KISHKINDA UNIVERSITY

ಕಿಷ್ಕಿಂದ ವಿಶ್ವವಿದ್ಯಾಲಯ

(AStatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

AdvancingKnowledgeTransformingLive

Department of Computer Science & Engineering.

KISHKINDA UNIVERSITY

ಕಿಷ್ಕಿಂದ ವಿಶ್ವವಿದ್ಯಾಲಯ

(AStatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

AdvancingKnowledgeTransformingLive

LABORATORY MANUAL
OPERATING SYSTEMS LAB

BE III Semester: 2024-25

KISHKINDA UNIVERSITY

(AStatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

Faculty of Engineering & Technology

Vision of the University

To develop an ethical and a competent global workforce with an inquisitive mind.

Mission of the University

- To equip the students with professional skills, ethical values and competency.
- To offer state of the art programs by collaborating with industry, academia and society.
- To undertake collaborative & inter disciplinary research for overall development.
- To develop leaders and entrepreneurs for the real and sustainable world

KISHKINDA UNIVERSITY

(AStatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

Faculty of Engineering & Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABORATORY MANUAL
OPERATING SYSTEMS LAB

**Prepared
By
J Anitha jyothi,
Assistant Professor
CSE Dept.**

KISHKINDA UNIVERSITY

(AStatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

Faculty of Engineering & Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision of the Department

To be a leading center of excellence in computer science education and research, dedicated to producing innovative, ethical, managerial and inquisitive professionals capable of addressing global challenges.

Mission of the department

Mission 1: Offer a cutting-edge curriculum that equips students with the knowledge and skills to excel in the rapidly evolving field of computer science.

Mission 2: Foster a vibrant research environment that encourages collaboration and breakthroughs in technology, driving advancements that address global challenges.

Mission 3: Equip students with a strong ethical and managerial foundation, ensuring they become responsible technologists dedicated to societal and environmental well-being.

KISHKINDA UNIVERSITY

(AStatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

Faculty of Engineering & Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM EDUCATIONAL OBJECTIVES

After 3-5 years of graduation, the graduates will be able to

PEO1: Graduates will create innovative solutions to global challenges using their computer science expertise.

PEO2: Graduates will lead with integrity, making decisions that benefit society and the environment.

PEO3: Graduates will pursue ongoing learning and research to stay at the forefront of technological advancements.

KISHKINDA UNIVERSITY

(AStatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

Faculty of Engineering & Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

Engineering graduates will be able to:

PO1: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: 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.

PO4: 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.

PO5: 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.

PO6: 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.

PO7: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communicate effectively on complex engineering activities with the engineering community and with the 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.

PO11: Demonstrate knowledge 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.

PO12: 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.

Program Specific Outcomes

At the end of 4 years, Computer Science and Engineering graduates will be able to:

PSO1: Develop proficiency in applying programming principles, algorithms, and modern software engineering practices to design, develop, and deploy robust software solutions across various platforms.

PSO2: Cultivate skills in employing data analysis techniques, machine learning models, and AI tools to extract insights, drive innovation, and support decision-making in real-world applications.

PSO3: Acquire proficiency to design, implement, and manage complex computer-based systems, ensuring seamless integration of hardware and software components to meet the specified needs.

Course Code	Course Title						Core/Elective
PC532CS	**OPERATING SYSTEMS LAB**						**CORE**
Prerequisite	Contact Hours Per Week				CIE	SEE	Credits
	L	T	D	P			
–	–	–	–	2	25	50	2

Course Objectives:

- To learn shell programming and the use of filters in the LINUX environment.
- To practice multithreaded programming.
- To implement CPU Scheduling Algorithms and memory management algorithms.

Course Outcomes:

- Evaluate the performance of different types of CPU scheduling algorithms..
- Simulate Banker's algorithm for deadlock avoidance.
- Implement paging replacement and disk scheduling techniques.
- Use different system calls for writing application programs.

I. CASESTUDY

Perform a case study by installing and exploring various types of operating systems on a physical or logical (virtual) machine.

II. List of Experiments (preferred programming language is C)

1. Write a C programs to implement UNIX system calls and file management.
2. Design, develop and implement program to simulate the working of Shortest Remaining Time First Scheduling algorithm. Experiment with different length jobs.
3. Design, develop and implement program to simulate the working of Round Robin (RR) Scheduling algorithms. Experiment with different quantum sizes for RR algorithm.
4. Design, develop and implement a Banker's algorithm. Assume suitable input required to demonstrate the results.
5. Design, develop and implement page replacement using FIFO algorithms. Assume suitable input required to demonstrate the results.
6. Design, develop and implement page replacement using LRU algorithms. Assume suitable input required to demonstrate the results.
7. Design, develop and implement optimal page replacement algorithms. Assume suitable input required to demonstrate the results.

KISHKINDA UNIVERSITY

(A StatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

Faculty of Engineering & Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SUBJECT NAME: OPERATING SYSTEMS LAB CODE:23CS33

SEMESTER: V

CONo.	Course Outcome	Taxonomy Level
23CS33.1	Evaluate the performance of different types of CPU scheduling algorithms.	Evaluating
23CS33.2	Simulate Banker's algorithm for deadlock avoidance.	Creating
23CS33.3	Implement paging replacement and disk scheduling techniques.	Applying
23CS33.4	Use different system calls for writing application programs.	Applying

PROGRAM I(CASE STUDY): Perform a case study by installing and exploring various types of operating systems on a physical or logical (virtual) machine. (Linux Installation). Instructions to Install Ubuntu Linux 12.04 (LTS) along with Windows

Back Up Your Existing Data!

This is highly recommended that you should take backup of your entire data before start with the installation process.

Obtaining System Installation Media

Download latest Desktop version of Ubuntu from this link:

http://www.ubuntu.com/download/desktop

Booting the Installation System

There are several ways to boot the installation system. Some of the very popular ways are , Booting from a CD ROM, Booting from a USB memory stick, and Booting from TFTP.

Here we will learn how to boot installation system using a CD ROM.

Before booting the installation system, one need to change the boot order and set CD-ROM as first boot device.

Changing the Boot Order of a Computers

1. As your computer starts, press the DEL, ESC, F1, F2, F8 or F10 during the initial startup screen. Depending on the BIOS manufacturer, a menu may appear. However, consult the hardware documentation for the exact key strokes. In my machine, its DEL key as shown in following screen-shot.
2. Find the Boot option in the setup utility. Its location depends on your BIOS. Select the Boot option from the menu, you can now see the options Hard Drive, CD-ROM Drive, Removable Devices Disk etc.
3. Change the boot sequence setting so that the CD-ROM is first. See the list of “Item Specific Help” in right side of the window and find keys which is used to toggle to change the boot sequence.
4. Insert the Ubuntu Disk in CD/DVD drive.

5. Save your changes. Instructions on the screen tell you how to save the changes on your computer. The computer will restart with the changed settings.

- Machine should boot from CD ROM, Wait for the CD to load...
- In a few minutes installation wizard will be started. Select your language and click the "Install Ubuntu" button to continue...
- Optionally, you can choose to download updates while installing and/or install third party software, such as MP3 support. Be aware, though, that if you select those options, the entire installation process will be longer!
- Since we are going to create partitions manually, select **Something else**, then click **Continue**. Keep in mind that even if you do not want to create partitions manually, it is better to select the same option as indicated here. This would insure that the installer will not overwrite your Windows , which will destroy your data. The assumption here is that **sdb** will be used just for Ubuntu 12.04, and that there are no valuable data on it.
- **Where are you?** Select your location and Click the "Continue" button.
- Select your keyboard layout and UK (English) and Click on “Continue” button.
- **Who are you?**
  Fill in the fields with your real name, the name of the computer (automatically generated, but can be overwritten), username, and the password.

Also at this step, there's an option called "Log in automatically." If you check it, you will automatically be logged in to the Ubuntu desktop without giving the password. Option "Encrypt my home folder," will encrypt your home folder. Click on the "Continue" button to continue...

- Now Ubuntu 12.04 LTS (Precise Pangolin) operating system will be installed.
  It will take approximately 10-12 minutes (depending on computer's speed), a pop-up window will appear, notifying you that the installation is complete, and you'll need to restart the computer in order to use the newly installed Ubuntu operating system. Click the "Restart Now" button.
- Please remove the CD and press the "Enter" key to reboot. The computer will be restarted. In a few seconds, you should see Windows 7′s boot menu with two entires listed – Windows 7 and Ubuntu 12.04 (LTS).
  Then you may choose to boot into Windows 7 or Ubuntu 12.04 using the UP/Down arrow key.
  Please select Ubuntu 12.04 (LTS) and press Enter to boot the machine in Ubuntu 12.04 Linux.
- Here you can see the users on the machine, Click on the user name and enter the password and press Enter key to login.
- We have successfully install and login to Ubuntu 12.04 LTS.

PROGRAM II : Write the program to implement fork () system call.

DESCRIPTION:
Used to create new processes. The new process consists of a copy of the address space of the original process. The value of process id for the child process is zero, whereas the value of process id for the parent is an integer value greater than zero.

Syntax:
fork ( );

ALGORITHM:

```
Step 1: Start the program.
Step 2: Declare the variables pid and child id.
Step 3: Get the child id value using system call fork().
Step 4: If child id value is greater than zero then print as “i am in the parent process”.
Step 5: If child id! = 0 then using getpid() system call get the process id.
Step 6: Print “i am in the parent process” and print the process id.
Step 7: If child id! = 0 then using getppid() system call get the parent process id.
Step 8: Print “i am in the parent process” and print the parent process id.
Step 9: Else If child id value is less than zero then print as “i am in the child process”.
Step 10: If child id! = 0 then using getpid() system call get the process id.
Step 11: Print “i am in the child process” and print the process id.
Step 12: If child id! = 0 then using getppid() system call get the parent process id.
Step 13: Print “i am in the child process” and print the parent process id.
Step 14: Stop the program.
```

PROGRAM :
SOURCE CODE:

```
/* fork system call */

#include<stdio.h>
#include <unistd.h>
#include<sys/types.h>
int main()
{
int id,childid;
id=getpid();
if((childid=fork())>0)
{
printf("\n i am in the parent process %d",id);
printf("\n i am in the parent process %d",getpid());
printf("\n i am in the parent process %d\n",getppid());
}
else
{
printf("\n i am in child process %d",id);
printf("\n i am in the child process %d",getpid());
printf("\n i am in the child process %d",getppid());
}
}
```

Program III (A): Write a program to implement the system calls wait ( ) and exit ( ).

DESCRIPTION:

1. **Wait():**

The parent waits for the child process to complete using the wait system call. The wait system call returns the process identifier of a terminated child, so that the parent can tell which of its possibly many children has terminated.

Syntax:

    wait (NULL);

2. **Exit ( ):**

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call. At that point, the process may return data (output) to its parent process (via the wait system call).

Syntax:

    exit (0);

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the variables pid and i as integers.
Step 3: Get the child id value using the system call fork ().
Step 4: If child id value is less than zero then print “fork failed”.
Step 5: Else if child id value is equal to zero, it is the id value of the child and then start the child process to execute and perform Steps 7 & 8.
Step 6: Else perform Step 9.
Step 7: Use a for loop for almost five child processes to be called.
Step 8: After execution of the for loop then print “child process ends”.
Step 9: Execute the system call wait ( ) to make the parent to wait for the child process to get over.
Step 10: Once the child processes are terminated, the parent terminates and hence prints “Parent process ends”.
Step 11: After both the parent and the child processes get terminated it execute the wait ( ) system call to permanently get deleted from the OS.
Step 12: Stop the program.

PROGRAM:
SOURCE CODE:

```
#include<stdio.h>
#include<unistd.h>
int main( )
{
int i, pid;
pid=fork( );
if(pid== -1)
{
printf("fork failed");
exit(0);
}
else if(pid==0)
{
printf("\n Child process starts");
for(i=0; i<5; i++)
{
printf("\n Child process %d is called", i);
}
printf("\n Child process ends");
}
else
{
wait(0);
printf("\n Parent process ends");
}
exit(0);
}
```

Program III (B): Write a program to implement the system calls wait ( ) and exit ( ).

WAIT () AND EXIT () SYSTEM CALLS
PROGRAM:
SOURCE CODE:

```
/* wait system call */

#include <stdlib.h>
#include <errno.h>
#include<stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
main()
{
pid_t pid;
int rv;
switch(pid=fork())
{
case -1:
        perror("fork");
        exit(1);
case 0:
        printf("\n CHILD: This is the child process!\n");
        fflush(stdout);
        printf("\n CHILD: My PID is %d\n", getpid());
        printf("\n CHILD: My parent's PID is %d\n",getppid());
        printf("\n CHILD: Enter my exit status (make it small):\n ");
        printf("\n CHILD: I'm outta here!\n");
        scanf(" %d", &rv);
        exit(rv);
default:
        printf("\nPARENT: This is the parent process!\n");
        printf("\nPARENT: My PID is %d\n", getpid());
        fflush(stdout);
        wait(&rv);
        fflush(stdout);
        printf("\nPARENT: My child's PID is %d\n", pid);
        printf("\nPARENT: I'm now waiting for my child to exit()...\n");
        fflush(stdout);
        printf("\nPARENT: My child's exit status is: %d\n",WEXITSTATUS(rv));
        printf("\nPARENT: I'm outta here!\n");
}
return(0);
}
```

Program IV: Write the program to implement the system calls opendir ( ), readdir ( ), closedir ( ).

DESCRIPTION:
UNIX offers a number of system calls to handle a directory. The following are most commonly used system calls.

SYSTEM CALLS USED:
i. opendir ( )
Open a directory.

Syntax:
DIR * opendir (const char * dirname);

opendir () takes dirname as the path name and returns a pointer to a DIR structure. On error returns NULL.

ii. readdir ( )
Read a directory.

Syntax:
struct dirent * readdir (DIR *dp) ;

A directory maintains the inode number and filename for every file in its fold. This function returns a pointer to a dirent structure consisting of inode number and filename. ‘dirent’ structure is defined in <dirent.h> to provide at least two members – inode number and directory name.

```
struct dirent
{
ino_t d_ino; // directory inode number
char d_name[]; // directory name
}
```

iii. closedir ():
Close a directory.

Syntax:
int closedir (DIR * dp);

Closes a directory pointed by dp. It returns 0 on success and -1 on error.

ALGORITHM:
```
Step 1: Start the program.
Step 2: In the main function pass the arguments.
Step 3: Create structure as stat buff and the variables as integer.
Step 4: Open the directory.
Step 5: Read the contents of the directory (filenames).
Step 6: Display the contents of the directory.
Step 7: Close the directory.
Step 8: Stop the program.
```

PROGRAM:
SOURCE CODE:

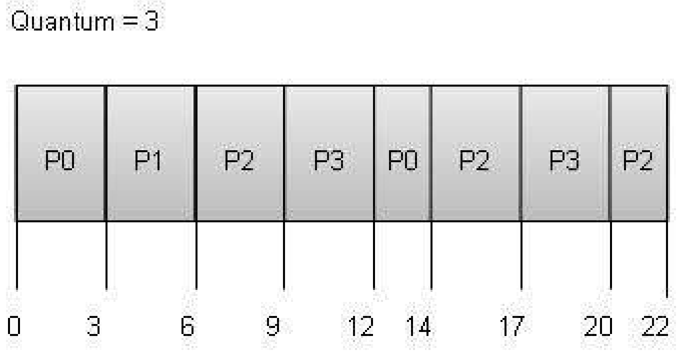
```
/* Recursively descend a directory hierarchy pointing a file */

#include<stdio.h>
#include<dirent.h>
#include<errno.h>
#include<fcntl.h>
#include<unistd.h>
int main(int argc,char *argv[])
{
struct dirent *direntp; DIR *dirp; if(argc!=2)
{
printf("ussage %s directory name \n",argv[0]);
return 1;
}
if((dirp=opendir(argv[1]))==NULL)
{
perror("Failed to open directory \n");
return 1;
}
while((direntp=readdir(dirp))!=NULL)
printf("%s\n",direntp->d_name);
while((closedir(dirp)==-1)&&(errno==EINTR));
return 0;
}
```

Program V: Design, develop and implement program to simulate the working of Round Robin (RR) scheduling algorithms. Experiment with different quantum sizes for RR algorithm.

DESCRIPTION:

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a quantum.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.



Wait time of each process is as follows –

Process	**Wait Time : Service Time - Arrival Time**
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

ALGORITHM:

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

Step 5: Calculate the average waiting time and average turnaround time
Step 6: Print the results of the step 4.
Step 7: Stop the program.

PROGRAM :
SOURCE CODE:

```
/* A program to simulate the Round Robin CPU scheduling algorithm */

#include<stdio.h>
struct process
{
int burst,wait,comp,f;
}p[20]={0,0};
int main()
{
int n,i,j,totalwait=0,totalturn=0,quantum,flag=1,time=0;
printf("\nEnter The No Of Process :");
scanf("%d",&n);
printf("\nEnter The Quantum time (in ms) :");
scanf("%d",&quantum);
for(i=0;i<n;i++)
{
printf("Enter The Burst Time (in ms) For Process #%2d :",i+1);
scanf("%d",&p[i].burst);
p[i].f=1;
}
printf("\nOrder Of Execution \n");
printf("\nProcess Starting Ending Remaining");
printf("\n\t\tTime \tTime \t Time");
while(flag==1)
{
flag=0;
for(i=0;i<n;i++)
{
if(p[i].f==1)
{
flag=1;
j=quantum;
if((p[i].burst-p[i].comp)>quantum)
{
p[i].comp+=quantum;
}
else
{
p[i].wait=time-p[i].comp;
j=p[i].burst-p[i].comp;
p[i].comp=p[i].burst;
p[i].f=0;
}
printf("\nprocess # %-3d %-10d %-10d %-10d", i+1, time, time+j, p[i].burst-p[i].comp);
time+=j;
}
}
}
```

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

Program VI: Design, develop and implement a Bankers's algorithm. Assume suitable input required to demonstrate the results.
AIM: To write a C program to implement bankers algorithm for dead lock avoidance.

DESCRIPTION:
In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process. Banker's algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.

ALGORITHM:
Step 1: Start the Program.
Step 2: Obtain the required data through char and int datatypes.
Step 3: Enter the filename, index block.
Step 4: Print the file name index loop.
Step 5: File is allocated to the unused index blocks.
Step 6: This is allocated to the unused linked allocation.
Step 7: Stop the execution.

PROGRAM:
SOURCE CODE:

```
/* Bankers algorithm for Deadlock Avoidance */

#include<stdio.h>
struct process
{
int allocation[3];
int max[3];
int need[3];
int finish;
}p[10];
int main()
{
int n,i,I,j,avail[3],work[3],flag,count=0,sequence[10],k=0;
printf("\nEnter the number of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter the %dth process allocated resources:",i);
scanf("%d%d%d",&p[i].allocation[0],&p[i].allocation[1],&p[i].allocation[2]);
printf("\nEnter the %dth process maximum resources:",i);
scanf("%d%d%d",&p[i].max[0],&p[i].max[1],&p[i].max[2]);

p[i].finish=0;
p[i].need[0]=p[i].max[0]-p[i].allocation[0];
p[i].need[1]=p[i].max[1]-p[i].allocation[1];
p[i].need[2]=p[i].max[2]-p[i].allocation[2];
}
printf("\nEnter the available vector:");
scanf("%d%d%d",&avail[0],&avail[1],&avail[2]);
for(i=0;i<3;i++)
work[i]=avail[i];
while(count!=n)
{
count=0;
for(i=0;i<n;i++)
{
flag=1;
if(p[i].finish==0)
if(p[i].need[0]<=work[0])
if(p[i].need[1]<=work[1])
if(p[i].need[2]<=work[2])
{
for(j=0;j<3;j++)
work[j]+=p[i].allocation[j];
p[i].finish=1;
sequence[k++]=i;
flag=0;
}
if(flag==1)
count++;
}
}
count=0;
for(i=0;i<n;i++)
if(p[i].finish==1)
count++;
printf("\n The safe sequence is:\t");
if(count++==n)
for(i=0;i<k;i++)
printf("%d\n",sequence[i]);
else
printf("SYSTEM IS NOT IN A SAFE STATE \n\n");
return 0;
}
```

Program VII: Design, develop and implement page replacement using FIFO algorithm. Assume suitable input required to demonstrate the results.

Page Replacement Algorithms:

Page replacement is basic to demand paging. It completes the separation between logical memory and physical memory. With this mechanism, an enormous virtual memory can be provided for programmers on a smaller physical memory. There are many different page-replacement algorithms. Every operating system probably has its own replacement scheme. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. If the recent past is used as an approximation of the near future, then the page that has not been used for the longest period of time can be replaced. This approach is the Least Recently Used (LRU) algorithm. LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time. Least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count.

ALGORITHM:

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

PROGRAM:

SOURCE CODE:

```
/* A program to simulate FIFO Page Replacement Algorithm */

#include<stdio.h>
main()
{
int a[5],b[20],n,p=0,q=0,m=0,h,k,i,q1=1;
char f='F';
printf("Enter the Number of Pages:");
scanf("%d",&n);
printf("Enter %d Page Numbers:",n);
for(i=0;i<n;i++)
scanf("%d",&b[i]);
for(i=0;i<n;i++)
{
if(p==0)
{
if(q>=3)
q=0;
a[q]=b[i];
q++;

if(q1<3)
{
q1=q;
}
}
printf("\n%d",b[i]);
printf("\t");
for(h=0;h<q1;h++)
printf("%d",a[h]);
if((p==0)&&(q<=3))
{
printf("-->%c",f);
m++;
}
p=0;
for(k=0;k<q1;k++)
{
if(b[i+1]==a[k])
p=1;
}
}
printf("\nNo of faults:%d",m);
}
```

Program VIII: Design, develop and implement page replacement using LRU algorithms. Assume suitable input required to demonstrate the results.

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

PROGRAM :
SOURCE CODE:

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

```
p=0;
if(q1==3)
{
for(k=0;k<q1;k++)
{
if(b[i+1]==a[k])
p=1;
}
for(j=0;j<q1;j++)
{
u=0;
k=i;
while(k>=(i-1)&&(k>=0))
{
if(b[k]==a[j])
u++;
k--;
}
if(u==0)
q=j;
}
}
else
{
for(k=0;k<q;k++)
{
if(b[i+1]==a[k])
p=1;
}
}
}
printf("\nNo of faults:%d",m);
}
```

Program IX: Design, develop and implement optimal page replacement algorithms. Assume suitable input required to demonstrate the results.