

# **Guru Nanak Institutions Technical Campus (Autonomous)**

School of Engineering & Technology

**DEPARTMENT OF COMPUTER SCIENCE &ENGINEERING**

Operating Systems Lab  
Lab Manual [Subject Code: **22PC0CS11**]  
For the Academic year 2023-24

II B Tech Semester-II [CSE]



Guru Nanak Institutions Technical  
Campus(Autonomous)  
Ibrahimpattanam, R R District – 501 506 (T.S.)



## **Department of Computer Science & Engineering**

### **LAB MANUAL FOR THE ACADEMIC YEAR 2023-24**

**SUB : Operating Systems Lab**

**SUB CODE : 22PC0CS11**

**SEMESTER : II**

**STREAM : CSE**

**Document No : GNITC(SB)/CSE/OS/R22**

**Date of Issue : 20-08-2024**

**Prepared : Mrs.B.Divya/Mrs.P.Mounika/Mrs.Mounika Reddy**

**PROGRAMMER :**

**VENUE :**

**BLOCK : CSE(SB)**

**Verified by:**

**Authorized by  
Dr. M.V.Narayana  
HOD-CSE**

## **GURU NANAK INSTITUTIONS TECHNICAL CAMPUS (Autonomous)**

### **Department of Computer Science & Engineering**

#### **VISION OF THE INSTITUTION: GNITC**

To be an internationally renowned institution in Engineering, Management, Pharmacy and related fields to produce scientists, engineers, entrepreneurs, leaders, academicians and thinkers of tomorrow with exemplary professional conduct and adherence to ethical values to serve for changing needs of industry and society.

#### **MISSION OF THE INSTITUTION: GNITC**

**M1:** Imbibe soft skills, technical skills, creatively and passion in students.

**M2:** Develop the faculty to reach the International standards.

**M3:** Maintain outcome based student centric teaching learning with high academic standards and quality that promotes the analytical thinking and independent judgement.

**M4:** Promote research, innovation, product development by collaborating with reputed industries & reputed universities in India and abroad. Offer collaborative industry programs in emerging areas and instill the spirit of enterprising.

**M5:** To instill the ethical values in the faculty and students to serve the society.

#### **VISION OF THE DEPARTMENT: CSE**

To become a premier Computer Science & Engineering department by imparting high quality education, ethical values, provide creative environment for innovation and global career opportunities.

#### **MISSION OF THE DEPARTMENT: CSE**

**M1:** Nurture young individuals into knowledgeable, skillful and ethical professionals in their pursuit of Computer Science & Engineering.

**M2:** Foster the students through excellent teaching learning process and sustain high performance through innovation.

**M3:** Provide high quality soft skills and advanced industry specific technical trainings to meet global career opportunities.

#### **PROGRAMME EDUCATIONAL OBJECTIVES (PEOs):**

**PEO 1:** Graduates shall have the ability to apply knowledge across the disciplines and in emerging areas of Computer Science & Engineering for higher studies, research, employability, product development and handle the realistic problems.

**PEO 2:** Graduates shall have good communication skills, possess ethical values, sense of responsibility to serve the society, and protect the environment.

**PEO 3:** Graduates shall possess academic excellence with innovative insight, managerial skills, leadership qualities, knowledge of contemporary issues and understand the need for lifelong learning for a successful professional career.

### **PROGRAMME OUTCOMES (POs): [Department of Computer Science & Engineering]**

The following list of programme outcomes describes what graduates are expected to know and be able to do at the time of graduation. Graduates at graduation will have:

- 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 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 lifelong learning in the broadest context of technological change.

### **PROGRAMME SPECIFIC OUTCOMES (PSOs):**

- 1:** Ability to apply software technical skills to design, develop and debug optimized solutions for all real world problems.
- 2:** Ascertain knowledge in diverse areas of Computer Science and experience an environment conducive in novel skills for successful career, entrepreneurship and higher studies.

## INDEX

S.No	Contents	Page.no
1	Lab Objective	6
2	Lab outcomes	6
3	Introduction About Lab	7
4	Guidelines to students a)Standard Operating Procedure (SOP) b) General guidelines	8-10
5	List of experiments as per the Autonomous curriculum	11
6	List of Additional Experiments	12
7	Text Books / Reference Books	12
8	Content of the experiment. Objective of the Experiment Hardware & Software Requirements Pre- requisite Flow chart/ algorithm with inputs and outputs Program Output Conclusion	13-63

## **1. OPERATING SYSTEMS LAB:**

### **LAB OBJECTIVE**

---

Upon successful completion of this Lab the student will be able to:

- To provide an understanding of the design aspects of operating system concepts through simulation
- Introduce basic Unix commands, system call interface for process management, interprocess communication and I/O in Unix

## **2. OPERATING SYSTEMS LAB:**

### **LAB OUTCOME**

---

Upon successful completion of this Lab the student will be able to:

- Simulate and implement operating system concepts such as scheduling, deadlock management, file management and memory management.
- Able to implement C programs using Unix system calls

### 3. INTRODUCTION ABOUT LAB

---

There are 60 systems (Compaq Presario) installed in this Lab. Their configurations are as follows:

Processor	:	Intel core i5
RAM	:	4 GB
Hard Disk	:	500 GB
Mouse	:	Optical Mouse

#### **Software:**

- ❖ All systems are configured with Linux Ubuntu or Windows 7 as per their lab requirement. This is very useful for students because they are familiar with different Operating Systems so that they can execute their programs in any programming environments.
- ❖ **Software installed:** Turbo C/ C++
- ❖ Systems are provided for students in the 1:1 ratio.
- ❖ Systems are assigned numbers and same system is allotted for students when they do the lab.

## **29. A. STANDARD OPERATING PROCEDURE – SOP**

a) Explanation on today's experiment by the concerned faculty using OHP/PPT/white Board covering the following aspects: 60 mins.

- 1) Name of the experiment/Aim
- 2) Software/Hardware required
- 3) Description about the program
- 4) C Program code

b) Writing of C programs by the student 30mins.

c) Compiling and execution of the program 90mins.

## **30. Writing of the experiment in the Observation Book:**

The students will write the today's experiment in the Observation book as per the following format:

- a) Name of the experiment/Aim
- b) Software/Hardware required
- c) Source Program
- d) Results for the written code
- e) Viva-Voce Questions and Answers
- f) Errors observed (if any) during compilation/execution
- g) Signature of the Faculty



#### 4. Guide Lines to Students in Lab

Students are advised to maintain discipline and follow the guidelines given below:

- Keep all your bags in the racks and carry the observation book and record book.
- Mobile phones/pen drives/ CDs are not allowed in the labs.
- Maintain proper dress code along with ID Card
- Occupy the computers allotted to you and maintain the discipline.
- Student must submit the record with the last week experiment details and observation book with the brief of the present experiment.
- Read the write up of the experiment given in the manual.
- Students must use the equipment with care. Any damage is caused student is punishable
- After completion of every experiment, the observation notes to be shown to the lab in - charge and after correction the record must be updated and submit to the lab in charge for correction.
- Lab marks are given on Continuous Evaluation Basis as per GNITC(A) guidelines
- If any student is absent for any lab, they need to be complete the same experiment in the free time before attending next lab session.

Steps to perform experiments in the lab by the student

Step1: Students have to write the Date, aim, Software and Hardware requirements for the scheduled experiment in the observation book.

Step2: Students have to listen and understand the experiment explained by the faculty and note down the important points in the observation book.

Step3: Students need to write procedure/algorithm in the observation book.

Step4: Analyze and Develop/implement the logic of the program by the student in respective platform

Step5: After approval of logic of the experiment by the faculty then the experiment has to be executed on the system.

Step6: After successful execution, the results have to be recorded in the observation book and shown to the lab in charge faculty..

Step7: Students need to attend the Viva-Voce on that experiment and write the same in the observation book.

Step8: Update the completed experiment in the record and submit to the concerned faculty in-charge.

Instructions to maintain the record

- Before starting of the first lab session students must buy the record book and bring the same to the lab.
- Regularly (Weekly) update the record after completion of the experiment and get it corrected with concerned lab in-charge for continuous evaluation.
- In case the record is lost, inform on the same day to the faculty in charge and submit the new record within 2 days for correction.
- If record is not submitted in time or record is not written properly, the record evaluation marks (5M) will be reduced accordingly.

**Awarding the marks for day to day evaluation:**

Total marks for day to day evaluation is 10 Marks  
as per JNTUH. These 15 Marks are distributed as:

Record	3 Marks
Exp setup/program written and execution	5 Marks
Result and Viva-Voce	2 Marks

**Allocation of Marks for Lab Internal**

Total marks for lab internal are 40 Marks as per GNITC (AUTONOMOUS).

These 40 Marks are distributed as:

**Average of day to day evaluation marks: 10 Marks**

**Lab Mid exam: 10 Marks**

**Viva Marks: 10 Marks**

**Additional lab project: 10 Marks**

**Allocation of Marks for Lab External**

Total marks for lab External are 60 Marks as per GNITC AUTONOMOUS).

These 60 Marks are distributed as:

**Program Written: 10 Marks**

**Program Execution and Result: 30 Marks**

**Viva-Voce: 10 Marks**

**Record: 10 Marks**

### 5. List of Lab Exercises:

S. No	Name of the experiment	Page No
1	Write C programs to simulate the following CPU Scheduling algorithms a) FCFS b) SJF c) Round Robin d) priority	13-21
2	Write programs using the I/O system calls of UNIX/LINUX operating system (open, read, write, close, fcntl, seek, stat, opendir, readdir)	22-26
3	Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention	27-36
4	Write a C program to implement the Producer – Consumer problem using Semaphores using UNIX/LINUX system calls.	37-39
5	Write C programs to illustrate the following IPC mechanisms a) Pipes b) FIFOs c) Message Queues d) Shared Memory	40-45
6	Write C programs to simulate the following memory management techniques a) Paging b) Segmentation	46-50
7	Write C programs to simulate Page replacement policies a) FCFS b) LRU c) Optimal	51-59

## 6. List of additional experiments for the semester

S. No	Name of the experiment	
1	Write a C program to Demonstrate Disk Scheduling a) FCFS.b)SSTF c)SCAN d) C-SCAN	60-63

### TEXT BOOKS:

1. Operating System Principles- Abraham Silberchatz, Peter B. Galvin, Greg Gagne 7th Edition, John Wiley
2. Advanced programming in the Unix environment, W.R.Stevens, Pearson education.

### REFERENCE BOOKS:

1. Operating Systems – Internals and Design Principles, William Stallings, Fifth Edition–2005, Pearson Education/PHI
2. Operating System - A Design Approach-Crowley, TMH.
3. Modern Operating Systems, Andrew S Tanenbaum, 2nd edition, Pearson/PHI
4. UNIX Programming Environment, Kernighan and Pike, PHI/Pearson Education
5. UNIX Internals: The New Frontiers, U. Vahalia, Pearson Education

## **7. Content of Lab Experiments**

### **WEEK-1:**

Simulate the following CPU scheduling algorithms

- a. FCFS
- b. SJF
- c. RR
- d. Priority

#### **1 a. First Come First Serve Scheduling Algorithms**

AIM: To Simulate the First Come First Serve Scheduling Algorithms.

Recommended Hardware/Software Requirements:

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- Software Requirements : Linux/unix
- Prerequisite:

Theory:

First-Come, First-Serve scheduling (FCFS): In this, which process enter the ready queue first is served first. The OS maintains DS that is ready queue. It is the simplest CPU scheduling algorithm. If a process request the CPU then it is loaded into the ready queue, which process is the head of the ready queue, connect the CPU to that process.

ALGORITHM:

Step-1- Input the processes along with their burst time (bt).

Step-2- Find waiting time (wt) for all processes.

Step-3- As first process that comes need not to wait so waiting time for process 1 will be 0 i.e.  $wt[0] = 0$ .

Step-4- Find waiting time for all other processes i.e. for process i ->  
 $wt[i+1] = bt[i] + wt[i]$ .

Step-5- Find turnaround time  $tt[i+1]=tt[i]+bt[i+1]$

Step-6- Find average waiting time =  $\text{total\_waiting\_time} / \text{no\_of\_processes}$ .

Step-7- Similarly, find average turnaround time =  $\text{total\_turn\_around\_time} / \text{No\_of\_processes}$ .

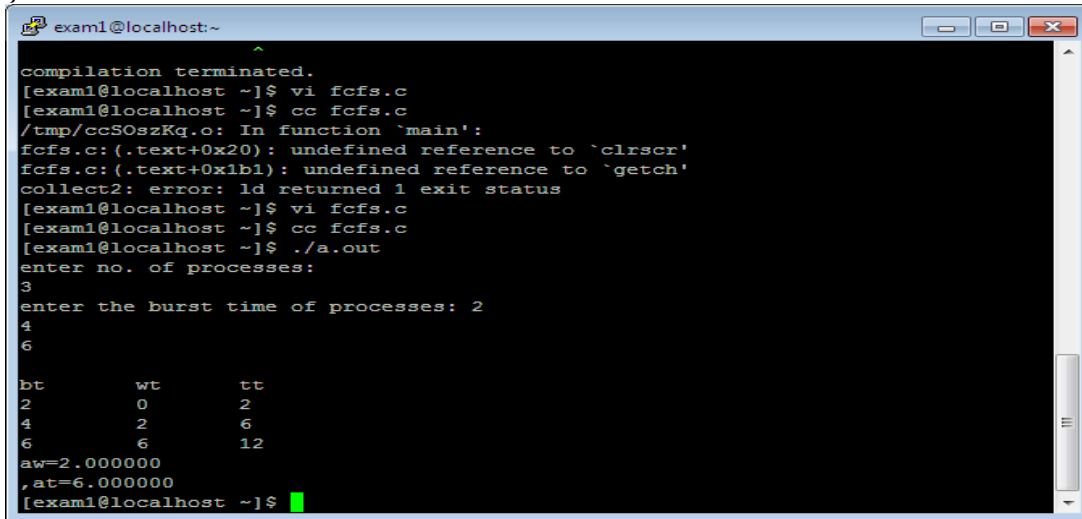
PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,bt[10],n,wt[10],tt[10],w1=0,t1=0;
float aw,at;
clrscr();
printf("enter no. of processes:\n");
scanf("%d",&n);
printf("enter the burst time of processes:\n");
for(i=0;i<n;i++)
scanf("%d",&bt[i]);
for(i=0;i<n;i++)
{
wt[0]=0;
tt[0]=bt[0];
wt[i+1]=bt[i]+wt[i];
```

```

tt[i+1]=tt[i]+bt[i+1];
w1=w1+wt[i];
t1=t1+tt[i];
}
aw=w1/n;
at=t1/n;
printf("\nbt\t wt\t tt\n");
for(i=0;i<n;i++)
printf("%d\t %d\t %d\n",bt[i],wt[i],tt[i]);
printf("aw=%f\nat=%f\n",aw,at);
getch();
}

```



```

exam1@localhost:~
^
compilation terminated.
[exam1@localhost ~]$ vi fcfs.c
[exam1@localhost ~]$ cc fcfs.c
/tmp/ccS0szKq.o: In function `main':
fcfs.c:(.text+0x20): undefined reference to `clrscr'
fcfs.c:(.text+0x1b1): undefined reference to `getch'
collect2: error: ld returned 1 exit status
[exam1@localhost ~]$ vi fcfs.c
[exam1@localhost ~]$ cc fcfs.c
[exam1@localhost ~]$ ./a.out
enter no. of processes:
3
enter the burst time of processes: 2
4
6

bt      wt      tt
2       0       2
4       2       6
6       6       12
aw=2.000000
,at=6.000000
[exam1@localhost ~]$

```

### VIVA QUESTIONS

1. What is First-Come-First-Served (FCFS) Scheduling?
2. Why CPU scheduling is required?
3. Which technique was introduced because a single job could not keep both the CPU and the I/O devices busy?
4. By using which attribute CPU performance is measured?
5. Which of the following is a criterion to evaluate a scheduling algorithm?

### 1 b. Shortest Job First Scheduling Algorithms.

AIM: To Simulate the Shortest Job First Scheduling Algorithms.

Recommended Hardware/Software Requirements:

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- Software Requirements : Linux/unix

Theory:

Shortest job First: The criteria of this algorithm are which process having the smallest CPU burst, CPU is assigned to that next process. If two process having the same CPU burst Time FCFS is used to break the tie.

### ALGORITHM

Step 1: Start the process

Step 2: Declare the array size

Step 3: Get the number of elements to be inserted

Step 4: Select the process which have shortest burst will execute first

Step 5: If two processes have same burst length then FCFS scheduling algorithm used

Step 6: Make the average waiting the length of next process

Step 7: Start with the first process from its selection as above and let other process to be in queue

Step 8: Calculate the total number of burst time

Step 9: Display the values

Step 10: Stop the process

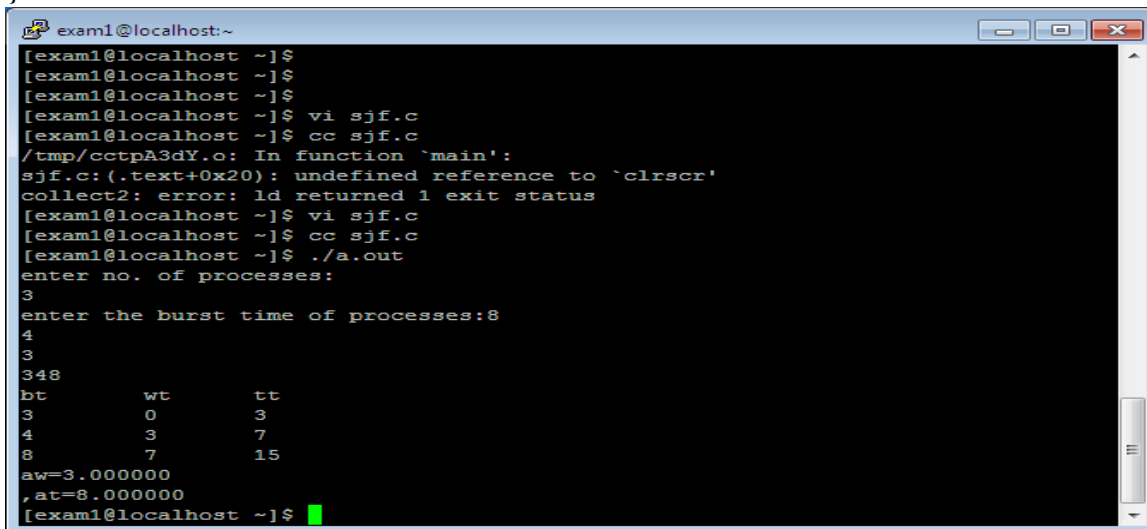
### PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,bt[10],t,n,wt[10],tt[10],w1=0,t1=0;
float aw,at;
clrscr();
printf("enter no. of processes:\n");
scanf("%d",&n);
printf("enter the burst time of processes:\n");
for(i=0;i<n;i++)
scanf("%d",&bt[i]);
for(i=0;i<n;i++) {
for(j=i;j<n;j++)
if(bt[i]>bt[j])
{
t=bt[i];
bt[i]=bt[j];
bt[j]=t;
}
}
```

```

for(i=0;i<n;i++)
printf("%d",bt[i]);
for(i=0;i<n;i++) {
wt[0]=0;
tt[0]=bt[0];
wt[i+1]=bt[i]+wt[i];
tt[i+1]=tt[i]+bt[i+1];
w1=w1+wt[i];
t1=t1+tt[i];
}
aw=w1/n;
at=t1/n;
printf("\nbt\t wt\t tt\n");
for(i=0;i<n;i++)
printf("%d\t %d\t %d\n",bt[i],wt[i],tt[i]);
printf("aw=%f\nat=%f\n",aw,at);
getch();
}

```



```

exam1@localhost:~$
[exam1@localhost ~]$
[exam1@localhost ~]$
[exam1@localhost ~]$ vi sjf.c
[exam1@localhost ~]$ cc sjf.c
/tmp/ccTpA3dY.o: In function 'main':
sjf.c:(.text+0x20): undefined reference to 'clrscr'
collect2: error: ld returned 1 exit status
[exam1@localhost ~]$ vi sjf.c
[exam1@localhost ~]$ cc sjf.c
[exam1@localhost ~]$ ./a.out
enter no. of processes:
3
enter the burst time of processes:8
4
3
348
bt      wt      tt
3        0        3
4        3        7
8        7       15
aw=3.000000
,at=8.000000
[exam1@localhost ~]$

```

### VIVA QUESTIONS:

- 1) What is the optimum CPU scheduling algorithm?
- 2) In terms of average wait time, which scheduling algorithm is optimum?
- 3) What are the disadvantages of SJF Scheduling Algorithm?
- 4) What are the advantages of SJF Scheduling Algorithm?
- 5) Define CPU Scheduling algorithm.



## 1 c. Round Robin CPU Scheduling Algorithms

AIM: To Simulate the Round Robin CPU Scheduling Algorithms.

Recommended Hardware/Software Requirements:

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- Software Requirements : Linux/unix

Theory:

Round Robin: It is a primitive scheduling algorithm it is designed especially for timesharing systems. In this, the CPU switches between the processes. When the time quantum expired, the CPU switches to another job. A small unit of time called a quantum or time slice. A time quantum is generally is a circular queue new processes are added to the tail of the ready queue.

If the process may have a CPU burst of less than one time slice then the process release the CPU voluntarily. The scheduler will then process to next process ready queue otherwise; the process will be put at the tail of the ready queue.

ALGORITHM:

Step 1: The queue structure in ready queue is of First in First out (FIFO) type.

Step 2: A fixed time is allotted to every process that arrives in the queue. This fixed time is known as time slice or time quantum.

Step 3: The first process that arrives is selected and sent to the processor for execution. If it is not able to complete its execution within the time quantum provided, then an interrupt is generated using an automated timer.

Step 4: The process is then stopped and is sent back at the end of the queue. However, the state is saved and context is thereby stored in memory. This helps the process to resume from the point where it was interrupted.

Step 5: The scheduler selects another process from the ready queue and dispatches it to the processor for its execution. It is executed until the time Quantum does not exceed.

Step 6: The same steps are repeated until all the process are finished.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int st[10],bt[10],wt[10],tat[10],n,tq;
int i,count=0,swt=0,stat=0,temp,sq=0;
float awt=0.0,atat=0.0;
clrscr();
printf("Enter number of processes:");
scanf("%d",&n);
printf("Enter burst time for sequences:");
for(i=0;i<n;i++) {
scanf("%d",&bt[i]);
st[i]=bt[i];
}
printf("Enter time quantum:");
scanf("%d",&tq);
while(1)
{
for(i=0,count=0;i<n;i++)
{
```

```

temp=tq;
if(st[i]==0)
{
count++;
continue;
}
if(st[i]>tq)
st[i]=st[i]-tq;
else if(st[i]>=0)
{
temp=st[i];
st[i]=0;
}
sq=sq+temp;
tat[i]=sq;
}
if(n==count)
break;
}
for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
swt=swt+wt[i];
stat=stat+tat[i];
}
awt=(float)swt/n;
atat=(float)stat/n;
printf("P_no\tBT\tWT\tTAT");
for(i=0;i<n;i++)
printf("\n%d\t %d\t %d\t %d",i+1,bt[i],wt[i],tat[i]);
printf("\nAvg wait time is %f\nAvg turn around time is %f",awt,atat);
getch();
}

```

```

exam1@localhost:~$
exam1@localhost:~$
exam1@localhost:~$
exam1@localhost:~$ vi sjf.c
exam1@localhost:~$ cc sjf.c
/tmp/cc7pA3dY.o: In function 'main':
sjf.c:(.text+0x20): undefined reference to 'clrscr'
collect2: error: ld returned 1 exit status
exam1@localhost:~$ vi sjf.c
exam1@localhost:~$ cc sjf.c
exam1@localhost:~$ ./a.out
enter no. of processes:
3
enter the burst time of processes:8
4
3
348
bt      wt      tt
3        0        3
4        3        7
8        7       15
aw=3.000000
,at=8.000000
exam1@localhost:~$

```

### VIVA QUESTIONS:

- Round Robin scheduling is used in  
(A) Disk scheduling. (B) CPU scheduling  
(C) I/O scheduling. (D) Multitasking
- What are the dis-advantages of RR Scheduling Algorithm?
- What are the advantages of RR Scheduling Algorithm?
- Super computers typically employ \_\_\_\_\_.  
(A) Real time Operating system  
(B) Multiprocessors OS  
(C) Desktop OS  
(D) None of the above

### 1 d. Priority Scheduling Algorithms

AIM: To Simulate the Priority Scheduling Algorithms.

Recommended Hardware/Software Requirements:

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- Software Requirements : Linux/unix

Theory:

Priority Scheduling: The cpu is allocated to the process with the highest priority. Equal priority processes are scheduled in the FCFS order. Priorities are generally some fixed range of numbers such as 0 to 409. The low numbers represent high priority

PROGRAM:

```

#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,pno[10],prior[10],bt[10],n,wt[10],tt[10],w1=0,t1=0,s;
float aw,at;
clrscr();

```

```

printf("enter the number of processes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("The process %d:\n",i+1);
printf("Enter the burst time of processes:");
scanf("%d",&bt[i]);
printf("Enter the priority of processes %d:",i+1);
scanf("%d",&prior[i]);
pno[i]=i+1;
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(prior[i]<prior[j])
{
s=prior[i];
prior[i]=prior[j];
prior[j]=s;
s=bt[i];
bt[i]=bt[j];
bt[j]=s;
s=pno[i];
pno[i]=pno[j];
pno[j]=s;
}
}
}
for(i=0;i<n;i++)
{
wt[0]=0;
tt[0]=bt[0];
wt[i+1]=bt[i]+wt[i];
tt[i+1]=tt[i]+bt[i+1];
w1=w1+wt[i];
t1=t1+tt[i];
aw=w1/n;
at=t1/n;
}
printf(" \n job \t bt \t wt \t tat \t prior\n");
for(i=0;i<n;i++)
printf("%d \t %d \t %d \t %d \t %d\n",pno[i],bt[i],wt[i],tt[i],prior[i]);
printf("aw=%f\nat=%f\n",aw,at);
getch();
}

```

```
exam1@localhost:~  
Avg wait time is 8.333333 Avg turn around time is 13.333333[exam1@localhost ~]$  
vi pri.c  
[exam1@localhost ~]$ vi pri.c  
[exam1@localhost ~]$ cc pri.c  
[exam1@localhost ~]$ ./a.out  
-bash: ./a.out: No such file or directory  
[exam1@localhost ~]$ ./a.out  
enter the number of processes:3  
The process 1:  
Enter the burst time of processes:8  
Enter the priority of processes 1:7  
The process 2:  
Enter the burst time of processes:6  
Enter the priority of processes 2:2  
The process 3:  
Enter the burst time of processes:4  
Enter the priority of processes 3:0  
  
  job    bt    wt    tat    prior  
3      4      0      4      0  
2      6      4     10      2  
1      8     10     18      7  
aw=4.000000    at=10.000000  
[exam1@localhost ~]$
```

### VIVA QUESTIONS:

1. In which OS we would like to use Priority CPU scheduling?
2. How much CPU is allocated to Priority Scheduling?
3. How to calculate average waiting time?
4. How you can achieve Maximum CPU Utilization?
5. By using what algorithms we can find minimum and maximum time?

## WEEK-2

*Write programs using the I/O system calls of UNIX/LINUX operating system (open, read, write, close, fcntl, seek, stat, opendir, readdir)*

**Aim:** C program using open, read, write, close system calls

### Theory:

There are 5 basic system calls that Unix provides for file I/O.

1. **Create:** Used to Create a new empty file

**Syntax:** `int creat(char *filename, mode_t mode)`

filename : name of the file which you want to create

mode : indicates permissions of new file.

2. **open:** Used to Open the file for reading, writing or both.

**Syntax:** `int open(char *path, int flags [ , int mode ] );`

Path : path to file which you want to use

flags : How you like to use

O\_RDONLY: read only, O\_WRONLY: write only, O\_RDWR: read and write, O\_CREAT: create file if it doesn't exist, O\_EXCL: prevent creation if it already exists

3. **close:** Tells the operating system you are done with a file descriptor and Close the file which pointed by fd.

**Syntax:** `int close(int fd);`

fd :file descriptor

4. **read:** From the file indicated by the file descriptor fd, the read() function reads cnt bytes of input into the memory area indicated by buf. A successful read() updates the access time for the file.

**Syntax:** `int read(int fd, char *buf, int size);`

fd: file descriptor

buf: buffer to read data from

cnt: length of buffer

5. **write:** Writes cnt bytes from buf to the file or socket associated with fd. cnt should not be greater than INT\_MAX (defined in the limits.h header file). If cnt is zero, write() simply returns 0 without attempting any other action.

**Syntax:** `int write(int fd, char *buf, int size);`

fd: file descriptor

buf: buffer to write data to

cnt: length of buffer

**\*File descriptor** is integer that uniquely identifies an open file of the process.

## ***Algorithm***

1. Start the program.
2. Open a file for O\_RDWR for R/W, O\_CREAT for creating a file, O\_TRUNC for truncate a file.
3. Using getchar(), read the character and stored in the string[] array.
4. The string [] array is write into a file close it.
5. Then the first is opened for read only mode and read the characters and displayed it and close the file.
6. Stop the program.

## **Program**

```
#include<sys/stat.h>
#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
int main()
{
    int n,i=0;
    int f1,f2;
    char c,strin[100];
    f1=open("data",O_RDWR|O_CREAT|O_TRUNC);
    while((c=getchar())!='\n')
    {
        strin[i++]=c;

    }
    strin[i]='\0';
    write(f1,strin,i);
    close(f1);
    f2=open("data",O_RDONLY);
    read(f2,strin,0);
    printf("\n%s\n",strin);
    close(f2);
    return 0;
}
```

## ***Output:***

Hai  
Hai

**b) Aim:** C program using lseek

**Theory:**

lseek is a system call that is used to change the location of the read/write pointer of a file descriptor. The location can be set either in absolute or relative terms.

**Syntax :** off\_t lseek(int fildes, off\_t offset, int whence);

int fildes : The file descriptor of the pointer that is going to be moved.

off\_t offset : The offset of the pointer (measured in bytes).

int whence : Legal values for this variable are provided at the end which are SEEK\_SET (Offset is to be measured in absolute terms), SEEK\_CUR (Offset is to be measured relative to the current location of the pointer), SEEK\_END (Offset is to be measured relative to the end of the file)

**Algorithm:**

1. Start the program
2. Open a file in read mode
3. Read the contents of the file
4. Use lseek to change the position of pointer in the read process
5. Stop

**Program:**

```
#include<stdio.h>
```

```
#include <unistd.h>
```

```
#include <fcntl.h>
```

```
#include <sys/types.h>
```

```
int main()
```

```
{
```

```
    int file=0;
```

```
    if((file=open("testfile.txt",O_RDONLY)) < -1)
```

```
        return 1;
```

```
    char buffer[19];
```

```
    if(read(file,buffer,19) != 19) return 1;
```

```
    printf("%s\n",buffer);
```

```
    if(lseek(file,10,SEEK_SET) < 0) return 1;
```

```
    if(read(file,buffer,19) != 19) return 1;
```

```
    printf("%s\n",buffer);
```

```
    return 0;
```

```
}
```

**Output:**

```
[188rla0501@localhost ~]$ vi testfile.txt
[188rla0501@localhost ~]$ cat testfile.txt
sdfghjkk;l;mnbbvrt;yuijnnb
ggtyuyjjg

[188rla0501@localhost ~]$ gcc w2c.c
[188rla0501@localhost ~]$ ./a.out
sdfghjkk;l;mnbbvrt;yu
mnbbvrt;yuijnnb
ggty
```



c) **Aim:** C program using opendir(), closedir(), readdir()

## Theory:

The following are the various operations using directories

1. Creating directories.

**Syntax :** int mkdir(const char \*pathname, mode\_t mode);

2. The 'pathname' argument is used for the name of the directory.

3. Opening directories

**Syntax :** DIR \*opendir(const char \*name);

4. Reading directories.

**Syntax:** struct dirent \*readdir(DIR \*dirp);

5. Removing directories.

**Syntax:** int rmdir(const char \*pathname);

6. Closing the directory.

**Syntax:** int closedir(DIR \*dirp);

7. Getting the current working directory.

**Syntax:** char \*getcwd(char \*buf, size\_t size);

## Algorithm:

1. Start the program
2. Print a menu to choose the different directory operations
3. To create and remove a directory ask the user for name and create and remove the same respectively.
4. To open a directory check whether directory exists or not. If yes open the directory .If it does not exists print an error message.
5. Finally close the opened directory.
6. Stop

## Program:

```
#include<stdio.h>
#include<fcntl.h>
#include<dirent.h>
main()
{
char d[10]; int c,op; DIR *e;
struct dirent *sd;
printf("**menu**\n1.create dir\n2.remove dir\n 3.read dir\n enter ur choice");
scanf("%d",&op);
switch(op)
{
case 1: printf("enter dir name\n"); scanf("%s",&d);
c=mkdir(d,777);
if(c==1)
printf("dir is not created");
else
printf("dir is created"); break;
case 2: printf("enter dir name\n"); scanf("%s",&d);
```

```

c=rmdir(d);
if(c==1)
printf("dir is not removed");
else
printf("dir is removed"); break;
case 3: printf("enter dir name to open");
scanf("%s",&d);
e=opendir(d);
if(e==NULL)
printf("dir does not exist"); else
{
printf("dir exist\n"); while((sd=readdir(e))!=NULL) printf("%s\t",sd->d_name);
}
closedir(e);
break;
}
}

```

### ***Output:***

```

[188r1a0501@localhost f]$ gcc w2e.c
[188r1a0501@localhost f]$ ./a.out
**menu**
1.create dir
2.remove dir
3.read dir
enter ur choice1
enter dir name
d
dir is created[188r1a0501@localhost f]$ ls
a.out a.txt d w2d.c w2e.c

```

### WEEK -3

Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention

#### a) Aim

Write a C program to simulate the Bankers Algorithm for Deadlock Avoidance.

#### Data structures

1. n- Number of process, m-number of resource types.
2. Available: Available[j]=k, k – instance of resource type R<sub>j</sub> is available.
3. Max: If max [i, j]=k, P<sub>i</sub> may request at most k instances resource R<sub>j</sub>.
4. Allocation: If Allocation [i, j]=k, P<sub>i</sub> allocated to k instances of resource R<sub>j</sub>
5. Need: If Need[I, j]=k, P<sub>i</sub> may need k more instances of resource type R<sub>j</sub>,
6. Need [I, j] =Max [I, j]-Allocation [I, j];

#### Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
2. Find an i such that both
3. Finish[i] =False
4. Need<=Work
5. If no such I exist go to step 4.
6. work=work+Allocation, Finish[i] =True;
7. If Finish [1] =True for all I, then the system is in safe state.

#### Resource request algorithm

1. Let Request i be request vector for the process P<sub>i</sub>, If request i=[j]=k, then process P<sub>i</sub> wants k instances of resource type R<sub>j</sub>.
2. If Request<=Need I go to step 2. Otherwise raise an error condition.
3. If Request<=Available go to step 3. Otherwise P<sub>i</sub> must since the resources are available.
4. Have the system pretend to have allocated the requested resources to process P<sub>i</sub> by modifying the state as follows;
5. Available=Available-Request I;
6. Allocation I =Allocation+Request I;
7. Need i=Need i-Request I;

If the resulting resource allocation state is safe, the transaction is completed and process P<sub>i</sub> is allocated its resources. However, if the state is unsafe, the P<sub>i</sub> must wait for Request i and the old resource-allocation state is restore.

**Algorithm:**

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether it is possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. Or not if we allow the request.
10. Stop the program.

**Program:**

```
#include<stdio.h>

int main ()
{
    int allocated[15][15], max[15][15], need[15][15], avail[15], tres[15],
        work[15], flag[15];
    int pno, rno, i, j, prc, count, t, total;
    count = 0;
    //clrscr ();

    printf ("\n Enter number of process:");
    scanf ("%d", &pno);
    printf ("\n Enter number of resources:");
    scanf ("%d", &rno);
    for (i = 1; i <= pno; i++)
    {
        flag[i] = 0;
    }
    printf ("\n Enter total numbers of each resources:");
    for (i = 1; i <= rno; i++)
        scanf ("%d", &tres[i]);

    printf ("\n Enter Max resources for each process:");
    for (i = 1; i <= pno; i++)
    {
```

```

    printf ("\n for process %d:", i);
    for (j = 1; j <= rno; j++)
        scanf ("%d", &max[i][j]);
}

printf ("\n Enter allocated resources for each process:");
for (i = 1; i <= pno; i++)
{
    printf ("\n for process %d:", i);
    for (j = 1; j <= rno; j++)
        scanf ("%d", &allocated[i][j]);

}

printf ("\n available resources:\n");
for (j = 1; j <= rno; j++)
{
    avail[j] = 0;
    total = 0;
    for (i = 1; i <= pno; i++)
    {
        total += allocated[i][j];
    }
    avail[j] = tres[j] - total;
    work[j] = avail[j];
    printf ("    %d \t", work[j]);
}

do
{

    for (i = 1; i <= pno; i++)
    {
        for (j = 1; j <= rno; j++)
        {
            need[i][j] = max[i][j] - allocated[i][j];
        }
    }
}

```

```

printf ("\n Allocated matrix      Max      need");
for (i = 1; i <= pno; i++)
{
    printf ("\n");
    for (j = 1; j <= rno; j++)
    {
        printf ("%4d", allocated[i][j]);
    }
    printf ("|");
    for (j = 1; j <= rno; j++)
    {
        printf ("%4d", max[i][j]);
    }
    printf ("|");
    for (j = 1; j <= rno; j++)
    {
        printf ("%4d", need[i][j]);
    }
}

prc = 0;

for (i = 1; i <= pno; i++)
{
    if (flag[i] == 0)
    {
        prc = i;

        for (j = 1; j <= rno; j++)
        {
            if (work[j] < need[i][j])
            {
                prc = 0;
                break;
            }
        }
    }
}
if (prc != 0)
    break;
}

```

```

if (prc != 0)
{
    printf ("\n Process %d completed", i);
    count++;
    printf ("\n Available matrix:");
    for (j = 1; j <= rno; j++)
    {
        work[j] += allocated[prc][j];
        allocated[prc][j] = 0;
        max[prc][j] = 0;
        flag[prc] = 1;
        printf (" %d", work[j]);
    }
}

}

while (count != pno && prc != 0);

if (count == pno)
    printf ("\nThe system is in a safe state!!");
else
    printf ("\nThe system is in an unsafe state!!");
return 0;

}

```

## Output:

```
[188r1a0501@localhost ~]$ vi dp.c
[188r1a0501@localhost ~]$ gcc dp.c
[188r1a0501@localhost ~]$ ./a.out

Enter number of process:5

Enter number of resources:3

Enter total numbers of each resources:10      5      7

Enter Max resources for each process:
for process 1:7      5      3
for process 2:3      2      2
for process 3:9      0      2
for process 4:2      2      2
for process 5:4      3      3

Enter allocated resources for each process:
for process 1:0      1      0
for process 2:2      0      0
for process 3:3      0      2
for process 4:2      1      1
for process 5:0      0      2
```



available resources:

3			3			2		
Allocated matrix						Max	need	
0	1	0	7	5	3	7	4	3
2	0	0	3	2	2	1	2	2
3	0	2	9	0	2	6	0	0
2	1	1	2	2	2	0	1	1
0	0	2	4	3	3	4	3	1

Process 2 completed

5			3			2		
Allocated matrix						Max	need	
0	1	0	7	5	3	7	4	3
0	0	0	0	0	0	0	0	0
3	0	2	9	0	2	6	0	0
2	1	1	2	2	2	0	1	1
0	0	2	4	3	3	4	3	1

Process 4 completed

7			4			3		
Allocated matrix						Max	need	
0	1	0	7	5	3	7	4	3
0	0	0	0	0	0	0	0	0
3	0	2	9	0	2	6	0	0
0	0	0	0	0	0	0	0	0
0	0	2	4	3	3	4	3	1

Process 1 completed

7			5			3		
Allocated matrix						Max	need	
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
3	0	2	9	0	2	6	0	0
0	0	0	0	0	0	0	0	0
0	0	2	4	3	3	4	3	1

Process 3 completed

10			5			5		
Allocated matrix						Max	need	
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	2	4	3	3	4	3	1

Process 5 completed

10			5			7		
----	--	--	---	--	--	---	--	--

**b) Aim**

Write a C program to simulate Bankers Algorithm for Deadlock Prevention

**Algorithm:**

1. Start
2. Attacking Mutex condition : never grant exclusive access. but this may not be possible for several resources.
3. Attacking preemption: not something you want to do.
4. Attacking hold and wait condition : make a process hold at the most 1 resource at a time. make all the requests at the beginning. All or nothing policy. If you feel, retry. eg. 2-phase locking 34
5. Attacking circular wait: Order all the resources. Make sure that the requests are issued in the correct order so that there are no cycles present in the resource graph. Resources numbered 1 ... n. Resources can be requested only in increasing order. ie. you cannot request a resource whose no is less than any you may be holding.
6. Stop

**Program:**

```
#include<stdio.h>

int max[10][10],alloc[10][10],need[10][10],avail[10],i,j,p,r,finish[10]={0},flag=0;
main( )
{

printf("\n SIMULATION OF DEADLOCK PREVENTION \n ");
printf("Enter no. of processes, resources\n ");
scanf("%d%d",&p,&r);
printf("Enter allocation matrix");
for(i=0;i<p;i++)
for(j=0;j<r;j++)
scanf("%d",&alloc[i][j]);
printf("\n enter max matrix");
for(i=0;i<p;i++) /*reading the maximum matrix and available matrix*/
for(j=0;j<r;j++)
scanf("%d",&max[i][j]);
printf(" \n enter available matrix");
for(i=0;i<r;i++)
scanf("%d",&avail[i]);
```

```

for(i=0;i<p;i++)
for(j=0;j<r;j++)
need[i][j]=max[i][j]-alloc[i][j];
fun(); /*calling function*/
if(flag==0)
{ if(finish[i]!=1)
{
printf("\n Failing :Mutual exclusion");
for(j=0;j<r;j++)
{ /*checking for mutual exclusion*/
if(avail[j]<need[i][j])
avail[j]=need[i][j];
}fun();
printf("\n By allocating required resources to process %d dead lock is prevented ",i);
printf("\n lack of preemption");
for(j=0;j<r;j++)
{
if(avail[j]<need[i][j])
avail[j]=need[i][j];
alloc[i][j]=0;
}
fun( );
printf("\n dead lock is prevented by allocating needed resources");

printf(" \n failing:Hold and Wait condition ");
for(j=0;j<r;j++)
{ /*checking hold and wait condition*/
if(avail[j]<need[i][j])
avail[j]=need[i][j];
}
fun( );
printf("\n AVOIDING ANY ONE OF THE CONDITION, U CAN PREVENT DEADLOCK");
}
}
}
fun()
{
while(1)
{
for(flag=0,i=0;i<p;i++)

```

```

{
if(finish[i]==0)
{
for(j=0;j<r;j++)
{
if(need[i][j]<=avail[j])
continue;
else
break;
}
if(j==r)
{
for(j=0;j<r;j++)
avail[j]+=alloc[i][j];
flag=1;
finish[i]=1;
}
}
}
}

```

## Output:

```

[188r1a0501@localhost ~]$ vi dp1.c
[188r1a0501@localhost ~]$ gcc dp1.c
[188r1a0501@localhost ~]$ ./a.out

SIMULATION OF DEADLOCK PREVENTION
Enter no. of processes, resources
3
2
Enter allocation matrix4
5
3
4
5
2

enter max matrix4
3
4
5
6
1

enter available matrix2
5

Failing :Mutual exclusion
By allocating required resources to process 3 dead lock is prevented
lack of preemption
dead lock is prevented by allocating needed resources
failing:Hold and Wait condition
AVOIDING ANY ONE OF THE CONDITION, U CAN PREVENT DEADLOCK[188r1a0501@localhost ~]$ █

```

## **WEEK-4**

Write a C program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

### ***Aim:***

Write a C program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

### **Algorithm:**

1. The Semaphore mutex, full & empty are initialized.
2. In the case of producer process
3. Produce an item in to temporary variable.  
If there is empty space in the buffer check the mutex value for enter into the critical section.  
If the mutex value is 0, allow the producer to add value in the temporary variable to the buffer.
4. In the case of consumer process
  - i) It should wait if the buffer is empty
  - ii) If there is any item in the buffer check for mutex value, if the mutex==0, remove item from buffer
  - iii) Signal the mutex value and reduce the empty value by 1.
  - iv) Consume the item.
5. Print the result

### **Program:**

```
#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 ("\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 ("Buffer is full!!");
            break;
        case 2:
            if ((mutex == 1) && (full != 0))
                consumer ();
            else
                printf ("Buffer 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);
    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 ("\nConsumer consumes item %d", x);  
x--;  
mutex = signal (mutex);  
}
```

### ***Output:***

```
[188r1a0501@localhost ~]$ vi pc.c  
[188r1a0501@localhost ~]$ gcc pc.c  
[188r1a0501@localhost ~]$ ./a.out  
  
1.Producer  
2.Consumer  
3.Exit  
Enter your choice:1  
  
Producer produces the item 1  
Enter your choice:1  
  
Producer produces the item 2  
Enter your choice:1  
  
Producer produces the item 3  
Enter your choice:2  
  
Consumer consumes item 3  
Enter your choice:2  
  
Consumer consumes item 2  
Enter your choice:2  
  
Consumer consumes item 1  
Enter your choice:2  
Buffer is empty!!  
Enter your choice:3  
[188r1a0501@localhost ~]$
```

## Week: 5: Write C programs to illustrate the following IPC mechanisms

**Aim:** Write C programs to illustrate the following IPC mechanisms

### **ALGORITHM:**

1. Start the program.
2. Declare the variables.
3. Read the choice.
4. Create a piping processing using IPC.
5. Assign the variable lengths
6. “*strcpy*” the message lengths.
7. To join the operation using IPC .
8. Stop the program

### ***Program : ( PIPE PROCESSING)***

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define MSG_LEN 64
int main()
{
    int result;
    int fd[2];
    char message[MSG_LEN];
    char recvd_msg[MSG_LEN];
    result = pipe (fd);
    //Creating a pipe//fd[0] is for reading and fd[1] is for writing
    if (result < 0)
    {
        perror("pipe ");
        exit(1);
    }

    strncpy(message,"Linux World!! ",MSG_LEN); result=write(fd[1],message,strlen(message)); if (result
    < 0){
        perror("write"); exit(2);
    }
    strncpy(message,"Understanding ",MSG_LEN);
    result=write(fd[1],message,strlen(message));
    if(result < 0)
    {
        perror("write");
        exit(2);
    }

    strncpy(message,"Concepts of ",MSG_LEN);
    result=write(fd[1],message,strlen(message));
    if (result <0)
```



```

{
perror("write");
exit(2);
}

strncpy(message,"Piping ", MSG_LEN);
result=write(fd[1],message,strlen(message));
if (result < 0)
{
perror("write");
exit(2);
}
result=read(fd[0],recvd_msg,MSG_LEN);
if (result < 0)
{
perror("read");
exit(3);
}

printf("%s\n",recvd_msg); return 0;
}

```

## ***a) FIFO***

### ***Program:***

```

#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>

#include <linux/stat.h>

#define FIFO_FILE    "MYFIFO"

int main(void)
{
    FILE *fp;
    char readbuf[80];

    /* Create the FIFO if it does not exist */
    umask(0);
    mknod(FIFO_FILE, S_IFIFO|0666, 0);

    while(1)
    {
        fp = fopen(FIFO_FILE, "r");
        fgets(readbuf, 80, fp);
        printf("Received string: %s\n", readbuf);
        fclose(fp);
    }
}

```

```

    }

    return(0);
}

#include <stdio.h>
#include <stdlib.h>

#define FIFO_FILE    "MYFIFO"

int main(int argc, char *argv[])
{
    FILE *fp;

    if ( argc != 2 ) {
        printf("USAGE: fifoclient [string]\n");
        exit(1);
    }

    if((fp = fopen(FIFO_FILE, "w")) == NULL) {
        perror("fopen");
        exit(1);
    }
    fputs(argv[1], fp);

    fclose(fp);
    return(0);
}

```

#### Program for Message Queue (Writer Process)

```

#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// structure for message queue
struct mesg_buffer {
    long  msg_type;
    char msg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;
    // ftok to generate unique key
    key = ftok("progfile", 65);
    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;
}

```

```

printf("Write Data : ");
gets(message.mesg_text);

// msgsnd to send message
msgsnd(msgid, &message, sizeof(message), 0);

// display the message
printf("Data send is : %s \n", message.mesg_text);

return 0;
}

```

### C Program for Message Queue (Reader Process)

```

#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    // msgrcv to receive message
    msgrcv(msgid, &message, sizeof(message), 1, 0);

    // display the message
    printf("Data Received is : %s \n",
        message.mesg_text);

    // to destroy the message queue
    msgctl(msgid, IPC_RMID, NULL);

    return 0;
}

```

## C Program for Message Queue (Reader Process)

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);

    // msgrcv to receive message
    msgrcv(msgid, &message, sizeof(message), 1, 0);

    // display the message
    printf("Data Received is : %s \n",
        message.mesg_text);

    // to destroy the message queue
    msgctl(msgid, IPC_RMID, NULL);

    return 0;
}
```

**OUTPUT:** Thus the Piping process using IPC program was executed and verified successfully

```
[sree@localhost ~]$ cc pp.c
[sree@localhost ~]$ ./a.out

Enter string:1
os
er
a
tingEnter 1 array elementz:1
The string length=1
Sum=0[sree@localhost ~]$ er
bash: er: command not found
[sree@localhost ~]$ ating1
bash: ating1: command not found
[sree@localhost ~]$ gedit pp.c
[sree@localhost ~]$ cc pp.c
[sree@localhost ~]$ ./a.out
Linux World!!!
[sree@localhost ~]$ gedit pp.c
[sree@localhost ~]$ cc pp.c
[sree@localhost ~]$ ./a.out
Linux World!! Understanding Concepts of Piping ,
[sree@localhost ~]$ █
```

## **Week: 6**

**Aim:** Write C programs to simulate the following memory management techniques

### **a) Paging**

**AIM:** To write a C program to implement memory management using paging technique.

#### **ALGORITHM:**

Step1 : Start the program.

Step2 : Read the base address, page size, number of pages and memory unit.

Step3 : If the memory limit is less than the base address display the memory limit is less than limit.

Step4 : Create the page table with the number of pages and page address.

Step5 : Read the page number and displacement value.

Step6 : If the page number and displacement value is valid, add the displacement value with the address corresponding to the page number and display the result.

Step7 : Display the page is not found or displacement should be less than page size.

Step8 : Stop the program.

#### **Program:**

```
#include<stdio.h>
#include<conio.h>
main()
{

int ms, ps, nop, np, rempages, i, j, x, y, pa, offset; int s[10], fno[10][20];
printf("\nEnter the memory size -- ");
scanf("%d",&ms);
printf("\nEnter the page size -- ");
scanf("%d",&ps);
nop = ms/ps;

printf("\nThe no. of pages available in memory are -- %d ",nop);
printf("\nEnter number of processes -- ");
scanf("%d",&np);
rempages = nop; for(i=1;i<=np;i++)
{
printf("\nEnter no. of pages required for p[%d]-- ",i);
scanf("%d",&s[i]);
if(s[i] > rempages)
{
printf("\nMemory is Full");
break;
}

rempages = rempages - s[i];
printf("\nEnter pagetable for p[%d] --- ",i);
for(j=0;j<s[i];j++)
scanf("%d",&fno[i][j]);
}
```

```

printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset -- ");
scanf("%d %d %d",&x,&y, &offset);
if(x>np || y>=s[i] || offset>=ps)

printf("\nInvalid Process or Page Number or offset");
else
{

pa=fno[x][y]*ps+offset;

printf("\nThe Physical Address is -- %d",pa);

}
getch();

}

```

### ***OUTPUT:***

```

Enter the memory size -- 1000

Enter the page size -- 200

The no. of pages available in memory are -- 5
Enter number of processes -- 2

Enter no. of pages required for p[1]-- 20

Memory is Full
Enter Logical Address to find Physical Address
Enter process no. and pagenumber and offset -- 1
2
5

The Physical Address is -- 5

..Program finished with exit code 0
Press ENTER to exit console.

```

## b) Segmentation

**Aim:** To write a C program to implement memory management using segmentation

### Algorithm:

- Step1 : Start the program.
- Step2 : Read the base address, number of segments, size of each segment, memory limit.
- Step3 : If memory address is less than the base address display “invalid memory limit”.
- Step4 : Create the segment table with the segment number and segment address and display it.
- Step5 : Read the segment number and displacement.
- Step6 : If the segment number and displacement is valid compute the real address and display the same.
- Step7 : Stop the program.

### Program:

```
#include<stdio.h>
#include<conio.h>
struct list
{
int seg;
int base;
int limit;
struct list *next;
} *p;
void insert(struct list *q,int base,int limit,int seg)
{
if(p==NULL)
{
p=malloc(sizeof(Struct list));
p->limit=limit;
p->base=base;
p->seg=seg;
p->next=NULL;
}
else
{
while(q->next!=NULL)
{
Q=q->next;
Printf(“yes”)
}
q->next=malloc(sizeof(Struct list));
q->next->limit=limit;
q->next->base=base;
q->next->seg=seg;
```



```

q->next ->next=NULL;
}
}
int find(struct list *q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
return q->limit;
}
int search(struct list *q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
return q->base;
}
main()
{

p=NULL;
int seg,offset,limit,base,c,s,physical;
printf("Enter segment table/n");
printf("Enter -1 as segment value for termination\n");
do
{
printf("Enter segment number");
scanf("%d",&seg);
if(seg!=-1)
{
printf("Enter base value:");
scanf("%d",&base);

printf("Enter value for limit:");
scanf("%d",&limit);
insert(p,base,limit,seg);
}
}
while(seg!=-1)
printf("Enter offset:");

```

```

scanf("%d",&offset);
printf("Enter bsegmentation
number:");scanf("%d",&seg);
c=find(p,s
eg);
s=search(
p,seg);
if(offset<c
)
{
physical=s+offset;
printf("Address in physical memory %d\n",physical);
}
else
{
printf("error");
}

```

#### OUTPUT:

```

C:\TURBOC3\BIN>TC
Enter segment table
Enter -1 as segmentation value for termination
Enter segment number:1
Enter base value:2000
Enter value for limit:100
Enter segment number:2
Enter base value:2500
Enter value for limit:100
Enter segmentation number:-1
Enter offset:90
Enter segment number:2
Address in physical memory 2590

Enter segment table
Enter -1 as segmentation value for termination
Enter segment number:1
Enter base value:2000
Enter value for limit:100
Enter segment number:2Enter base value:2500

```

7. Write C programs to simulate Page replacement policies  
a. FIFO    b. LRU    c. Optimal

**a. FIFO Page Replacement Algorithm**

**AIM:** To Simulate FIRST IN FIRST OUT Page Replacement Algorithm

**Recommended Hardware/Software Requirements:**

- Hardware Requirements: Intel based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- Software Requirements : Turbo C

**Prerequisite:**

**Theory:**

**a) FIFO (First in First Out) algorithm:** FIFO is the simplest page replacement algorithm, the idea behind this is, “Replace a page that page is oldest page of main memory” or “Replace the page that has been in memory longest”. FIFO focuses on the length of time a page has been in the memory rather than how much the page is being used.

**ALGORITHM**

```
Step1:      Start
Step2:      Global Declaration fifo(),t[5],pgf,n,a[20],I,j,frm
Step3:      pgf  fifo()
Step4:      Read pos,flag,i  0
Step5:      while i<n do
Step6:      for pos=0 to pos<frm & i<n do
Step7:      for j=0 to j<pos or j<frm do
Step8:      if a[i]=t[j] then
              Flag  1, break
Step9:      if flag=1 then continue
Step10:     t [pos]  a[i]
Step11:     for j  0 to j<frm do
              Print t[j]
Step12:     pgf  pgf+1,pos  pos+1
Step13:     return pgf to Step3
Step14:     end
```

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
void main()
{
clrscr();
printf("\n \t\t\t FIFI PAGE REPLACEMENT ALGORITHM");
printf("\n Enter no.of frames....");
scanf("%d",&nof);
printf("Enter number of Pages.\n");
scanf("%d",&nor);
printf("\n Enter the Page No...");
```

```

for(i=0;i<nor;i++)
scanf("%d",&ref[i]);
printf("\nThe given Pages are:");
for(i=0;i<nor;i++)
printf("%4d",ref[i]);
for(i=1;i<=nof;i++)
frm[i]=-1;
printf("\n");
for(i=0;i<nor;i++)
{
flag=0;
printf("\n\t page no %d->\t",ref[i]);
for(j=0;j<nof;j++)
{
if(frm[j]==ref[i])
{
flag=1;
break;
}}
if(flag==0)
{
pf++;
victim++;
victim=victim%nof;
frm[victim]=ref[i];
for(j=0;j<nof;j++)
printf("%4d",frm[j]);
}
}
printf("\n\n\t\t No.of pages faults...%d",pf);
getch();
}

```

## **OUTPUT:**

```
exami@localhost:~$ ./a.out
No.of pages faults...3[exami@localhost ~]$ ./a.out
FIFI PAGE REPLACEMENT ALGORITHM
Enter no.of frames....3
Enter number of Pages.
7
Enter the Page No...2
4
0
0
0
0
1
1
The given Pages are: 2 4 3 0 8 2 1
page no 2-> 2 -1 -1
page no 4-> 2 4 -1
page no 3-> 2 4 3
page no 0-> 0 4 3
page no 8-> 0 8 3
page no 2-> 0 8 2
page no 1-> 1 8 2
No.of pages faults...7[exami@localhost ~]$
```

### **VIVA QUESTIONS:**

1. Define FIFO.
2. Which of the following statement is not true?
  - a) Multiprogramming implies multitasking
  - b) Multi-user does not imply multiprocessing
  - c) Multitasking does not imply multiprocessing
  - d) Multithreading implies multi-user
3. Define page.
4. Define Frame.
5. Write advantages and dis-advantages of FIFO.

## **b. LRU Page Replacement Algorithm**

**AIM:** To Simulate LEAST RECENTLY USED Page Replacement Algorithm

**Recommended Hardware/Software Requirements:**

- Hardware Requirements: Intel based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- Software Requirements : Turbo C

**Prerequisite:**

**Theory:**

**b) LRU (Least Recently Used ):** the criteria of this algorithm is “Replace a page that has been used for the longest period of time”. This strategy is the page replacement algorithm looking backward in time, rather than forward.

**ALGORITHM:**

- Step 1. Start the process
- Step 2. Declare the size
- Step 3. Get the number of pages to be inserted
- Step 4. Get the value
- Step 5. Declare counter and stack
- Step 6. Select the least recently used page by counter value
- Step 7. Stack them according the selection.
- Step 8. Display the values
- Step 9. Stop the process

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
main()
{
int i, j, k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;
clrscr();
printf("Enter the length of reference string -- ");
scanf("%d",&n);
printf("Enter the reference string -- ");
for(i=0;i<n;i++)
{
scanf("%d",&rs[i]);
flag[i]=0;
}
printf("Enter the number of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
{
count[i]=0;
m[i]=-1;
}
printf("\nThe Page Replacement process is -- \n");
for(i=0;i<n;i++)
```

```

{
for(j=0;j<f;j++)
{
if(m[j]==rs[i])
{
flag[i]=1;
count[j]=next;
next++;
}
}
if(flag[i]==0)
{
if(i<f)
{
m[i]=rs[i];
count[i]=next;
next++;
}
else
{
min=0;
for(j=1;j<f;j++)
if(count[min] > count[j])
min=j;
m[min]=rs[i];
count[min]=next;
next++;
}
pf++;
}
for(j=0;j<f;j++)
printf("%d\t", m[j]);
if(flag[i]==0)
printf("PF No. -- %d" , pf);
printf("\n");
}
printf("\nThe number of page faults using LRU are %d",pf);
getch();
}

```

**OUTPUT:**

```

exam1@localhost:~
[exam1@localhost ~]$ vi lru.c
[exam1@localhost ~]$ cc lru.c
[exam1@localhost ~]$ ./a.out
Enter the length of reference string -- 8
Enter the reference string -- 2
3
4
5
0
1
2
7
Enter the number of frames -- 4

The Page Replacement process is --
2      -1      -1      -1      PF No. -- 1
2       3      -1      -1      PF No. -- 2
2       3       4      -1      PF No. -- 3
2       3       4       5      PF No. -- 4
0       3       4       5      PF No. -- 5
0       1       4       5      PF No. -- 6
0       1       2       5      PF No. -- 7
0       1       2       7      PF No. -- 8

The number of page faults using LRU are 8[exam1@localhost ~]$

```

### **VIVA QUESTIONS:**

1. In which of the following page replacement policies, Belady's anomaly occurs?  
(A) FIFO (B) LRU (C) LFU (D) SRU
2. Explain the difference between FIFO and LRU?
3. The operating system manages \_\_\_\_\_.  
(A) Memory (B) Processor (C) Disk and I/O devices (D) All of the above
4. A program at the time of executing is called \_\_\_\_\_.  
(A) Dynamic program (B) Static program (C) Binded Program (D) A Process
5. The principle of locality of reference justifies the use of \_\_\_\_\_.  
(A) Virtual Memory (B) Interrupts (C) Main memory (D) Cache memory

### **c. Optimal Page Replacement Algorithm**

**AIM:** To Simulate Optimal Page Replacement Algorithm

#### **Recommended Hardware/Software Requirements:**

- Hardware Requirements: Intel based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- Software Requirements : Turbo C

#### **Prerequisite:**

#### **Theory:**

#### **c) OPTIMAL Page Replacement:**

Optimal page replacement algorithm says that if page fault occurs then that page should be removed that will not be used for maximum time in future. It is also known as clairvoyant replacement algorithm or Bélády's optimal page replacement policy.

#### **ALGORITHM:**



- Step 1. Start
- 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 will not be used for the longest period of time.
- Step 7. Display the number of page faults.
- Step 8. Stop

**PROGRAM:**

```
#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("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:

```
exam1@localhost:~  
[exam1@localhost ~]$ vi optimal.c  
[exam1@localhost ~]$ cc optimal.c  
[exam1@localhost ~]$ ./a.out  
Enter number of frames: 3  
Enter number of pages: 10  
Enter page reference string: 2  
3  
4  
2  
1  
3  
7  
5  
4  
3  
  
2      -1      -1  
2      3      -1  
2      3      4  
2      3      4  
1      3      4  
1      3      4  
7      3      4  
5      3      4  
5      3      4  
5      3      4  
[exam1@localhost ~]$
```

## VIVA QUESTIONS:

1. What is the Optimal Page replacement?
2. Explain when page replacement occurs?
3. Which is the best page replacement algorithm? Why?
4. Explain various page replacement algorithms?
5. What do you mean by page fault?

### **Additional Program: 1**

**AIM:** Write a C program to Demonstrate Disk Scheduling a) FCFS.b)SSTF c)SCAN d) C-SCAN

### **Recommended Hardware/Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- Software Requirements : Linux/Unix

## **FCFS DISK SCHEDULING ALGORITHM**

### **Program:**

```
#include<stdio.h>
#include<math.h>
void fcfs(int noq, int qu[10], int st)
{
    int i,s=0;
    for(i=0;i<noq;i++)
    {
        s=s+abs(st-qu[i]);
        st=qu[i];
    }
    printf("\n Total seek time :%d",s);
}

void sstf(int noq, int qu[10], int st, int visit[10])
{
    int min,s=0,p,i;
    while(1)
    {
        min=999;
        for(i=0;i<noq;i++)
            if (visit[i] == 0)
            {
                if(min > abs(st - qu[i]))
                {
                    min = abs(st-qu[i]);
                    p = i;
                }
            }
        if(min == 999)
            break;
        visit[p]=1;
        s=s + min;
        st = qu[p];
    }
    printf("\n Total seek time is: %d",s);
}

void scan(int noq, int qu[10], int st, int ch)
```

```

int i,j,s=0;
for(i=0;i<noq;i++)
{
    if(st < qu[i])
    {
        for(j=i-1; j>= 0;j--)
        {
            s=s+abs(st - qu[j]);
            st = qu[j];
        }
        if(ch == 3)
        {
            s = s + abs(st - 0);
            st = 0;
        }
        for(j = 1;j < noq;j++)
        {
            s= s + abs(st - qu[j]);
            st = qu[j];
        }
        break;
    }
}
printf("\n Total seek time : %d",s);
}

```

```

int main()
{
    int n,qu[20],st,i,j,t,noq,ch,visit[20];
    printf("\n Enter the maximum number of cylinders : ");
    scanf("%d",&n);
    printf("enter number of queue elements");
    scanf("%d",&noq);
    printf("\n Enter the work queue");
    for(i=0;i<noq;i++)
    {
        scanf("%d",&qu[i]);
        visit[i] = 0;
    }
    printf("\n Enter the disk head starting posision: \n");
    scanf("%d",&st);
    while(1)
    {
        printf("\n\n\t\t MENU \n");
        printf("\n\n\t\t 1. FCFS \n");
        printf("\n\n\t\t 2. SSTF \n");
        printf("\n\n\t\t 3. SCAN \n");
        printf("\n\n\t\t 4. EXIT \n");
        printf("\nEnter your choice: ");
        scanf("%d",&ch);
        if(ch > 2)

```

```

{
for(i=0;i<noq;i++)
for(j=i+1;j<noq;j++)
if(qu[i]>qu[j])
{
t=qu[i];
qu[i] = qu[j];
qu[j] = t;
}
}
switch(ch)
{
case 1: printf("\n FCFS \n");
printf("\n*****\n");
fcfs(noq,qu,st);
break;

case 2: printf("\n SSTF \n");
printf("\n*****\n");
sstf(noq,qu,st,visit);
break;

case 3: printf("\n SCAN \n");
printf("\n*****\n");
scan(noq,qu,st,ch);
break;

case 4: exit(0);
}
}
}

```

## Output:

```

exami@localhost:~
[exami@localhost ~]$ vi disk.c
[exami@localhost ~]$ cc disk.c
[exami@localhost ~]$ ./a.out

Enter the maximum number of cylinders : 200
enter number of queue elements5

Enter the work queue23
89
132
42
187

Enter the disk head starting posision:
100

      MENU

      1. FCFS

      2. SSTF

      3. SCAN

      4. EXIT

Enter your choice: 1

FCFS

*****

Total seek time :421

      MENU

      1. FCFS

```

**VIVA QUESTIONS:**

1. What is Disk Scheduling?
2. What is FCFS Disk Scheduling?
3. What is SSTF Disk Scheduling?
4. Explain SCAN, C-SCAN Scheduling.
5. What is LOOK Scheduling?