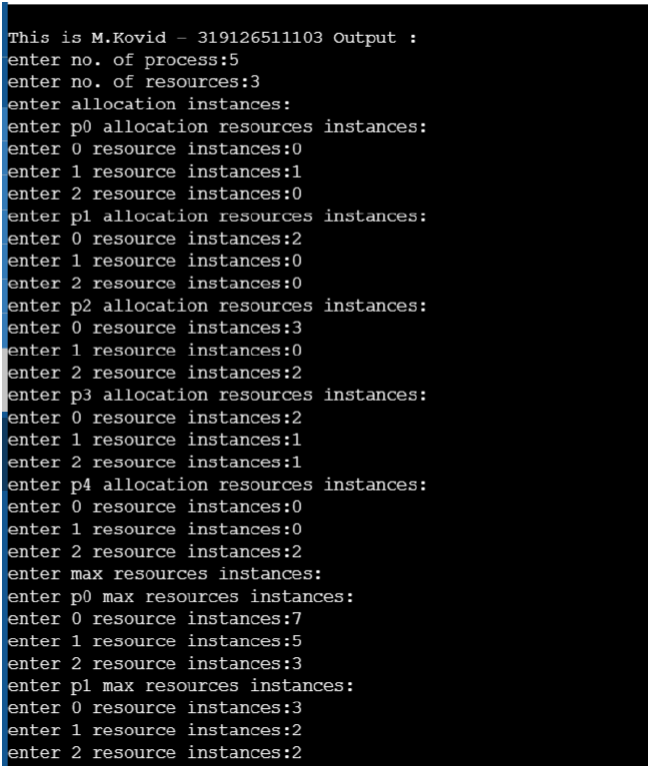
{avl[l]=avl[l]+all[i][j];

if(p[i]==0)

{printf("p%d is safe\n",i);

p[i]=1;

Output :





**9. Implement free space management strategies such as First fit, Best fit and Worstfit**

# **FIRST FIT**

#include<stdio.h>

void main()

{

int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

for(i = 0; i < 10; i++)

{

flags[i] = 0;

allocation[i] = -1;

}

printf("\nThis is M.Kovid - 319126511103 Output :\n");

printf("Enter no. of blocks: ");

scanf("%d", &bno);

printf("\nEnter size of each block: ");

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

scanf("%d", &bsize[i]);

printf("\nEnter no. of processes: ");

scanf("%d", &pno);

printf("\nEnter size of each process: ");

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

scanf("%d", &psize[i]);

for(i = 0; i < pno; i++) //allocation as per first fit

for(j = 0; j < bno; j++)

if(flags[j] == 0 && bsize[j] >= psize[i])

{

allocation[j] = i;

flags[j] = 1;

break;

}

//display allocation details

printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");

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

{

printf("\n%d\t\t%d\t\t", i+1, bsize[i]);

if(flags[i] == 1)

printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);

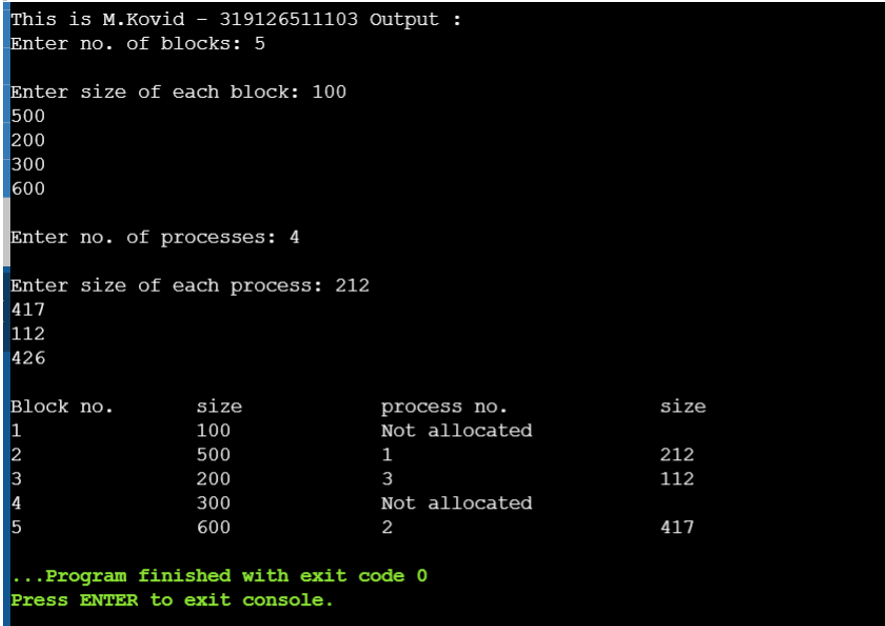
else

printf("Not allocated");

}

}

**Output :**

****

**Best Fit:**

#include<stdio.h>

void main()

{

int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;

static int barray[20],parray[20];

printf("\n\t\t\tMemory Management Scheme - Best Fit");

printf("\nThis is M.Kovid – 319126511103 Output : \n");

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

printf("Enter the number of processes:");

scanf("%d",&np);

printf("\nEnter the size of the blocks:-\n");

for(i=1;i<=nb;i++)

{

printf("Block no.%d:",i);

scanf("%d",&b[i]);

}

printf("\nEnter the size of the processes :-\n");

for(i=1;i<=np;i++)

{

printf("Process no.%d:",i);

scanf("%d",&p[i]);

for(j=1;j<=nb;j++)

{

if(barray[j]!=1)

{

temp=b[j]-p[i];

if(temp>=0)

if(lowest>temp)

{

parray[i]=j;

lowest=temp;

}

}

}

fragment[i]=lowest;

barray[parray[i]]=1;

lowest=10000;

}

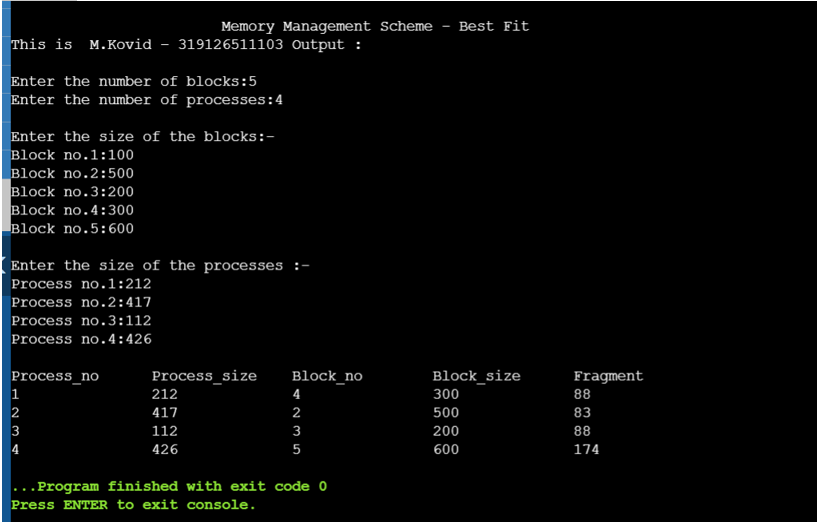
printf("\nProcess\_no\tProcess\_size\tBlock\_no\tBlock\_size\tFragment");

for(i=1;i<=np && parray[i]!=0;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);

}

**Output :**

****

**Worst Fit:**

#include<stdio.h>

int main()

{

int n,m,a[100],b[100],c[100],d[100],i,j,t,t1;

printf("\nThis is M.Kovid – 319126511103 Output : \n");

printf("enter no of process u want to ...\n");

scanf("%d",&n);

printf("enter number of backsize elements...\n");

scanf("%d",&m);

printf("enter process elements\n");

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

{

scanf("%d",&a[i]);

c[i]=a[i];

}

printf("enter block size elements....\n");

for(j=0;j<m;j++)

{

scanf("%d",&b[j]);

d[j]=j;

}

//sort the elements in the block size according to increasing order

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

{

for(j=0;j<m-i-1;j++)

{

if(b[j]<b[j+1])

{

t=b[j];

b[j]=b[j+1];

b[j+1]=t;

t1=d[j];

d[j]=d[j+1];

d[j+1]=t1;

}

}

}

//after sorting apply best fit algorithm

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

{

for(j=0;j<m;j++)

{ if(b[j]>=a[i])

{

b[j]=b[j]-a[i];

a[i]=d[j];//passing previous index value of the process to array a

break;

}

}

if(j==m)

{

a[i]=-1;//process not allocations

}

}

//printing table

printf("\nProcess\_size\tBlock\_no");

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

{

printf("\n%d\t\t%d\t",c[i],a[i]);

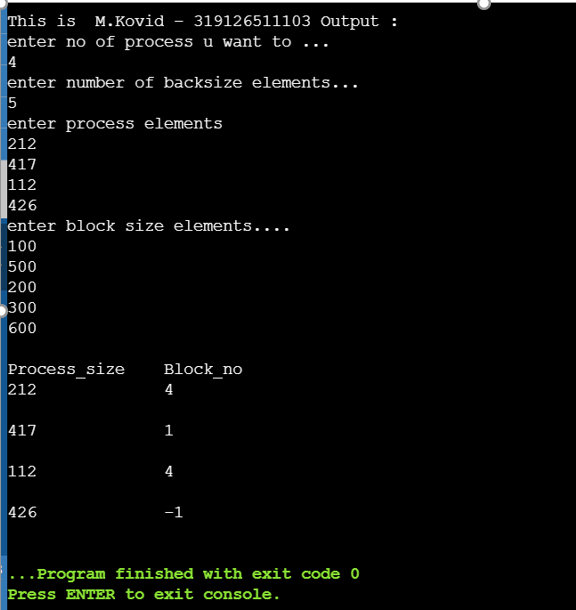
printf("\n");

}

return 0;

}

**Output :**



**10.Implement page replacement algorithms such as FIFO,LRU,Optimal**

FIFO

#include<stdio.h>

int search(int);

void enqueue();

int i,front=-1,rear=-1,e,max,q[100];

void main()

{

int s=0,j;

float count=0,n;

printf("\nThis is M.Kovid - 319126511103 Output : \n");

printf("enter no. of frames:");

scanf("%d",&max);

printf("enter no. of pages:");

scanf("%f",&n);

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

{

q[i]=-1;

}

j=0;

while(j<n)

{

printf("Enter page into frame:");

scanf("%d",&e);

j++;

s=search(e);

if(s==0)

{

++count;

enqueue();

}

}

printf("No. of pages:%.2f\n",n);

printf("Miss Count:%.2f\n",count);

printf("Page miss ratio:%.2f\n",(count/n));

}

int search(int a)

{

int k=0;

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

{

if(q[i]==a)

{

k=1;

break;

}

}

if(k==1)

return 1;

else

return 0;

}

void enqueue()

{

if(rear==-1)

{

front=0;

rear=0;

q[rear]=e;

}

else if(rear==max-1)

{

rear=0;

q[rear]=e;

}

else

{

rear++;

q[rear]=e;

}

printf("Pages in frame:\n");

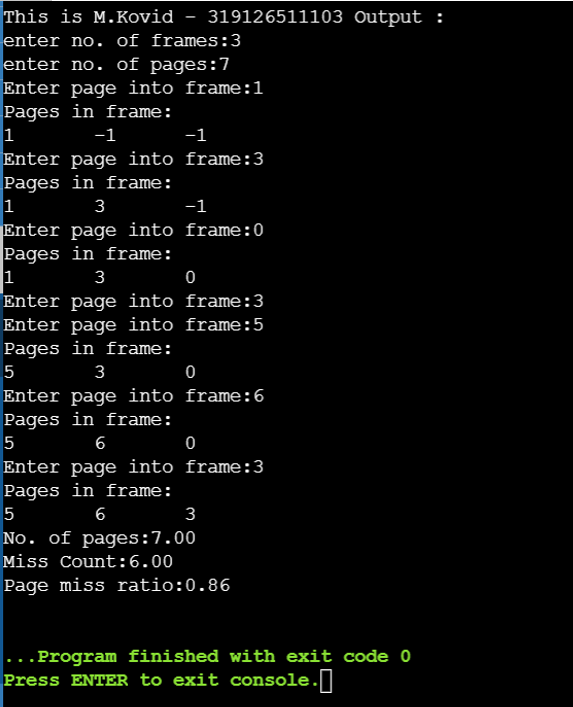
for(i=0;i<=max-1;i++)

printf("%d\t",q[i]);

printf("\n");

}

Output:



LRU:

#include<stdio.h>

int findLRU(int time[], int n)

{

int i, minimum = time[0], pos = 0;

for(i = 1; i < n; ++i)

{

if(time[i] < minimum)

{

minimum = time[i];

pos = i;

}

}

return pos;

}

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;

printf("\nThis is M.Kovid - 319126511103 Output :\n");

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter reference string: ");

for(i = 0; i < no\_of\_pages; ++i)

{

scanf("%d", &pages[i]);

}

for(i = 0; i < no\_of\_frames; ++i)

{

frames[i] = -1;

}

for(i = 0; i < no\_of\_pages; ++i)

{

flag1 = flag2 = 0;

for(j = 0; j < no\_of\_frames; ++j)

{

if(frames[j] == pages[i])

{

counter++;

time[j] = counter;

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0)

{

for(j = 0; j < no\_of\_frames; ++j)

{

if(frames[j] == -1)

{

counter++;

faults++;

frames[j] = pages[i];

time[j] = counter;

flag2 = 1;

break;

}

}

}

if(flag2 == 0)

{

pos = findLRU(time, no\_of\_frames);

counter++;

faults++;

frames[pos] = pages[i];

time[pos] = counter;

}

printf("\n");

for(j = 0; j < no\_of\_frames; ++j)

{

printf("%d\t", frames[j]);

}

}

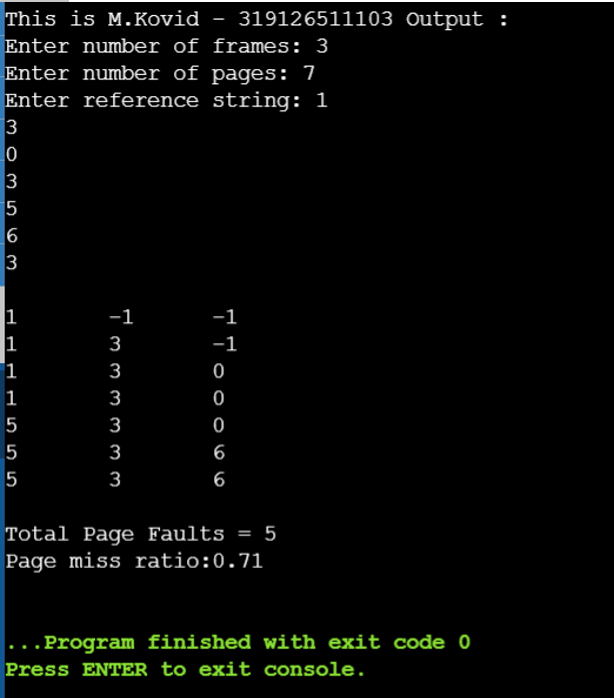
printf("\n\nTotal Page Faults = %d", faults);

printf("\nPage miss ratio:%.2f\n",(float)faults/no\_of\_pages);

return 0;

}

Output:



Optimal:

#include<stdio.h>

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;

printf("\nThis is M.Kovid - 319126511103 Output: \n");

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter page reference string: ");

for(i = 0; i < no\_of\_pages; ++i)

{

scanf("%d", &pages[i]);

}

for(i = 0; i < no\_of\_frames; ++i)

{

frames[i] = -1;

}

for(i = 0; i < no\_of\_pages; ++i)

{

flag1 = flag2 = 0;

for(j = 0; j < no\_of\_frames; ++j)

{

if(frames[j] == pages[i])

{

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0)

{

for(j = 0; j < no\_of\_frames; ++j)

{

if(frames[j] == -1)

{

faults++;

frames[j] = pages[i];

flag2 = 1;

break;

}

}

}

if(flag2 == 0)

{

flag3 =0;

for(j = 0; j < no\_of\_frames; ++j)

{

temp[j] = -1;

for(k = i + 1; k < no\_of\_pages; ++k)

{

if(frames[j] == pages[k])

{

temp[j] = k;

break;

}

}

}

for(j = 0; j < no\_of\_frames; ++j)

{

if(temp[j] == -1)

{

pos = j;

flag3 = 1;

break;

}

}

if(flag3 ==0)

{

max = temp[0];

pos = 0;

for(j = 1; j < no\_of\_frames; ++j)

{

if(temp[j] > max)

{

max = temp[j];

pos = j;

}

}

}

frames[pos] = pages[i];

faults++;

}

printf("\n");

for(j = 0; j < no\_of\_frames; ++j)

{

printf("%d\t", frames[j]);

}

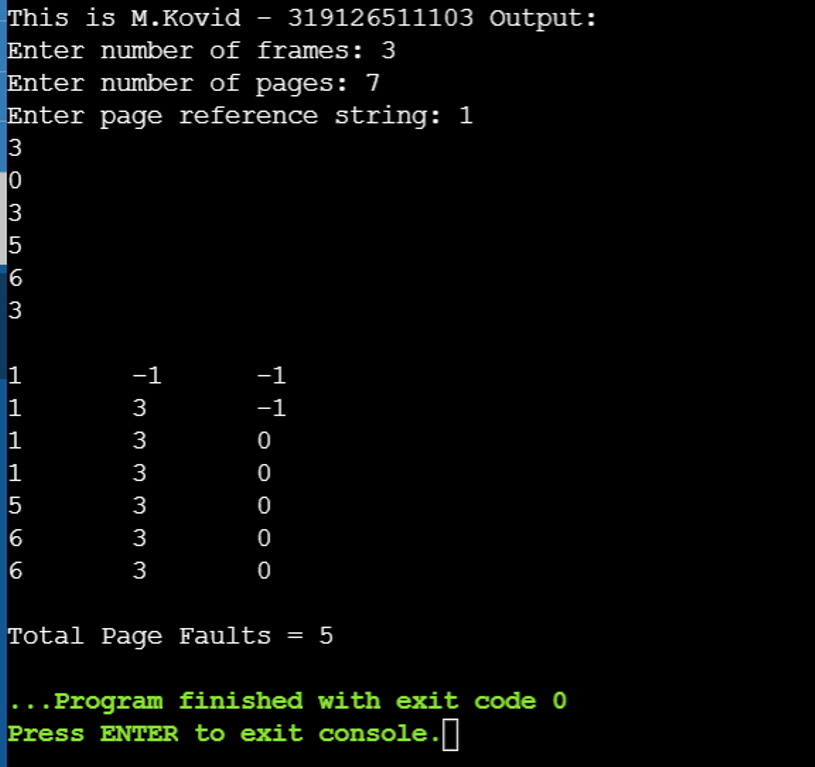
}

printf("\n\nTotal Page Faults = %d", faults);

return 0;

}

Output:



# **11. Implement file allocation techniques (Linked, Indexed and Contiguous)**

Sequential:

#include<stdio.h>

void main()

{

int f[50],i,st,len,j,c,k,count = 0;

for(i=0;i<50;i++)

f[i]=0;

printf("\nThis is M.Kovid - 319126511103 Output :\n");

printf("Files Allocated are : \n");

x: count=0;

printf("Enter starting block and length of files:");

scanf("%d%d", &st,&len);

for(k=st;k<(st+len);k++)

if(f[k]==0)

count++;

if(len==count)

{

for(j=st;j<(st+len);j++)

if(f[j]==0)

{

f[j]=1;

printf("%d\t%d\n",j,f[j]);

}

if(j!=(st+len-1))

printf("The file is allocated to disk\n");

}

else

printf("The file is not allocated \n");

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

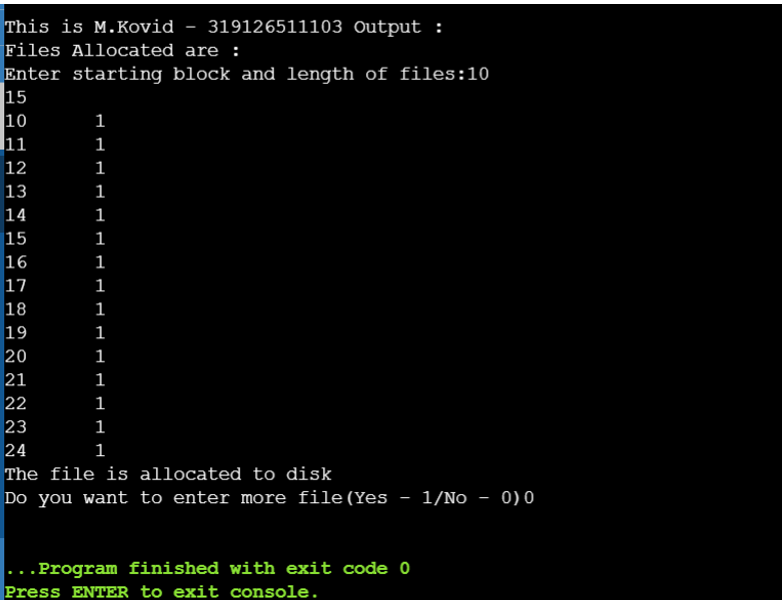
if(c==1)

goto x;

else

exit(0);}

Output:



Linked:

#include<stdio.h>

#include<stdlib.h>

void main()

{

int f[50], p,i, st, len, j, c, k, a;

for(i=0;i<50;i++)

f[i]=0;

printf("\nThis is M.Kovid - 319126511103 Output :\n");

printf("Enter how many blocks already allocated: ");

scanf("%d",&p);

printf("Enter blocks already allocated: ");

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

{

scanf("%d",&a);

f[a]=1;

}

x: printf("Enter index starting block and length: ");

scanf("%d%d", &st,&len);

k=len;

if(f[st]==0)

{

for(j=st;j<(st+k);j++)

{

if(f[j]==0)

{

f[j]=1;

printf("%d-------->%d\n",j,f[j]);

}

else

{

printf("%d Block is already allocated \n",j);

k++;

}

}}

else

printf("%d starting block is already allocated \n",st);

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

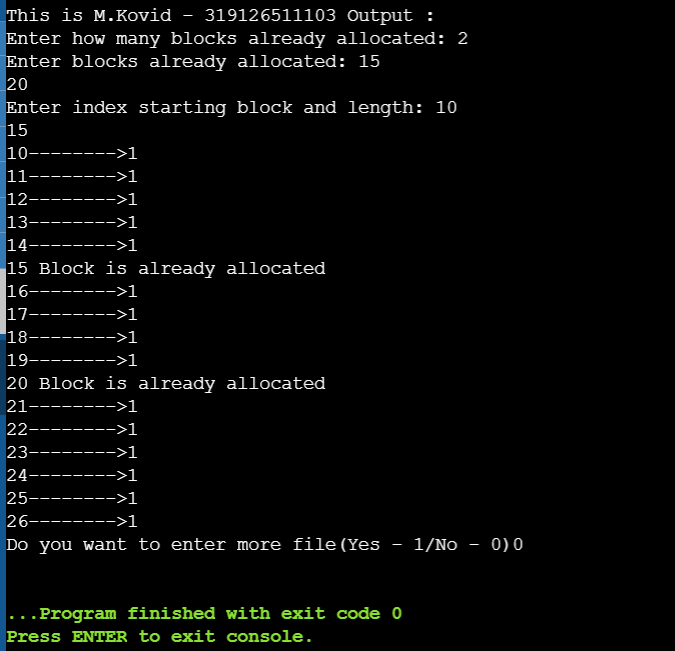
if(c==1)

goto x;

else

exit(0);}

Output:



Indexed:

#include<stdio.h>

#include<stdlib.h>

void main()

{

int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;

for(i=0;i<50;i++)

f[i]=0;

printf("\nThis is M.Kovid - 319126511103 Output: \n");

x:printf("Enter the index block: ");

scanf("%d",&ind);

if(f[ind]!=1)

{

printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);

scanf("%d",&n);

}

else

{

printf("%d index is already allocated \n",ind);

goto x;

}

y: count=0;

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

{

scanf("%d", &index[i]);

if(f[index[i]]==0)

count++;

}

if(count==n)

{

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

f[index[j]]=1;

printf("Allocated\n");

printf("File Indexed\n");

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

printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);

}

else

{

printf("File in the index is already allocated \n");

printf("Enter another file indexed");

goto y;

}

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)

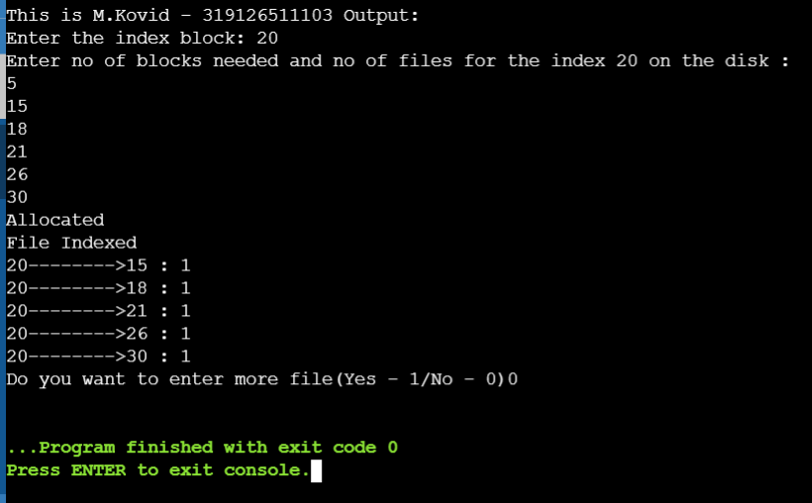
goto x;

else

exit(0);

}

Output:



**12. Implement disk arm scheduling algorithms such as FCFS,SSTF**

Disk FCFS

Program:

#include<stdio.h>

void main()

{

int queue[10],n,head,i,j,k,seek=0,max,diff;

float avg;

printf("\nThis is M.Kovid - 319126511103 Output : ");

printf("\nenter max disk range:\n");

scanf("%d",&max);

printf("enter queue size:\n");

scanf("%d",&n);

printf("enter the queue of disk positions to be read\n");

for(i=1;i<=n;i++)

scanf("%d",&queue[i]);

printf("enter the initial head position\n");

scanf("%d",&head);

queue[0]=head;

for(j=0;j<=n-1;j++)

{

diff=queue[j+1]-queue[j];

seek+=diff;

printf("disk head moves from %d to %d with seek %d\n",queue[j],queue[j+1],diff);

}

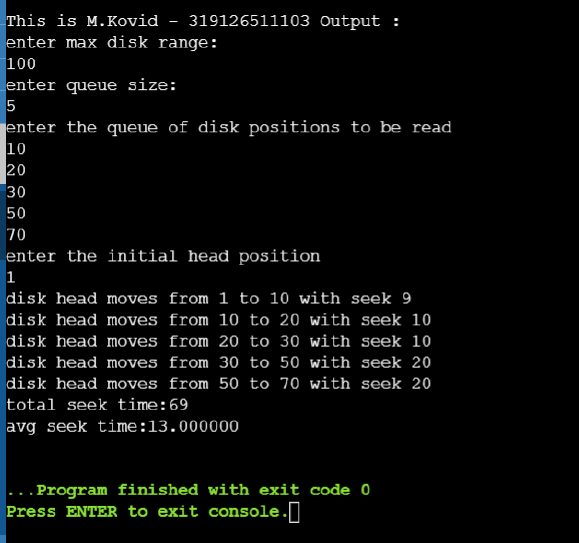
printf("total seek time:%d\n",seek);

avg=seek/n;

printf("avg seek time:%f\n",avg);

}

Output:



Disk SSTF:

Program:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int main()

{

int queue[100], queue2[100], q\_size, head, seek=0, temp,i,j;

float avg;

printf("\nThis is M.Kovid - 319126511103 Output : ");

printf("%s\n", "-----SSTF Disk Scheduling Algorithm-----");

printf("%s\n", "Enter the size of the queue");

scanf("%d", &q\_size);

printf("%s\n", "Enter queue elements");

for(i=0; i<q\_size; i++)

{

scanf("%d",&queue[i]);

}

printf("%s\n","Enter initial head position");

scanf("%d", &head);

for(i=0; i<q\_size; i++)

{

queue2[i] = abs(head-queue[i]);

}

for(i=0; i<q\_size; i++)

{

for(j=i+1; j<q\_size;j++)

{

if(queue2[i]>queue2[j]){

temp = queue2[i];

queue2[i]=queue[j];

queue2[j]=temp;

temp=queue[i];

queue[i]=queue[j];

queue[j]=temp;

}

}

}

for(i=1; i<q\_size; i++)

{

seek = seek+abs(head-queue[i]);

head = queue[i];

}

printf("\nTotal seek time is %d\t",seek);

avg = seek/(float)q\_size;

printf("\nAverage seek time is %f\t", avg);

return 0;

}

Output:

