**Raj Kumar Goel Institute of Technology, Ghaziabad**

# LABORATORY MANUAL

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| **Faculty Name** | : | Mr. Sanjay Srivastava | **Department** | : | CSE |
| **Course Name** | : | Operating System | **Course Code** | : | KCS-451 |
| **Year/Sem** | : | 2nd/4th | **NBA Code** | : | C216 |
| **Email ID** | : | sansrfcs@rkgit.edu.in | **Academic Year** | : | 2019-20 |

**Department of Computer Science and Engineering**

## VISION OF THE INSTITUTE

To continually develop excellent professionals capable of providing sustainable solutions to challenging problems in their fields and prove responsible global citizens.

## MISSION OF THE INSTITUTE

We wish to serve the nation by becoming a reputed deemed university for providing value based professional education.

## VISION OF THE DEPARTMENT

To be recognized globally for delivering high quality education in the ever changing field of computer science & engineering, both of value & relevance to the communities we serve.

## MISSION OF THE DEPARTMENT

1. To provide quality education in both the theoretical and applied foundations of Computer Science and train students to effectively apply this education to solve real world problems.
2. To amplify their potential for lifelong high quality careers and give them a competitive advantage in the challenging global work environment.

## PROGRAM EDUCATIONAL OUTCOMES (PEOs)

**PEO 1:** **Learning:** Our graduates to be competent with sound knowledge in field of Computer Science & Engineering.

**PEO 2:** **Employable:** To develop the ability among students to synthesize data and technical concepts for application to software product design for successful careers that meet the needs of Indian and multinational companies.

**PEO 3:** **Innovative:** To develop research oriented analytical ability among students to prepare them for making technical contribution to the society.

**PEO 4:** **Entrepreneur / Contribution:** To develop excellent leadership quality among students which they can use at different levels according to their experience and contribute for progress and development in the society.

## PROGRAM OUTCOMES (POs)

**Engineering Graduates will be able to:**

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

**PO2: 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.

**PO3: 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.

**PO4: 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.

**PO5: Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

**PO6: 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.

**PO7: 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.

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

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

**PO10: 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.

**PO11: Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**PO12: Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

## PROGRAM SPECIFIC OUTCOMES (PSOs)

**PSO1:** The ability to use standard practices and suitable programming environment to develop software solutions.

**PSO2:** The ability to employ latest computer languages and platforms in creating innovative career opportunities

## COURSE OUTCOMES (COs)

| | |
|---|---|
| **C216.1** | Understand and implement basic services and functionalities of various operating system using system calls. |
| **C216.2** | Apply the concept of preemptive CPU scheduling algorithms |
| **C216.3** | Implement Banker's algorithm for Deadlock Avoidance |
| **C216.4** | Implement first fit, best fit and worst fit memory allocation schemes. |
| **C216.5** | Compare the performance of various page replacement algorithm |

## CO-PO MAPPING

| | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **C216.1** | 2 | 2 | 1 | 1 | | 1 | | | | | 1 | 1 |
| **C216.2** | 2 | 1 | 1 | 1 | | 1 | | | | | 1 | 1 |
| **C216.3** | 2 | 2 | 1 | 1 | | | | | | | | 1 |
| **C216.4** | 2 | 1 | 1 | 1 | | | | | | | | 1 |
| **C216.5** | 2 | 1 | 1 | 1 | | | | | | | | 1 |
| **C216** | 2 | 1.4 | 1 | 1 | | 1 | | | | | 1 | 1 |

## CO-PSO MAPPING

| | **PSO1** | **PSO2** |
|---|---|---|
| **C216.1** | 2 | |
| **C216.2** | 2 | |
| **C216.3** | 1 | |
| **C216.4** | 1 | 1 |
| **C216.5** | 1 | 1 |
| **C216** | 1.4 | 1 |

# LIST OF EXPERIMENTS

| Expt. No. | Title of experiment | Correspondin g |
|---|---|---|
| 1. | Study of hardware and software requirements of different operating systems (UNIX, LINUX, WINDOWS XP, WINDOWS 7/8) | C216.1 |
| 2. | To Execute UNIX system call for process management. | C216.1 |
| 3. | To Execute UNIX system call for File management. | C216.1 |
| 4. | To Execute UNIX system call for Input Output System Calls. | C216.1 |
| 5. | Simulation of SJF scheduling algorithm. | C216.2 |
| 6. | Simulation of Priority scheduling algorithm. | C216.2 |
| 7. | Simulation of FCFS scheduling algorithm. | C216.2 |
| 8. | Simulation of Multi-Level Queue scheduling algorithm. | C216.2 |
| 9. | Implement file storage allocation technique: Contiguous(using array). | C216.4 |
| 10. | Implement file storage allocation technique: Linked–list(using linked-list). | C216.4 |
| 11. | Implement file storage allocation technique: Indirect allocation (indexing). | C216.4 |
| 12 | Implementation of worst fit allocation technique. | C216.4 |
| 13. | Implementation of Best-fit allocation technique. | C216.4 |
| 14. | Implementation of First-fit allocation technique. | C216.4 |
| 15. | Calculation of external and internal fragmentation, free space list of blocks from system, List process file from the system. | C216.1 |
| 16. | Implementation of Compaction for the continually changing memory layout and calculate | C216.1 |
| 17. | Implementation of resource allocation graph (RAG). | C216.1 |
| 18. | Implementation of Banker‟s algorithm. | C216.3 |
| 19. | Implement the solution for Bounded Buffer (producer-consumer) problem using inter process communication. | C216.3 |
| 20. | Implement the solutions for Readers-Writers problem using inter-process communication technique -Semaphore. | C216.3 |

| | | |
|---|---|---|
| **21.** | Simulate FIFO page replacement algorithms. | C216.5 |
| **22.** | Simulate LRU page replacement algorithms | C216.5 |
| **23.** | Simulate Optimal page replacement algorithms | C216.5 |

# INTRODUCTION

An operating system acts as an intermediary between the user of a computer and computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.

An operating system is software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.

- An operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.

Every computer must have an operating system to run other programs. The operating system coordinates the use of the hardware among the various system programs and application programs for a various users. It simply provides an environment within which other programs can do useful work.

The operating system is a set of special programs that run on a computer system that allows it to work properly. It performs basic tasks such as recognizing input from the keyboard, keeping track of files and directories on the disk, sending output to the display screen and controlling peripheral devices.

OS is designed to serve two basic purposes:

1. It controls the allocation and use of the computing System's resources among the various user and tasks.
2. It provides an interface between the computer hardware and the programmer that simplifies and makes feasible for coding, creation, debugging of application programs.

The Operating system must support the following tasks. The task are:

1. Provides the facilities to create, modification of programs and data files using an editor.
2. Access to the compiler for translating the user program from high level language to machine language.
3. Provide a loader program to move the compiled program code to the computer's memory for execution.
4. Provide routines that handle the details of I/O programming.

**Types of Operating System** –

- Batch Operating System- Sequence of jobs in a program on a computer without manual interventions.
- Time sharing operating System- allows many users to share the computer resources.(Max utilization of the resources).
- Distributed operating System- Manages a group of different computers and make appear to be a single computer.
- Network operating system- computers running in different operating system can participate in common network (It is used for security purpose).
- Real time operating system – meant applications to fix the deadlines.

Examples of Operating System are –

- Windows (GUI based, PC)
- GNU/Linux (Personal, Workstations, ISP, File and print server, Three-tier client/Server)
- macOS (Macintosh), used for Apple's personal computers and work stations (MacBook, iMac).
- Android (Google's Operating System for smartphones/tablets/smartwatches)
- iOS (Apple's OS for iPhone, iPad and iPod Touch)

# PREFACE

In order to develop efficient software systems, it is essential that efficient algorithms and appropriate data structures are used. The purpose of this laboratory manual is to introduce undergraduate students to techniques for developing efficient data structures and algorithms in a systematic manner. The manual serves as a guide for learning and implementing the CPU scheduling algorithms and page replacement algorithms in C programming language. It basically focuses on memory management and various other operations on data with algorithm analysis and design. The manual contains procedures, and pre-experiment questions to help students prepare for experiments.

This practical manual will be helpful for students of Computer Science & Engineering for understanding the course from the point of view of applied aspects. Though all the efforts have been made to make this manual error free, yet some errors might have crept in inadvertently. Suggestions from the readers for the improvement of the manual are most welcomed.

# DO’S AND DONT'S

## DO's

1. Conform to the academic discipline of the department.
2. Enter your credentials in the laboratory attendance register.
3. Read and understand how to carry out an activity thoroughly before coming to the laboratory.
4. Ensure the uniqueness with respect to the methodology adopted for carrying out the experiments.
5. Shut down the machine once you are done using it.

## DONT'S

1. Eatables are not allowed in the laboratory.
2. Usage of mobile phones is strictly prohibited.
3. Do not open the system unit casing.
4. Do not remove anything from the computer laboratory without permission.
5. Do not touch, connect or disconnect any plug or cable without your faculty/laboratory technician’s permission.

## GENERAL SAFETY INSTRUCTIONS

1. Know the location of the fire extinguisher and the first aid box and how to use them in case of an emergency.
2. Report fire or accidents to your faculty /laboratory technician immediately.
3. Report any broken plugs or exposed electrical wires to your faculty/laboratory technician immediately.
4. Do not plug in external devices without scanning them for computer viruses.

# DETAILS OF THE EXPERIMENTS CONDUCTED

**(TO BE USED BY THE STUDENTS IN THEIR RECORDS)**

| S. No | DATE OF CONDUCTION | EXP. No | TITLE OF THE EXPERIMENT | PAGE No. | MARKS AWARDED (20) | FACULTY SIGNATURE WITH REMARK |
|---|---|---|---|---|---|---|
| 1 | | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |
| 6 | | | | | | |
| 7 | | | | | | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 12 | | | | | | |
| 13 | | | | | | |
| 14 | | | | | | |
| 15 | | | | | | |
| 16 | | | | | | |
| 17 | | | | | | |
| 18 | | | | | | |
| 19 | | | | | | |
| 20 | | | | | | |

# GUIDELINES FOR LABORTORY RECORD PREPARATION

While preparing the lab records, the student is required to adhere to the following guidelines:

## Contents to be included in Lab Records:

1. Cover page
2. Vision
3. Mission
4. PEOs
5. POs
6. PSOs
7. COs
8. CO-PO-PSO mapping
9. Index
10. Experiments
    - Aim
    - Source code
    - Input-Output

A separate copy needs to be maintained for pre-lab written work.

The student is required to make the Lab File as per the format given on the next two pages.

# Raj Kumar Goel Institute of Technology, Ghaziabad

**OPERATING SYSTEM LAB FILE (KCS 451)**

| | |
|---|---|
| **Name** | |
| **Roll No.** | |
| **Section- Batch** | |

# INDEX

| Experiment No. | Experiment Name | Date of Conduction | Date of Submission | Faculty Signature |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

## GUIDELINES FOR ASSESSMENT

Students are provided with the details of the experiment (Aim, pre-experimental questions, procedure etc.) to be conducted in next lab and are expected to come prepared for each lab class.

Faculty ensures that students have completed the required pre-experiment questions and they complete the in-lab programming assignment(s) before the end of class. Given that the lab programs are meant to be formative in nature, students can ask faculty for help before and during the lab class.

Students' performance will be assessed in each lab based on the following Lab Assessment Components:

**Assessment Criteria-1:** Performance (Max. marks = 5)

**Assessment Criteria-2:** VIVA (Max. marks = 5)

**Assessment Criteria-3:** Record (Max. marks = 5)

In each lab class, students will be awarded marks out of 5 under each component head, making it total out of 15 mark

## EXPERIMENT - 1

**Aim:** Study of hardware and software requirements of different operating systems (UNIX, LINUX, WINDOWS XP, WINDOWS 7/8

**Description:** The OBJECTIVE of this practical is to obtain general overview of various popular OS

(a) Their development and distribution
(b) Compatibility
(c) Security issues and Thread detection and its solution
(d) Commands

### SYSTEM REQUIREMENT FOR UNIX

Unix was first developed in 1969 at Bell Labs (Thompson & Ritchie) by Ritchie & Thompson. They identified the (new) features of Unix: A hierarchical file system incorporating demountable volumes.

Compatible file, device and inter-process IO (naming schemes, access control)

Ability to initiate asynchronous processes (i.e., address-spaces = heavyweight)System command language selectable on a per-user basis

Completely novel at the time: prior to this, everything was "inside" the OS. In Unix separation between essential things (kernel) and everything else

Among other things: allows user wider choice without increasing size of core OS allows easy replacement of functionality — resulted in over 100 subsystems including a dozen languages Highly portable due to use of high-level language Features which were not included: real time, multiprocessor support

### SYSTEM REQUIREMENT FOR LINUX

Linux systems include user interfaces and applications in addition to the kernel

• Borrows from the UNIX layered system approach

• System contains kernel threads to perform services– Implemented as daemons, which sleep until awakened by a kernel component

• Multiuser system– Restricts access to important operations to users with superuser (also called root) privileges

**User Interface**

Can be accessed via the command-line via shells such as bash, csh and esh

• Most Linux GUIs are layered– X Window System

• Lowest level

• Provides to higher GUI layers mechanisms to create and manipulate graphical components– Window manager

• Builds on mechanisms in the X Window System interface to control the placement, appearance, size and other window attributes– Desktop environment (e.g., KDE, GNOME)

• Provide user applications and service

Linux increasingly conforms to popular standards such as POSIX

• The Single UNIX Specification (SUS)– Suite of standards that define user and application programming interfaces for UNIX operating systems, shells and utilities– Version 3 of the SUS combines several standards (including POSIX, ISO standards and previous versions of the SUS)

• Linux Standards Base (LSB)– Project that aims to standardize Linux so that applications written for one LSB-compliant distribution will compile and behave exactly the same on any other LSB-compliant distribution

**Hardware Platforms**

• Supports a large number of platforms, including – x86 (including Intel IA-32), HP/Compaq Alpha AXP, Sun SPARC, Sun UltraSPARC, Motorola 68000, PowerPC, PowerPC64, ARM, Hitachi SuperH, IBM S/390 and zSeries, MIPS, HP PA-RISC, Intel IA-64,AMD x86-64,H8/300,V850 and CRIS

• Architecture-specific code– Performs operations implemented differently across platforms

• Porting– Modifying the kernel to support a new platform

• Source tree– Loosely organizes kernel into separate components by directory

• User-mode Linux (UML)– Important tool for kernel development

Loadable Kernel Modules

• Loadable kernel modules– Contains object code that, when loaded, is dynamically linked to a running kernel– Enables code to be loaded on demand

• Reduces the kernel's memory footprint– Modules written for versions of the kernel other than the current one may not work properly–Kmod: a kernel subsystem that manages modules without user intervention

• Determines module dependencies and loads them on demand

**System Requirements: Windows 7, Windows 8/8.1, Windows 10**

All these operating system requires the following elements as minimum requirements.

1. **Processor:** 1 GHz or faster processor (32-bit or 64-bit). Processor must support PAE, NX and SSE2 to use Windows 8 and Windows 8.1
2. **RAM:** 1 GB for 32-bit and 2GB for 64-bit
3. **Free Hard Disk Space**: Windows 10 requires 16GB free hard disk space for installation. Windows 7 and Windows 8/8.1 requires 16 GB for 32-bit and 20 GB for 64-bit operating system.
4. **Graphic Card:** Microsoft DirectX 9 graphics with WDDM driver.

| **Facility** | **Bit** | **Windows 10** | **Windows 8/ 8.1** | **Windows 7** |
|---|---|---|---|---|
| Processor | 32-bit | 1 GHz or Higher | 1 GHz or Higher | 1 GHz or Higher |
| | 64-bit | 1 GHz or Higher | 1 GHz or Higher | 1 GHz or Higher |
| RAM | 32-bit | 1 GB | 1 GB | 1 GB |
| | 64-bit | 2 GB | 2 GB | 2 GB |
| Free Hard Disk Space | 32-bit | 16 GB | 16 GB | 16 GB |
| | 64-bit | 20 GB | 20 GB | 16 GB |
| Graphics Card | | DirectX 9 Graphics With WDDM 1.0 or Higher Driver | DirectX 9 Graphics With WDDM 1.0 or Higher Driver | DirectX 9 Graphics With WDDM 1.0 or Higher Driver |

**COMMAND :**

**1.Date Command :**

This command is used to display the current data and time.

**Syntax:** $date

**2.Calender Command :**

This command is used to display the calendar of the year or the particular month of calendar year.

**Syntax :**

a. $cal<year>
b. $cal<month><year>

Here the first syntax gives the entire calendar for given year & the second Syntax gives the calendar of reserved month of that year.

**3.Echo Command :**

This command is used to print the arguments on the screen .

**Syntax :** $echo <text>

**Multi line echo command :**

To have the output in the same line , the following commands can be used.

**Syntax :** $echo <text\>text

To have the output in different line, the following command can be used.

**Syntax :** $echo “text

>line2

>line3”

**4.’who’ Command :**

It is used to display who are the users connected to our computer currently.

**Syntax :** $who – option‟s

**Options : -**

H–Display the output with headers.

b–Display the last booting date or time or when the system was lastely rebooted.

**5.’who am i’ Command :**

Display the details of the current working directory.

**Syntax :** $who am i

**6.’tty’ Command :**

It will display the terminal name.

**Syntax :** $tty

**7.’CLEAR’ Command :**

It is used to clear the screen.

**Syntax :** $clear

**8.’MAN’ Command :**

It help us to know about the particular command and its options & working. It is like "help" command in windows .

**Syntax :** $man <command name>

**9.MANIPULATION Command :**

It is used to manipulate the screen. **Syntax :**$tput<argument>

**Arguments :**

1.Clear – to clear the screen.

2.Longname – Display the complete name of the terminal.

3.SMSO – background become white and foreground become black color. 4.rmso – background become black and foreground becomes white color. 5.Cols – Display the number of columns in our terminals.

**10.LIST Command :**

It is used to list all the contents in the current working directory.

**Syntax :**$ls–options <arguments>

If the command does not contain any argument means it is working in the Current directory.

**Options :**

a– used to list all the files including the hidden files. c– list all the files columnwise.

d- list all the directories.

m- list the files separated by commas.

p- list files include „/‟ to all the directories. r- list the files in reverse alphabetical order.

f- list the files based on the list modification date. x-list in column wise sorted order.

**DIRECTORY RELATED COMMANDS :**

**1.Present Working Directory Command :**

To print the complete path of the current working directory.

**Syntax :** $pwd

**2.MKDIR Command :**

To create or make a new directory in a current directory .

**Syntax :**$mkdir<directory name>

**3.CD Command :**

To change or move the directory to mentioned directory .

**Syntax :**$cd <directory name>

**4.RMDIR Command :**

To remove a directory in the current directory & not the current directory itself

**Syntax :**$rmdir<directory name>

**FILE RELATED COMMANDS : 1.CREATE A FILE :**

To create a new file in the current directory we use CAT command.

**Syntax :**

$cat  > filename.

The > symbol is redirectory we use cat command.

**2.DISPLAY A FILE :**

To display the content of file mentioned we use CAT command without „>‟ operator.

**Syntax :**

$cat <filename.

Options –s = to neglect the warning /error message.

**3.COPYING CONTENTS :**

To copy the content of one file with another. If file doesnot exist, a new file is created and if the file exists with some data then it is overwritten.

**Syntax :**

$ cat<filename source>>><destination filename>

**Options : -**

-n content of file with numbers included with blank lines.

**Syntax :**

$cat –n <filename>

**4.SORTING A FILE :**

To sort the contents in alphabetical order or in reverse order.

**Syntax :**

$sort <filename >

**Option :** $ sort –r <filename>

**5.COPYING CONTENTS FROM ONE FILE TO ANOTHER :**

To copy the contents from source to destination file .so that both contents are same.

**Syntax :**

$cp<source filename><destination filename>

$cp<source filename path ><destination filename path>

**6.MOVE Command :**

To completely move the contents from source file to destination file and to remove the source file.

**Syntax :**

$ mv<source filename><destination filename>

**7.REMOVE Command :**

To permanently remove the file we use this command .

**Syntax :**

$rm<filename>

**8.WORD Command :**

To list the content count of no of lines , words, characters .

**Syntax :**

$wc<filename>

**Options :**

-c – to display no of characters.-l – to display only the lines.

-w – to display the no of words.

**9. LINE PRINTER :**

To print the line through the printer, we use lp command.

**Syntax :**

$lp<filename>

**10. PAGE Command :**

This command is used to display the contents of the file page wise & next page can be viewed by pressing the enter key.

**Syntax :**

$pg<filename>

**11. FILTERS AND PIPES**

**HEAD :**It is used to display the top ten lines of file. **Syntax:** $head<filename>

**TAIL :**This command is used to display the last ten lines of file.

**Syntax:** $tail<filename>

**PAGE :** This command shows the page by page a screen full of information is displayed after which the page command displays a prompt and passes for the user to strike the enter key to continue scrolling.

**Syntax:**

**MORE :**It also displays the file page by page .To continue scrolling with more command ,press the space bar key.

**Syntax:** $more<filename>

**GREP :**This command is used to search and print the specified patterns from the file.

**Syntax:** $grep [option] pattern <filename>

**SORT :**This command is used to sort the datas in some order.

**Syntax:** $sort<filename>

**PIPE :**It is a mechanism by which the output of one command can be channeled into the inputof another command.

$who | wc-l

**Syntax:**

**TR :**Thetr filter is used to translate one set of characters from the standard inputs to another.

**Syntax:** $tr "[a-z]" "[A-Z]"

**Post-Experiment Questions:**

1. What is the difference between UNIX and LINUX?
2. What is BASH?
3. What are the basic components of Linux?
4. What is Linux Kernel?

**Experiment 2**

**Aim:** To Execute UNIX system call for process management.

**Description:**

**Fork System Call:** Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

It takes no parameters and returns an integer value. Below are different values returned by fork().

**Negative Value:** creation of a child process was unsuccessful. **Zero:** Returned to the newly created child process. **Positive value:** Returned to parent or caller. The value contains process ID of newly created child process.

**Program:**

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main ()
{
int pid;
printf ("I'm the original process with Process id: %d and my parent id is: %d.\n",
getpid (), getppid ());
pid = fork (); /* Duplicate. Child and parent continue from here */
if (pid != 0) /* pid is non-zero, so I must be the parent */
{
printf ("I'm the parent process with process id %d and my parent id is %d.\n",
getpid (), getppid ());
printf ("My child's process id is %d\n", pid);
}
else /* pid is zero, so I must be the child */
{
printf ("I'm the child process with process id %d and my parent id is %d.\n",
getpid (), getppid ());
}
printf("Process with process id %d terminates.\n",getpid ());/*Both procs execute this */
}
```

**Output:**

```
I'm the original process with Process id: 3514 and my parent id is: 1923.
I'm the parent process with process id 3514 and my parent id is 1923.
My child's process id is 3515
Process with process id 3514 terminates.
I'm the child process with process id 3515 and my parent id is 3514.
Process with process id 3515 terminates.
```

**Post-Experiment Questions:**

1. Explain c program to implement the Unix or Linux command to implement ls -l >output.txt?
2. Explain the mount and unmount system calls?

## Experiment 3

**Aim:** To Execute UNIX system call for File management.

**Description:**

The file structure related system calls available in the UNIX system let you create, open, and close files, read and write files, randomly access files, alias and remove files, get information about files, check the accessibility of files, change protections, owner, and group of files, and control devices. These operations either use a character string that defines the absolute or relative path name of a file, or a small integer called a file descriptor that identifies the I/O channel.
There are four system calls for file management:

1. **open ()**
2. **read ()**
3. **write ()**
4. **close ()**

**open ()**

**open() system call** is used to know the file descriptor of user-created files. Since read and write use file descriptor as their 1st parameter so to know the file descriptor open() system call is used.

**Syntax:**

fd = open (file_name, mode, permission);

Example:

fd = open ("file", O_CREAT | O_RDWR, 0777);

**Here,**

- file_name is the name to the file to open.
- mode is used to define the file opening modes such as create, read, write modes.
- permission is used to define the file permissions.

**Return value:** Function returns the file descriptor.

**read ()**

**read() system call** is used to read the content from the file. It can also be used to read the input from the keyboard by specifying the **0** as file descriptor (see in the program given below).

**Syntax:**

length = read(file_descriptor , buffer, max_len);
Example:
n = read(0, buff, 50);

**Here,**

- file_descriptor is the file descriptor of the file.
- buffer is the name of the buffer where data is to be stored.

- max_len is the number specifying the maximum amount of that data can be read

**Return value:** If successful read returns the number of bytes actually read.

**write ()**

**write() system call** is used to write the content to the file.

**Syntax:**

```
length = write(file_descriptor , buffer, len);
Example:
n = write(fd, "Hello world!", 12);
```

**Here,**

- file descriptor is the file descriptor of the file.
- buffer is the name of the buffer to be stored.
- len is the length of the data to be written.

**Return value:** If successful write() returns the number of bytes actually written.

**close ()**

**close() system call** is used to close the opened file, it tells the operating system that you are done with the file and close the file.

**Syntax:**

```
int close(int fd);
```

**Here,**

- fd is the file descriptor of the file to be closed.

**Return value:** If file closed successfully it returns 0, else it returns -1.

**SOLUTION**

```
#include<unistd.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<stdio.h>
int main()
{
    int n,fd;
    char buff[50];  // declaring buffer
    //message printing on the display
    printf("Enter text to write in the file:\n");
    //read from keyboard, specifying 0 as fd for std input device
    //Here, n stores the number of characters
    n= read(0, buff, 50);
    // creating a new file using open.
    fd=open("file",O_CREAT | O_RDWR, 0777);
    //writting input data to file (fd)
    write(fd, buff, n);
    //Write to display (1 is standard fd for output device)
    write(1, buff, n);

    //closing the file
    int close(int fd);
    return 0;
}
```

**Output**

```
Enter text to write in the file:
Hello world, welcome @ IncludeHelp
Hello world, welcome @ IncludeHelp
```

**Post-Experiment Questions:**

1. What is system call and how it is handled by operating system?
2. What are 5 different categories of system calls?

**Experiment 4**

**Aim:** To Execute UNIX system call for Input Output System Calls.

**Description:**

**Algorithm:**

1. Start the program.

2. open a file for O_RDWR for R/W,O_CREATE 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.

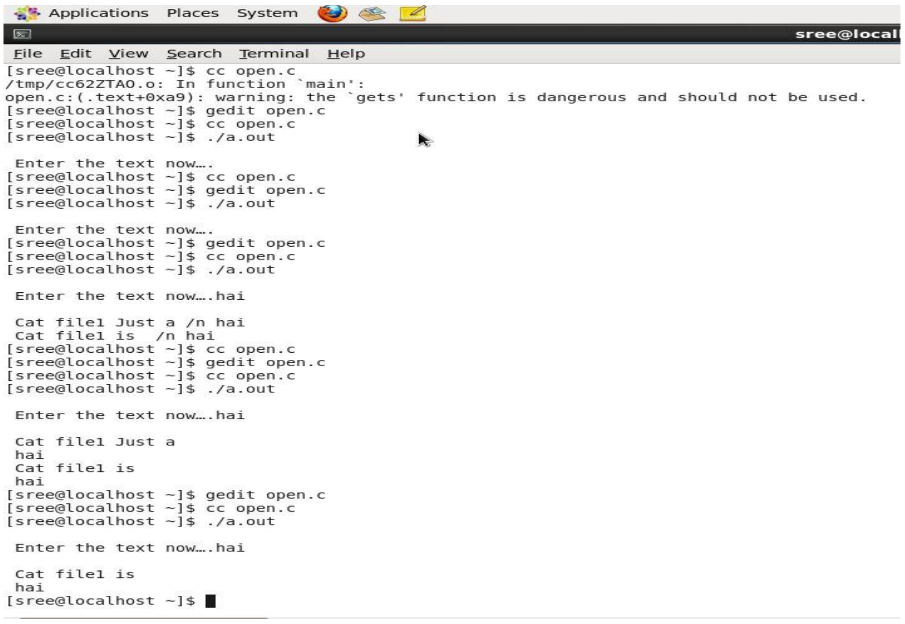
5. Then the first is opened for read only mode and read the characters and displayed It and close the file

6.Stop the program

**Program:**

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<fcntl.h>
main( )
{
int fd[2];
char buf1[25]= "just a test\n";
char buf2[50];
fd[0]=open("file1",O_RDWR);
fd[1]=open("file2",O_RDWR);
write(fd[0], buf1, strlen(buf1));
printf("\n Enter the text now….");
scanf("\n %s",buf1);
printf("\n Cat file1 is \n hai");
write(fd[0], buf1, strlen(buf1));
lseek(fd[0], SEEK_SET, 0);
read(fd[0], buf2, sizeof(buf1));
write(fd[1], buf2, sizeof(buf2));
close(fd[0]);
close(fd[1]);
printf("\n");
return 0;
}
```

**OUTPUT:**


Applications Places System
sree@local
File Edit View Search Terminal Help

```
[sree@localhost ~]$ cc open.c
/tmp/cc62ZTAO.o: In function `main':
open.c:(.text+0xa9): warning: the `gets' function is dangerous and should not be used.
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ ./a.out

 Enter the text now….
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ ./a.out

 Enter the text now….
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ ./a.out

 Enter the text now….hai

 Cat file1 Just a /n hai
 Cat file1 is  /n hai
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ ./a.out

 Enter the text now….hai

 Cat file1 Just a
 hai
 Cat file1 is
 hai
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ ./a.out

 Enter the text now….hai

 Cat file1 is
 hai
[sree@localhost ~]$ █
```

**Post-Experiment Questions:**

1. What is file descriptor?
2. Explain some Input/output system calls?

## Experiment 5

**Aim:** Simulation of SJF scheduling algorithm.

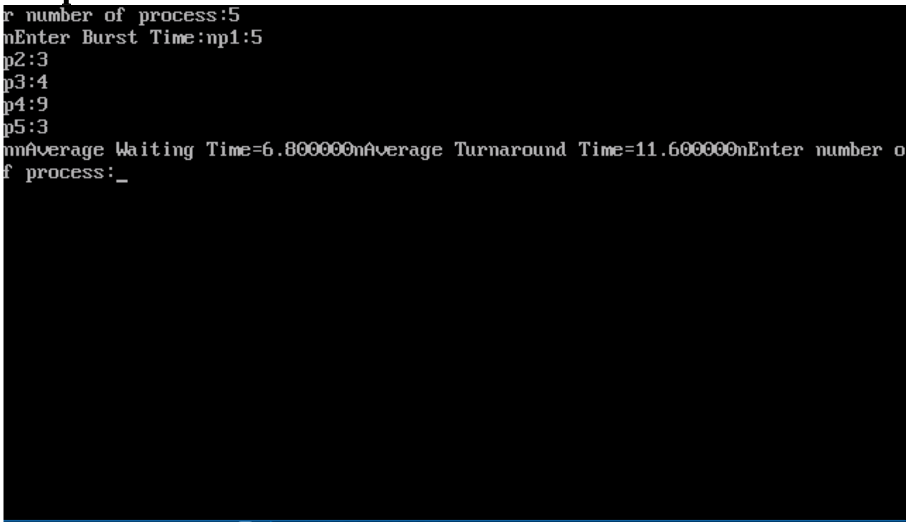
**Description:** Shortest Job First (SJF)

- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

**Program:**

```
#include<stdio.h>
int main()
{
int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
float avg_wt,avg_tat;
printf("Enter number of process:");
scanf("%d",&n);
printf("nEnter Burst Time:n");
for(i=0;i<n;i++)
{
printf("p%d:",i+1);
scanf("%d",&bt[i]);
p[i]=i+1;
}
//sorting of burst times
for(i=0;i<n;i++)
{
pos=i;
for(j=i+1;j<n;j++)
{
if(bt[j]<bt[pos])
pos=j;
}
temp=bt[i];
bt[i]=bt[pos];
bt[pos]=temp;
temp=p[i];
p[i]=p[pos];
p[pos]=temp;
}
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
total+=wt[i];
}
avg_wt=(float)total/n;
total=0;
printf("nProcesst Burst Time tWaitingTimetTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i];
total+=tat[i];
printf("np%dtt %dtt %dttt%d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=(float)total/n;
printf("nnAverage Waiting Time=%f",avg_wt);
printf("nAverage Turnaround Time=%fn",avg_tat);
}
```

**Output:**



```
r number of process:5
nEnter Burst Time:np1:5
p2:3
p3:4
p4:9
p5:3
nnAverage Waiting Time=6.800000nAverage Turnaround Time=11.600000nEnter number o
f process:_
```

**Post-Experiment Questions:**

1. What is scheduling?
2. Explain short, medium and long term scheduler?

## Experiment 6

**Aim:** Simulation of Priority scheduling algorithm.

**Description:** Priority scheduling is one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. Process with the highest priority is to be executed first and so on. Processes with the same priority are executed on first come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement.

**Program:**

```
void main()
{
int i,j,n,t,turn[20],burst[20],p[20],wt[20],c[20];
float await,aturn,twait=0,tturn=0;
printf("\nEnter the value of n:");
scanf("%d",&n);
printf("\n Enter the process no burst and arrivaltime");
for(i=0;i<n;i++)
{
scanf("%d",&c[i]);
scanf("%d",&burst[i]);
scanf("%d",&p[i]);
}
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
{
if(p[i]>p[j])
{
t=p[i];
p[i]=p[j];
p[j]=t;
t=burst[i];
burst[i]=burst[j];
burst[j]=t;
t=c[i];
c[i]=c[j];
c[j]=t;
}
}
for(i=0;i<n;i++)
{
if(i==0)
{
wt[i]=0;
turn[i]=burst[i];
}
}
else
{
turn[i]=turn[i-1]+burst[i];
wt[i]=turn[i]-burst[i];
twait=twait+wt[i];
tturn=tturn+turn[i];
}await=twait/n;
aturn=tturn/n;
printf("pno\tbtime\tatime\twtime\tttime");
for(i=0;i<n;i++)
{
printf("\n%d\t%d\t%d\t%d\t%d\n",c[i],burst[i],p[i],wt[i],turn[i]);
}
printf("\n The average waiting time is:%f",await);
printf("\n The average turn around time is:%f",aturn);
}
```

**Output:**

```
Enter the process burst no and priority
1 15 2
2 5 1
3 10 3
p no    btime           priority        wtime           ttime
2         5                1               0               5
1        15                2               5              20
3        10                3              20              30
The average waiting time is :8.333333
The average turn around time is:18.333334
```

**Post-Experiment Questions:**

1. Why CPU scheduling is necessary?
2. What is context switching?

## Experiment 7

**Aim:** Simulation of FCFS scheduling algorithm.

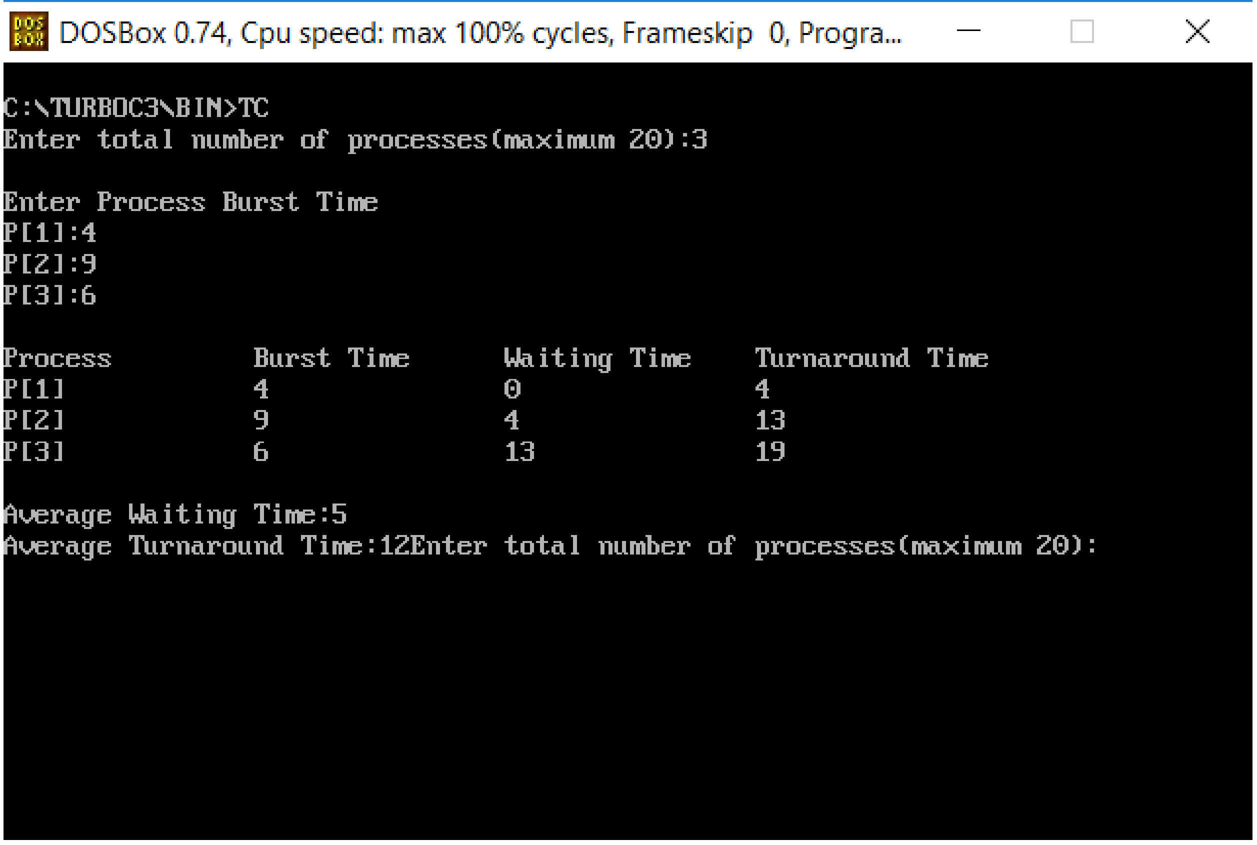
**Description:** First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

**Program:**

```
#include<stdio.h>
int main()
{
int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
printf("Enter total number of processes(maximum 20):");
scanf("%d",&n);
printf("\nEnter Process Burst Time\n");
for(i=0;i<n;i++)
{
printf("P[%d]:",i+1);
scanf("%d",&bt[i]);
}
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
}
printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i];
avwt+=wt[i];
avtat+=tat[i];
printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
}
avwt/=i;
avtat/=i;
printf("\n\nAverage Waiting Time:%d",avwt);
printf("\nAverage Turnaround Time:%d",avtat);
return 0;
}
```

**Output:**



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...

```
C:\TURBOC3\BIN>TC
Enter total number of processes(maximum 20):3

Enter Process Burst Time
P[1]:4
P[2]:9
P[3]:6

Process         Burst Time      Waiting Time    Turnaround Time
P[1]            4               0               4
P[2]            9               4               13
P[3]            6               13              19

Average Waiting Time:5
Average Turnaround Time:12Enter total number of processes(maximum 20):
```

**Post-Experiment Questions:**

1. Explain about FCFS scheduling algorithm?
2. What is average waiting time?

## Experiment 8

**Aim:** Simulation of Multi-Level Queue scheduling algorithm.

**Description:**

Multi-level queue scheduling algorithm is used in scenarios where the processes can be classified into groups based on property like process type, CPU time, IO access, memory size, etc. In a multi-level queue scheduling algorithm, there will be 'n' number of queues, where 'n' is the number of groups the processes are classified into. Each queue will be assigned a priority and will have its own scheduling algorithm like round-robin scheduling or FCFS. For the process in a queue to execute, all the queues of priority higher than it should be empty, meaning the process in those high priority queues should have completed its execution. In this scheduling algorithm, once assigned to a queue, the process will not move to any other queues.

**Program:**

```
main()
 { int p[20],bt[20], su[20], wt[20],tat[20],i, k, n, temp;
float wtavg, tatavg; clrscr();
printf("Enter the number of processes --- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{ p[i] = i;
printf("Enter the Burst Time of Process %d --- ", i);
scanf("%d",&bt[i]); printf("System/User Process (0/1) ? --- ");
scanf("%d", &su[i]);
}
for(i=0;i<n;i++) for(k=i+1;k<n;k++)
if(su[i] > su[k])
{ temp=p[i]; p[i]=p[k]; p[k]=temp;
temp=bt[i]; bt[i]=bt[k]; bt[k]=temp;
 temp=su[i]; su[i]=su[k]; su[k]=temp;
 }
wtavg = wt[0] = 0;
tatavg = tat[0] = bt[0];
for(i=1;i<n;i++)
{ wt[i] = wt[i-1] + bt[i-1];
 tat[i] = tat[i-1] + bt[i];
wtavg = wtavg + wt[i]; tatavg = tatavg + tat[i];
}
printf("\nPROCESS\t\t SYSTEM/USER PROCESS \tBURST TIME\tWAITING
TIME\tTURNAROUND TIME");
for(i=0;i<n;i++) printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],su[i],bt[i],wt[i],tat[i]);
 printf("\nAverage Waiting Time is --- %f",wtavg/n);
printf("\nAverage Turnaround Time is --- %f",tatavg/n);
 getch();
}
```

**OUTPUT**

```
Enter the number of processes --- 4
Enter the Burst Time of Process 0 --- 3 System/User Process (0/1) ? --- 1
Enter the Burst Time of Process 1 --- 2 System/User Process (0/1) ? --- 0
Enter the Burst Time of Process 2 --- 5 System/User Process (0/1) ? --- 1
Enter the Burst Time of Process 3 --- 1 System/User Process (0/1) ? --- 0
```

**RESULT**

```
 PROCESS SYSTEM/USER PROCESS BURST TIME WAITING TIME TURNAROUND
TIME 1 0 2 0 2 3 0 1 2 3 2 1 5 3 8 0 1 3 8 11
 Average Waiting Time is --- 3.250000
 Average Turnaround Time is --- 6.000000
```

**Post-Experiment Questions:**

1. Describe Multi-Level Queue scheduling algorithm?
2. Define turnaround time?

## Experiment 9

**Aim:** Implement file storage allocation technique: Contiguous (using array).

**Description:** A file is a collection of data, usually stored on disk. As a logical entity, a file enables to divide data into meaningful groups. As a physical entity, a file should be considered in terms of its organization. The term "file organization" refers to the way in which data is stored in a file and, consequently, the method(s) by which it can be accessed.

**Program:**

```
#include < stdio.h>
#include<conio.h>
void main()
{
int f[50], i, st, len, j, c, k, count = 0;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("Files Allocated are : \n");
x: count=0;
printf("Enter starting block and length of files: ");
scanf("%d%d", &st,&len);
for(k=st;k<(st+len);k++)
if(f[k]==0)
count++;
if(len==count)
{
for(j=st;j<(st+len);j++)
if(f[j]==0)
{
f[j]=1;
printf("%d\t%d\n",j,f[j]);
}
if(j!=(st+len-1))
printf(" The file is allocated to disk\n");
}
else
printf(" The file is not allocated \n");
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit();
getch();
}
```

**Output:**

```
Files Allocated are:
Enter starting block and length of files: 14 3
14 1
15 1
16 1

The file is allocated to disk

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

Enter starting block and length of files: 14 1

The file is not allocated

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

Enter starting block and length of files: 14 4

The file is not allocated

Do you want to enter more file(Yes - 1/No - 0)0
```

**Post-Experiment Questions:**

1. Describe contiguous file allocation scheme?
2. Write advantages & disadvantages of contiguous file allocation scheme?

## Experiment 10

**Aim:** Implement file storage allocation technique: Linked–list (using linked-list).

**Description:** This method solves all problems of contiguous allocation. In linked allocation scheme each file is a linked list of disk blocks scattered on the disk. The first word of each block is used as a pointer to the next one and the rest of block is used for data.

**Program:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int f[50], p,i, st, len, j, c, k, a;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks already allocated: ");
scanf("%d",&p);
printf("Enter blocks already allocated: ");
for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
x: printf("Enter index starting block and length: ");
scanf("%d%d", &st,&len);
k=len;
if(f[st]==0)
{
for(j=st;j<(st+k);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("%d-------->%d\n",j,f[j]);
}
else
{
printf("%d Block is already allocated \n",j);
k++;
}
}
}
else
printf("%d starting block is already allocated \n",st);
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
getch();
}
```

**Output:**

```
Enter how many blocks already allocated: 3

Enter blocks already allocated: 1 3 5

Enter index starting block and length: 2 2
2-------->1

3 Block is already allocated
4-------->1

Do you want to enter more file(Yes - 1/No - 0)0
```

**Post-Experiment Questions:**

1. Describe linked list allocation scheme?
2. Write advantages & disadvantages of linked list allocation scheme?

## Experiment-11

**Aim:** Implement file storage allocation technique: Indirect allocation (indexing).

**Description:** Indexed allocation method eliminates the disadvantages of linked list allocation by bringing all the pointers together into one location, called the index block.

**Program:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
x:printf("Enter the index block: ");
scanf("%d",&ind);
if(f[ind]!=1)
{
printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);
scanf("%d",&n);
}
else
{
printf("%d index is already allocated \n",ind);
goto x;
}
y: count=0;
for(i=0;i<n;i++)
{
scanf("%d", &index[i]);
if(f[index[i]]==0)
count++;
}
if(count==n)
{
for(j=0;j<n;j++)
f[index[j]]=1;
printf("Allocated\n");
printf("File Indexed\n");
for(k=0;k<n;k++)
printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);
}
else
{
```

```
printf("File in the index is already allocated \n");
printf("Enter another file indexed");
goto y;
}
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
getch();
}
```

**Output:**

```
Enter the index block: 5
Enter no of blocks needed and no of files for the index 5 on the disk :  4
1 2 3 4
Allocated  File Indexed
5-------->1 : 1
5-------->2 : 1
5-------->3 : 1
5-------->4 : 1
Do you want to enter more file(Yes - 1/No - 0)1
Enter the index block: 4
4 index is already allocated
Enter the index block: 6
Enter no of blocks needed and no of files for the index 6 on the disk : 2
7 8
Allocated  File Indexed
6-------->7 : 1
6-------->8 : 1
Do you want to enter more file(Yes - 1/No - 0)0
```

**Post-Experiment Questions:**

1. Describe indexed allocation scheme?
2. Write advantages & disadvantages of indexed allocation scheme?

## Experiment-12

**Aim:** Implementation of worst fit allocation technique.

**Description:** One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory Worst-fit chooses the largest available block.

**Program:**

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - First Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j;
break;
}
}
}
flag[i]=temp;
bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

**Output:**

```
Input
Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of the blocks: -
Block 1: 5
Block 2: 2
Block 3: 7

Enter the size of the files:-
File 1: 1
File 2: 4

Output
File No      File Size    Block No     Block Size    Fragment
1                  1        1              5             4
2                  4        3              7             3
```

**Post-Experiment Questions:**

1. Does Worst fit suffers from fragmentation and why?
2. Write advantages & disadvantages of worst fit scheme?

## Experiment-13

**Aim:** Implementation of Best-fit allocation technique.

**Description:** One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory Best-fit strategy chooses the block that is closest in size to the request.

**Program:**

```
#include<stdio.h>
#include<process.h>
void main()
{
int a[20],p[20],i,j,n,m;
printf("Enter no of Blocks.\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the %dst Block size:",i);
scanf("%d",&a[i]);
}
printf("Enter no of Process.\n");
scanf("%d",&m);
for(i=0;i<m;i++)
{
printf("Enter the size of %dstProcess:",i);
scanf("%d",&p[i]);
}
for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
                {
if(p[j]<=a[i])
                        {
printf("The Process %d allocated to %d\n",j,a[i]);
p[j]=10000;
break;
                        }
                }
}
for(j=0;j<m;j++)
{
if(p[j]!=10000)
                    {
printf("The Process %d is not allocated\n", j);
                    }
}
}
```

**Output:**

```
Enter no of Blocks.
5
Enter the 0st Block size:500
Enter the 1st Block size:400
Enter the 2st Block size:300
Enter the 3st Block size:200
Enter the 4st Block size:100

Enter no of Process. 5
Enter the size of 0st Process:100
Enter the size of 1st Process:350
Enter the size of 2st Process:400
Enter the size of 3st Process:150
Enter the size of 4st Process:200

The Process 0 allocated to 500
The Process 1 allocated to 400
The Process 3 allocated to 200
The Process 2 is not allocated
The Process 4 is not allocated
```

**Post-Experiment Questions:**

1. What type of fragmentation best fit strategy suffers?
2. Write advantages & disadvantages of best fit scheme?

## Experiment-14

**Aim:** Implementation of First-fit allocation technique.

**Description:** One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory First-fit chooses the first available block that is large enough.

**Program:**

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{

for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
ff[i]=j;
highest=temp;
}
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

**Output:**

```
Input
Enter the number of blocks: 3
Enter the number of files: 2

Enter the size of the blocks: -
Block 1: 5
Block 2: 2
Block 3: 7

Enter the size of the files: -
File 1: 1
File 2: 4

Output
File No          File Size        Block No        Block Size       Fragment
1                 1                 3                7                 6
2                 4                 1                5                 1
```

**Post-Experiment Questions:**

1. What is fragmentation? Explain its types.
2. Write advantages & disadvantages of first fit scheme?

## Experiment-15

**Aim:** Calculation of external and internal fragmentation, free space list of blocks from system, List process file from the system.

**Description:**

**IMPLEMENTATION DETAILS:**
**INPUT/s:**
(i) Free space list of blocks from system (as in program 4).

(ii) List processes and files from the system (as in program 4).

**STEPS TO PERFORM:-**
(i) Completing experiment contiguous allocation techniques, we end up getting list of allotted files, remaining part of allotted block and blocks which cannot be allotted

(ii) After implementing each allocation algorithm, list the amount of free space blocks left out after performing allocation.

(iii) When a block which is not at all allocated to any process or file, it adds to external fragmentation.

(iv) When a file or process is allocated the free block and still some part of it is left unused, we count such unused portion into internal fragmentation.

**OUTPUT/s:**
Processes and files allocated to free blocks. From the list of unused blocks, we determine the count of total internal and external fragmentation.

## Experiment-16

**<u>Aim:</u>** Implementation of Compaction for the continually changing memory layout and calculate total movement of data.

**<u>Description:</u>**

Compaction is a process in which the free space is collected in a large memory chunk to make some space available for processes.

In memory management, swapping creates multiple fragments in the memory because of the processes moving in and out.

Compaction refers to combining all the empty spaces together and processes.

Compaction helps to solve the problem of fragmentation, but it requires too much of CPU time.

It moves all the occupied areas of store to one end and leaves one large free space for incoming jobs, instead of numerous small ones.

In compaction, the system also maintains relocation information and it must be performed on each new allocation of job to the memory or completion of job from memory.

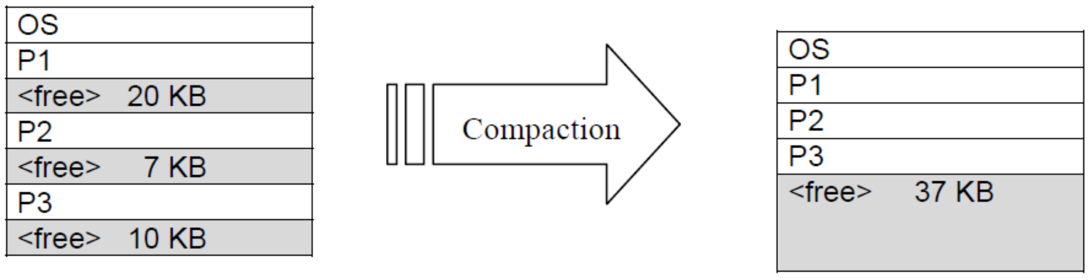
**ALGORITHM**

Compaction is a method to overcome the external fragmentation problem.

All free blocks are brought together as one large block of free space.

Compaction requires dynamic relocation. Certainly, compaction has a cost and selection of an optimal compaction strategy is difficult.

One method for compaction is swapping out those processes that are to be moved within the memory, and swapping them into different memory locations.



**Solution:**

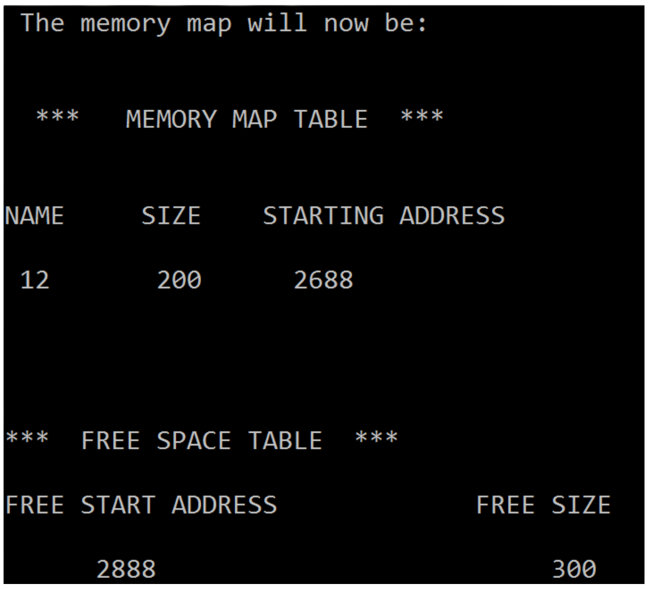
```
#include<stdio.h>
#include<conio.h>

void create(int,int);
void del(int);
void compaction();
void display();

int fname[10],fsize[10],fstart[10],freest[10],freesize[10],m=0,n=0,start;

int main()
  {
    int name,size,ch,i;
    int *ptr;
   // clrscr();
    ptr=(int *) malloc(sizeof(int)*100);
    start=freest[0]=(int)ptr;
    freesize[0]=500;

    printf("\n\n");
    printf(" Free start address         Free Size        \n\n");

    for(i=0;i<=m;i++)
      printf("    %d                     %d\n",freest[i],freesize[i]);
      printf("\n\n");
     while(1)
    {

     printf("1.Create.\n");
     printf("2.Delete.\n");
     printf("3.Compaction.\n");
     printf("4.Exit.\n");
     printf("Enter your choice: ");
        scanf("%d",&ch);
     switch(ch)
      {
        case 1:
           printf("\nEnter the name of file: ");
             scanf("%d",&name);
           printf("\nEnter the size of the file: ");
             scanf("%d",&size);
           create(name,size);
           break;
        case 2:
           printf("\nEnter the file name which u want to delete: ");
             scanf("%d",&name);
           del(name);
```

```
          break;
        case 3:
            compaction();
            printf("\nAfter compaction the tables will be:\n");
            display();
            break;
        case 4:
            exit(1);
        default:
            printf("\nYou have entered a wrong choice.\n");
      }
   }

}


void create(int name,int size)
  {
    int i,flag=1,j,a;

     for(i=0;i<=m;i++)
   if( freesize[i] >= size)
      a=i,flag=0;
     if(!flag)
      {
    for(j=0;j<n;j++);
        n++;
    fname[j]=name;
    fsize[j]=size;
    fstart[j]=freest[a];
    freest[a]=freest[a]+size;
              freesize[a]=freesize[a]-size;

              printf("\n The memory map will now be: \n\n");
    display();
      }
     else
      {
       printf("\nNo enough space is available.System compaction......");

        flag=1;

        compaction();
```

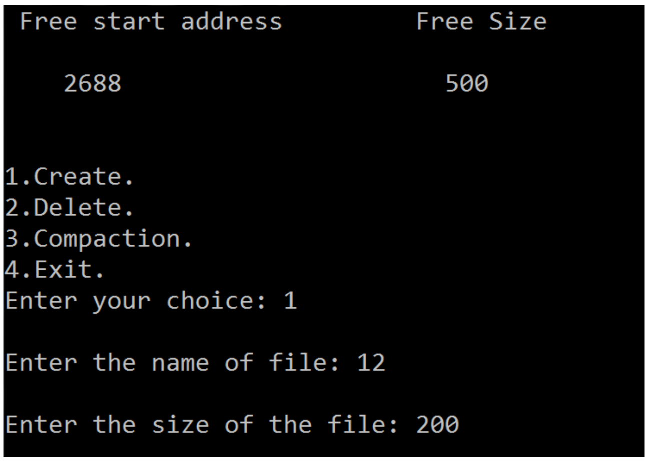
```
         display();

        for(i=0;i<=m;i++)
           if( freesize[i] >= size)
        a=i,flag=0;
     if(!flag)
      {
       for(j=0;j<n;j++);
          n++;
       fname[j]=name;
       fsize[j]=size;
       fstart[j]=freest[a];
       freest[a]+=size;
       freesize[a]-=size;
       printf("\n The memory map will now be: \n\n");
       display();
      }
     else
     printf("\nNo enough space.\n");
       }
  }

void del(int name)
  {
    int i,j,k,flag=1;
     for(i=0;i<n;i++)
       if(fname[i]==name)
           break;
     if(i==n)
       {
    flag=0;
    printf("\nNo such process exists......\n");
       }
      else
       {
       m++;
       freest[m]=fstart[i];
       freesize[m]=fsize[i];
     for(k=i;k<n;k++)
                 {
          fname[k]=fname[k+1];
                  fsize[k]=fsize[k+1];
                  fstart[k]=fstart[k+1];
```

```
                }
                n--;
        }
       if(flag)
        {
      printf("\n\n After deletion of this process the memory map will be : \n\n");
      display();
        }
    }

void compaction()
  {
    int i,j,size1=0,f_size=0;
      if(fstart[0]!=start)
        {
    fstart[0]=start;
    for(i=1;i<n;i++)
      fstart[i]=fstart[i-1]+fsize[i-1];
      }
     else
        {
      for(i=1;i<n;i++)
        fstart[i]=fstart[i-1]+fsize[i-1];
        }
     f_size=freesize[0];

     for(j=0;j<=m;j++)
        size1+=freesize[j];
      freest[0]=freest[0]-(size1-f_size);
      freesize[0]=size1;
      m=0;
 }

void display()
    {
     int i;

     printf("\n  ***   MEMORY MAP TABLE  ***        \n");
     printf("\n\nNAME     SIZE    STARTING ADDRESS      \n\n");
      for(i=0;i<n;i++)
        printf(" %d%10d%10d\n",fname[i],fsize[i],fstart[i]);
     printf("\n\n");
     printf("\n\n***  FREE SPACE TABLE  ***\n\n");
    printf("FREE START ADDRESS          FREE SIZE       \n\n");
    for(i=0;i<=m;i++)
    printf("      %d                    %d\n",freest[i],freesize[i]);
  }
```

**OUTPUT:**



```
 The memory map will now be:


  ***   MEMORY MAP TABLE   ***


NAME      SIZE     STARTING ADDRESS

 12        200       2688




***  FREE SPACE TABLE  ***

FREE START ADDRESS             FREE SIZE

      2888                         300
```



```
 Free start address          Free Size

    2688                       500


1.Create.
2.Delete.
3.Compaction.
4.Exit.
Enter your choice: 1

Enter the name of file: 12

Enter the size of the file: 200
```

**Post-Experiment Questions:**

1. Which problem is solved by using the technique of compaction?
2. Write disadvantages of compaction?

## Experiment-17

**Aim:** Implementation of resource allocation graph (RAG).

**Description**

A resource allocation graph tracks which resource is held by which process and which process is waiting for a resource of a particular type. It is very powerful and simple tool to illustrate how interacting processes can deadlock. If a process is using a resource, an arrow is drawn from the resource node to the process node. If a process is requesting a resource, an arrow is drawn from the process node to the resource node.

If there is a cycle in the Resource Allocation Graph and each resource in the cycle provides only one instance, then the processes will deadlock. For example, if process 1 holds resource A, process 2 holds resource B and process 1 is waiting for B and process 2 is waiting for A, then process 1 and process 2 will be deadlocked.

**ALGORITHM**

(i) List the processes and resources.
(ii)Read input from user for each [$P_i$ ,$R_i$] and also how many instances of each resource to a particular process (for multiple instances case).
(iii) While user completes the input, we end up constructing adjacency matrices/ list.
Two cases to be considered:
**Case 1–** Each resource has single instance (simpler problem).
**Case 2–** Multiple instances of resources (complex problem).

**a.** Methods used for representing graph:

**(i) Adjacency matrix**: A 2-D array of size N x N where N is the number of vertices in the graph (includes processes and resources). For each adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Since resource allocation graph is directed graph, hence it is not necessary to be symmetric.

**(ii) Adjacency list:** An array of linked list is used. Size of the array is equal to number of vertices (processes) in the graph. An entry arr[i] represents the linked list of vertices (resources requested by process) adjacent to the i$^{th}$ vertex.

**Questions:**

1. What is the deadlock in OS?
2. What is the resource allocation graph?

## Experiment 18

**Aim:** Implementation of Banker‟s algorithm.

**Description:** The banker‟s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Following **Data structures** are used to implement the Banker‟s Algorithm:

Let **'n'** be the number of processes in the system and **'m'** be the number of resources types.

**Available:**

It is a 1-d array of size **'m'** indicating the number of available resources of each type.

Available[ j ] = k means there are **'k'** instances of resource type **Rj**

**Max:**

It is a 2-d array of size „**n*m'** that defines the maximum demand of each process in a system.

Max[ i, j ] = k means process **Pi** may request at most **'k'** instances of resource type **Rj.**

**Allocation:**

It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.

Allocation[ i, j ] = k means process **Pi** is currently allocated **'k'** instances of resource type **Rj**

**Need:**

It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.

Need [ i, j ] = k means process **Pi** currently need **'k'** instances of resource type **Rj** for its execution.

Need [ i, j ] = Max [ i, j ] – Allocation [ i, j ]

Allocationi specifies the resources currently allocated to process Pi and Needi specifies the additional resources that process Pi may still request to complete its task.

Banker‟s algorithm consists of Safety algorithm and Resource request algorithm

**Safety Algorithm**

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1) Let Work and Finish be vectors of length „m‟ and „n‟ respectively. Initialize:

Work = Available Finish[i] = false; for i=1, 2, 3, 4….n

2) Find an i such that both

a) Finish[i] = false

b) Needi <= Work if no such i exists goto step (4)

3) Work = Work + Allocation[i] Finish[i] = true goto step (2)

4) if Finish [i] = true for all i then the system is in a safe state

**Resource-Request Algorithm**

Let Requesti be the request array for process Pi. Requesti [j] = k means process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken:

1) If Requesti <= Needi Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum clAim.

2) If Requesti <= Available Goto step (3); otherwise, Pi must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

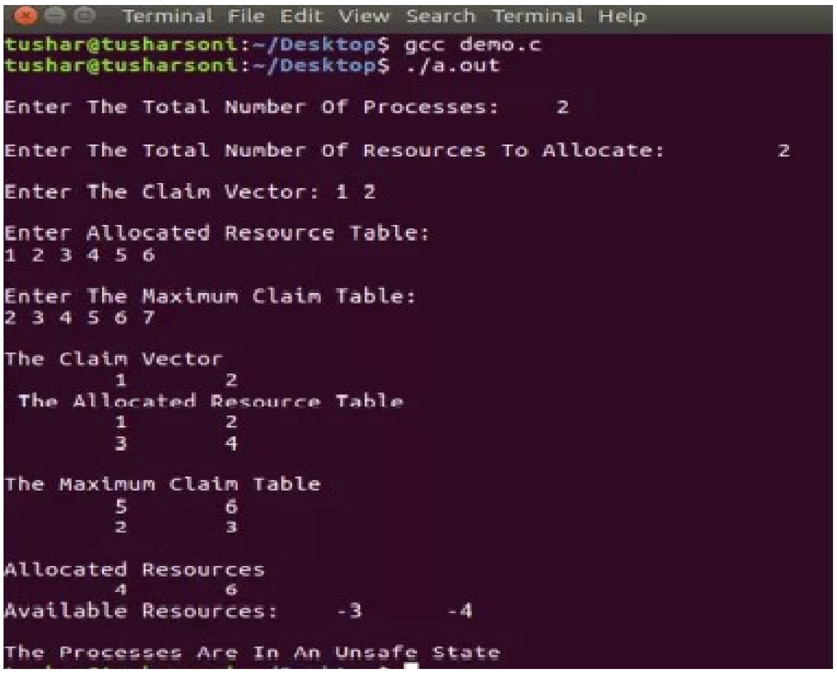
Available = Available – Requesti Allocationi = Allocationi + Requesti Needi = Needi– Requesti

**Program:**

```
#include <stdio.h>
int main()
{
    int count = 0, m, n, process, temp, resource;
    int allocation_table[5] = {0, 0, 0, 0, 0};
    int available[5], current[5][5], maximum_clAim[5][5];
    int maximum_resources[5], running[5], safe_state = 0;
    printf("\nEnter The Total Number Of Processes:\t");
    scanf("%d", &process);
    for(m = 0; m < process; m++)
    {
        running[m] = 1;
        count++;
    }
    printf("\nEnter The Total Number Of Resources To Allocate:\t");
    scanf("%d", &resource);
    printf("\nEnter The ClAim Vector:\t");
    for(m = 0; m < resource; m++)
    {
        scanf("%d", &maximum_resources[m]);
    }
    printf("\nEnter Allocated Resource Table:\n");
    for(m = 0; m < process; m++)
    {
        for(n = 0; n < resource; n++)
        {
            scanf("%d", &current[m][n]);
        }
    }
    printf("\nEnter The Maximum ClAim Table:\n");
    for(m = 0; m < process; m++)
```

```
{
    for(n = 0; n < resource; n++)
    {
        scanf("%d", &maximum_clAim[m][n]);
    }
}
printf("\nThe ClAim Vector \n");
for(m = 0; m < resource; m++)
{
    printf("\t%d ", maximum_resources[m]);
}
printf("\n The Allocated Resource Table\n");
for(m = 0; m < process; m++)
{
    for(n = 0; n < resource; n++)
    {
        printf("\t%d", current[m][n]);
    }
    printf("\n");
}
printf("\nThe Maximum ClAim Table \n");
for(m = 0; m < process; m++)
{
    for(n = 0; n < resource; n++)
    {
        printf("\t%d", maximum_clAim[m][n]);
    }
    printf("\n");
}
for(m = 0; m < process; m++)
{
    for(n = 0; n < resource; n++)
    {
            allocation_table[n] = allocation_table[n] + current[m][n];
        }
    }
    printf("\nAllocated Resources \n");
    for(m = 0; m < resource; m++)
    {
        printf("\t%d", allocation_table[m]);
    }
    for(m = 0; m < resource; m++)
    {
        available[m] = maximum_resources[m] - allocation_table[m];
    }
    printf("\nAvailable Resources:");
    for(m = 0; m < resource; m++)
    {
        printf("\t%d", available[m]);
    }
    printf("\n");
    while(count != 0)
    {
        safe_state = 0;
        for(m = 0; m < process; m++)
        {
            if(running[m])
            {
                temp = 1;
                for(n = 0; n < resource; n++)
                {
                    if(maximum_clAim[m][n] - current[m][n] > available[n])
                    {
                        temp = 0;
                        break;
                    }
                }
                if(temp)
                {
                    printf("\nProcess %d Is In Execution \n", m + 1);
                    running[m] = 0;
                    count--;
                    safe_state = 1;
                    for(n = 0; n < resource; n++)
                    {
                        available[n] = available[n] + current[m][n];
                        }
                        break;
                    }
                }
            }
            if(!safe_state)
            {
                printf("\nThe Processes Are In An Unsafe State \n");
                break;
            }
            else
            {
                printf("\nThe Process Is In A Safe State \n");
                printf("\nAvailable Vector\n");
                for(m = 0; m < resource; m++)
                {
                    printf("\t%d", available[m]);
                }
                printf("\n");
            }
        }
        return 0;
    }
```

**OUTPUT:**



```
tushar@tusharsoni:~/Desktop$ gcc demo.c
tushar@tusharsoni:~/Desktop$ ./a.out

Enter The Total Number Of Processes:    2

Enter The Total Number Of Resources To Allocate:        2

Enter The Claim Vector: 1 2

Enter Allocated Resource Table:
1 2 3 4 5 6

Enter The Maximum Claim Table:
2 3 4 5 6 7

The Claim Vector
        1       2
 The Allocated Resource Table
        1       2
        3       4

The Maximum Claim Table
        5       6
        2       3

Allocated Resources
        4       6
Available Resources:    -3      -4

The Processes Are In An Unsafe State
```

**Post-Experiment Questions:**

1. The Banker's algorithm is used for ____________________.

2.__________ is the situation in which a process is waiting on another process, which is also waiting on another process ... which is waiting on the first process. None of the processes involved in this circular wait are making progress.

3. What is safe state?

4. What are the conditions that cause deadlock?

## Experiment-19

**Aim:** Implement the solution for Bounded Buffer (producer-consumer) problem using inter process communication.

**Description:** In computing, the producer–consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue.

- The producer‟s job is to generate data, put it into the buffer, and start again.
- At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

**Problem :** To make sure that the producer won‟t try to add data into the buffer if it‟s full and that the consumer won‟t try to remove data from an empty buffer.

**Solution :** The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. **An inadequate solution could result in a deadlock** where both processes are waiting to be awakened.

In the post Producer-Consumer solution, we have discussed above solution by using inter-thread communication(wait(), notify(), sleep()). In this post, we will use Semaphores to implement the same.

**The below solution consists of four classes:**

1. **Q** : the queue that you‟re trying to synchronize
2. **Producer :** the threaded object that is producing queue entries
3. **Consumer :** the threaded object that is consuming queue entries
4. **PC :** the driver class that creates the single Q, Producer, and Consumer.

**Program:**

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
int buf[256];
int in = 0;
int out = 0;
sem_t full;
sem_t empty;
sem_t mutex;

int buf_size;
int counter = 0;
void *producer(void *arg)
{
int i, item, *index;
index = (int *)arg;
for (i = 0;; i++)
{
item = 1000 + i;
sem_wait(&empty);
sem_wait(&mutex);
buf[in] = item;
in = (in + 1) % (*index);
counter++;
printf("\n%d [P%d] ", item, *index);
sem_post(&mutex);
sem_post(&full);
/* if (i % 5 == 0)
sleep(1); */
}
}
void *consumer(void *arg)
{
int i, item, *index;
index = (int *)arg;
for (i = 0;;i++)
{
sem_wait(&full);
sem_wait(&mutex);
item = buf[out];
out = (out + 1) % (*index);
counter--;
printf("\n%d [C%d] ", item, *index);
sem_post(&mutex);
sem_post(&empty);
/* if (i % 5 == 0)
sleep(1); */
}
}
int main()
{
int produce, consume;
int i;
printf("\nThe Buffer Size:");
scanf("%d", &buf_size);
printf("\nThe Producer:");
scanf("%d", &produce);
printf("\nThe Consumer:");
scanf("%d", &consume);
pthread_t prod, cons;
void* exit_status;
sem_init(&full, 0, 0);
sem_init(&empty, 0, buf_size);
sem_init(&mutex, 0, 1);
for (i = 0; i < produce; i++)
{
pthread_create(&prod, NULL, producer, &i);
}
for (i = 0; i < consume; i++)
{
pthread_create(&cons, NULL, consumer, &i);
}
pthread_join(prod, &exit_status);
pthread_join(cons, &exit_status);
// pthread_exit(NULL);
return 0;
}
```

**Output**:

```
Producer produced item: 0
Consumer consumed item: 0
Producer produced item: 1
Consumer consumed item: 1
Producer produced item: 2
Consumer consumed item: 2
Producer produced item: 3
Consumer consumed item: 3
Producer produced item: 4
Consumer consumed item: 4
```

**Post Experiment Questions:**

1. Explain the term race condition with example?

2. State and explain critical section problem?

## Experiment-20

**Aim:** Implement the solutions for Readers-Writers problem using inter-process communication technique -Semaphore.

**Description:** Consider a situation where we have a file shared between many people.

- If one of the people tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her.
- However if some person is reading the file, then others may read it at the same time.

Precisely in OS we call this situation as the **readers-writers problem**

Problem parameters:

- One set of data is shared among a number of processes
- Once a writer is ready, it performs its write. Only one writer may write at a time
- If a process is writing, no other process can read it  If at least one reader is reading, no other process can write
- Readers may not write and only read

**Solution when Reader has the Priority over Writer**

Here priority means, no reader should wait if the share is currently opened for reading.

Three variables are used: **mutex, wrt, readcnt** to implement solution

1. **semaphore** mutex, wrt; // semaphore **mutex** is used to ensure mutual exclusion when **readcnt** is updated i.e. when any reader enters or exit from the critical section and semaphore **wrt** is used by both readers and writers

2. **int** readcnt; // **readcnt** tells the number of processes performing read in the critical section, initially 0

**Functions for sempahore :**

– wait() : decrements the semaphore value.

– signal() : increments the semaphore value.

**Writer process:**

1. Writer requests the entry to critical section.

2. If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.

3. It exits the critical section.

**Program:**

```
#include<stdio.h>
#include<conio.h>
#include<stdbool.h>
struct semaphore
{
int mutex;
int rcount;
int rwait;
bool wrt;
};
void addR(struct semaphore *s)
{
if (s->mutex == 0 && s->rcount == 0)
{
printf("\nSorry, File open in Write mode.\nNew Reader added to queue.\n");
s->rwait++;
}
Else {
printf("\nReader Process added.\n");
s->rcount++;
s->mutex--;
}
return ;
}
void addW(struct semaphore *s)
{
if(s->mutex==1)
{
s->mutex--;
s->wrt=1;
printf("\nWriter Process added.\n");
}
else if(s->wrt) printf("\nSorry, Writer already operational.\n");
else printf("\nSorry, File open in Read mode.\n");
return ;
}
void remR(struct semaphore *s)
{
if(s->rcount == 0) printf("\nNo readers to remove.\n");
else
{
printf("\nReader Removed.\n");
s->rcount--;
s->mutex++;
}
return ;
}
void remW(struct semaphore *s)
{
if(s->wrt==0) printf("\nNo Writer to Remove");
else
printf("\nWriter Removed\n");
s->mutex++;
s->wrt=0;
if(s->rwait!=0)
{
s->mutex-=s->rwait;
s->rcount=s->rwait;
s->rwait=0;
printf("%d waiting Readers Added.",s->rcount);
}
}
}
int main()
{
struct semaphore S1={1,0,0};
while(1)
{
//system("cls");
printf("Options :-\n1.Add Reader.\n2.Add Writer.\n3.Remove Reader.\n4.Remove
Writer.\n5.Exit.\n\n\tChoice : ");
int ch;
scanf("%d", &ch);
switch(ch)
{
case 1: addR(&S1); break;
case 2: addW(&S1); break;
case 3: remR(&S1); break;
case 4: remW(&S1); break;
case 5: printf("\n\tGoodBye!"); getch(); return 0;
default: printf("\nInvalid Entry!"); continue;
}
printf("\n\n<<<<<< Current Status >>>>>>\n\n\tMutex\t\t:\t%d\n\tActive Readers\t:\t
%d\n\tWaiting Readers\t:\t%d\n\tWriter Active\t:\t%s\n\n", S1.mutex, S1.rcount, S1.rwait,
(S1.mutex==0 && S1.rcount==0) ? "YES" : "NO");
system("pause");
}}
```

**OUTPUT**

File open in Write mode

```
New reader added to queue
```

**Post-Experiment Questions:**

1. Explain about semaphore?
2. What is inter process communication?

## EXPERIMENT-21

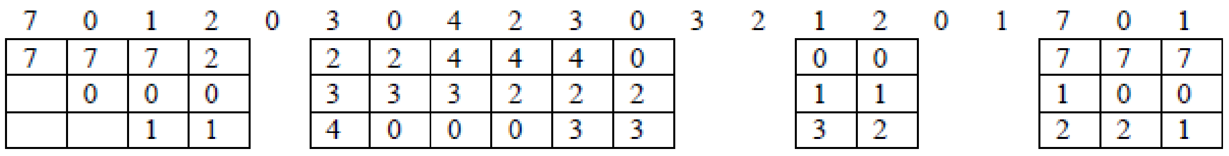
**Aim:** Simulate FIFO page replacement algorithms.

**Descriptions:**

**THEORY:**

**FIFO algorithm**:

The simpler page replacement algorithm is a FIFO algorithm. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replace, the oldest page is chosen. We can create a FIFO queue to hold all pages in memory. We replace the page at the head of the queue when a page is brought into memory; we insert it at the tail of the queue.



**ALGORITHM:**

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

**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:**

```
FIFO PAGE REPLACEMENT ALGORITHM
Enter no.of frames....4
Enter number of reference string. 6
Enter the reference string.
5 6 4 1 2 3
The given reference string:
...................................... 5 6 4 1 2 3
Reference np5-> 5 -1 -1 -1
Reference np6-> 5 6 -1 -1
Reference np4-> 5 6 4 -1
Reference np1-> 5 6 4 1
Reference np2-> 2 6 4 1
Reference np3-> 2 3 4 1
No.of pages faults...6
```

**Post-Experiment 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?

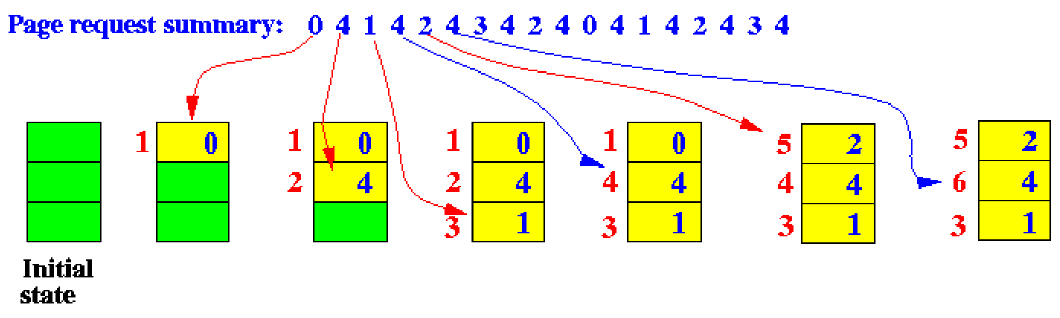
4.Define Frame?

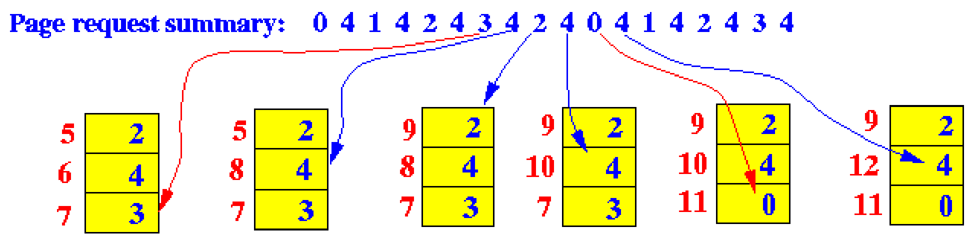
## EXPERIMENT-22

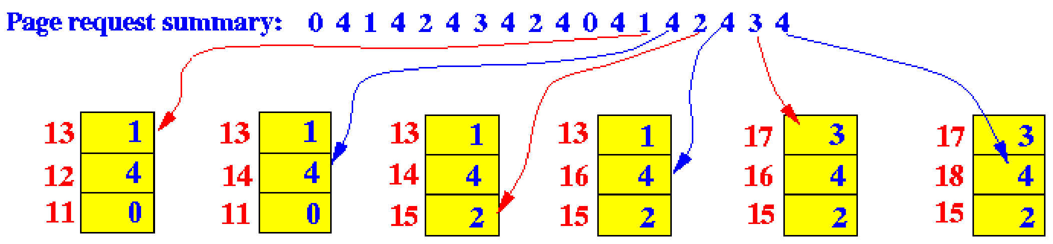
**Aim:** Simulate LRU page replacement algorithms

**Descriptions:**

- In the Least Recently Used (LRU) page replacement policy, the page that is **used least recently** will be replaced.
- Implementation:
  - Add a register to every page frame - contain the last time that the page in that frame was accessed
  - Use a "logical clock" that advance by 1 tick each time a memory reference is made.
  - Each time a page is referenced, update its register
- The following figure shows the behaviour of the program in paging using the LRU page replacement policy:
  - We can see notably that the **bad** replacement decisions made by FIFO **is not present** in LRU.
  - There are total of **9 page read operations** to satisfy the total of 18 page requests - that is almost a 20% improvement over FIFO in such a short experiment
  - (I only want to make the point here that page replacement policy can affect the system performance. I do not want to get into the question of "how much better is LRU than FIFO").
- In fact, it has been shown empirically that LRU is the **preferred page replacement policy**.







**ALGORITHM :**

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

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],lrucal[50],count=0;
int lruvictim();
void main()
{
```

```
clrscr();
printf("\n\t\t\t LRU PAGE REPLACEMENT ALGORITHM");
printf("\n Enter no.of Frames....");
scanf("%d",&nof);
printf(" Enter no.of reference string..");
scanf("%d",&nor);
printf("\n Enter reference string..");
for(i=0;i<nor;i++)
scanf("%d",&ref[i]);
printf("\n\n\t\t LRU PAGE REPLACEMENT ALGORITHM "); 41
printf("\n\t The given reference string:");
printf("\n………………………………..");
for(i=0;i<nor;i++)
printf("%4d",ref[i]);
for(i=1;i<=nof;i++)
{
frm[i]=-1;
lrucal[i]=0;
}
for(i=0;i<10;i++)
recent[i]=0;
printf("\n");
for(i=0;i<nor;i++)
{
flag=0;
printf("\n\t Reference NO %d->\t",ref[i]);
for(j=0;j<nof;j++)
{
if(frm[j]==ref[i])
{
flag=1;
break;
}
}
if(flag==0)
{
count++;
if(count<=nof)
victim++;
else
victim=lruvictim();
pf++;
frm[victim]=ref[i];
for(j=0;j<nof;j++)
printf("%4d",frm[j]);
}
recent[ref[i]]=i;
}
printf("\n\n\t No.of page faults...%d",pf);
getch();
}
int lruvictim()
{
int i,j,temp1,temp2;
for(i=0;i<nof;i++) 42
{
temp1=frm[i];
lrucal[i]=recent[temp1];
}
temp2=lrucal[0];
for(j=1;j<nof;j++)
{
if(temp2>lrucal[j])
temp2=lrucal[j];
}

for(i=0;i<nof;i++)
if(ref[temp2]==frm[i])
return i;
return 0;
}
```

**OUTPUT:**

```
LRU PAGE REPLACEMENT ALGORITHM

Enter no.of Frames....3

Enter no.of reference string..6

Enter reference string..

6 5 4 2 3 1

LRU PAGE REPLACEMENT ALGORITHM

The given reference string:

…………………. 6 5 4 2 3 1

Reference NO 6-> 6 -1 -1

Reference NO 5-> 6 5 -1

Reference NO 4-> 6 5 4
Reference NO 2-> 2 5 4
Reference NO 3-> 2 3 4
Reference NO 1-> 2 3 1
No.of page faults...6
```

**Post-Experiment Questions:**

1.In which of the following page replacement policies, Bolady's anomaly occurs?
(A)FIFO (B)LRU (C)LFU (D)SRU

2. Explain the difference between FIFO and LRU?

3. The operating system manages ________. 1 Memory 2 Processor 3 Disk and I/O devices 4 All of the above

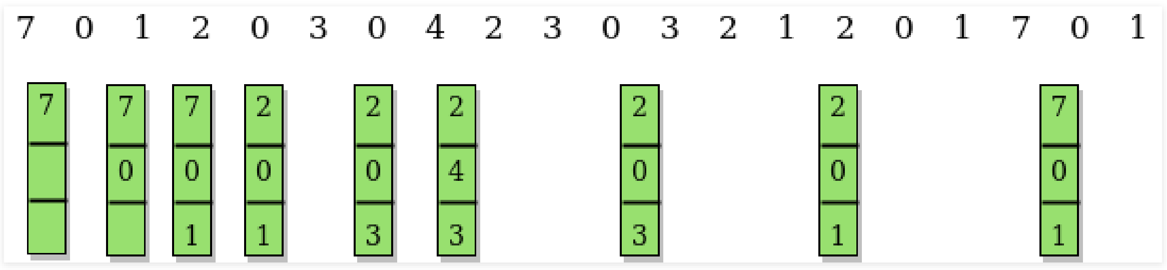
4. A program at the time of executing is called ________. 1 Dynamic program 2 Static program 3 Binded Program 4 A Process

5.The principle of locality of reference justifies the use of ________. 1 Virtual Memory 2 Interrupts 3 Main memory 4 Cache memory

## EXPERIMENT- 23

**Aim:** Simulate Optimal page replacement algorithms

**Descriptions:** In operating systems, whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.
In this algorithm, OS replaces the page that will not be used for the longest period of time in future.



**ALGORITHM**:

1. Start
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that will not be used for the longest period of time.
7. Display the number of page faults.
8. stop

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],optcal[50],count=0;
int optvictim();
void main()
```

```
{
clrscr();
printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHN");
printf("\n................................");
printf("\nEnter the no.of frames");
scanf("%d”, &nof);
printf("Enter the no.of reference string");
scanf("%d",&nor);
printf("Enter the reference string");
for (i=0; i<nor;i++)
scanf("%d",&ref[i]);
clrscr();
printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
printf("\n................................");
printf("\nThe given string");
printf("\n....................\n");
for (i=0; i<nor;i++)
printf("%4d”, ref[i]);
for (i=0; i<nof;i++)
{
frm[i]=-1;
optcal[i]=0;
}
for (i=0; i<10; i++)
recent[i]=0;
printf("\n");
for (i=0; i<nor;i++)
{
flag=0;
printf("\n\tref no %d ->\t",ref[i]);
for (j=0; j<nof; j++)
{
if(frm[j]==ref[i])
{
flag=1;
break;
}
}
if(flag==0)
{
count++;
if(count<=nof)
victim++;
else
victim=optvictim(i);
pf++;
frm[victim]=ref[i];
for (j=0; j<nof; j++)
printf("%4d”, frm[j]);
}
}
printf("\n Number of page faults: %d",pf);
getch();
}
int optvictim(int index)
{
int i,j,temp,notfound;
for (i=0; i<nof;i++)
{
notfound=1;
for(j=index;j<nor;j++)
if(frm[i]==ref[j])
{
notfound=0;
optcal[i]=j;
break;
}
if(notfound==1)
return i;
}
temp=optcal[0];
for (i=1; i<nof;i++)
if(temp<optcal[i])
temp=optcal[i];
for (i=0; i<nof; i++)
if(frm[temp]==frm[i])
return i;
return 0;
}
```

**OUTPUT:**

```
OPTIMAL PAGE REPLACEMENT ALGORITHM

Enter no. of Frames....3

Enter no. of reference string.6

Enter reference string.

6 5 4 2 3 1
```

```
OPTIMAL PAGE REPLACEMENT ALGORITHM
The given reference string:
…………………. 6 5 4 2 3 1
Reference NO 6-> 6 -1 -1
Reference NO 5-> 6 5 -1
Reference NO 4-> 6 5 4
Reference NO 2-> 2 5 4
Reference NO 3-> 2 3 4
Reference NO 1-> 2 3 1
No. of page faults...6
```

**Post-Experiment Questions:**

1. What is the full form of LRU?
2. Explain when page replacement occurs?
3. Which is the best page replacement algorithm? why?
4. FIFO scheduling is ________.
5. Explain various page replacement algorithms?
6. what do u mean by page fault?

# APPENDIX

## AKTU SYLLABUS

## Operating Systems Lab (KCS451)

1. Study of hardware and software requirements of different operating systems (UNIX, LINUX, WINDOWS XP, WINDOWS7/8)
2. Execute various UNIX system calls for
   i. Process management
   ii. File management
   iii. Input/output Systems calls
3. Implement CPU Scheduling Policies:
   i. SJF
   ii. Priority
   iii. FCFS
   iv. Multi-level Queue
4. Implement file storage allocation technique:
   i. Contiguous (using array)
   ii. Linked –list (using linked-list)
   iii. Indirect allocation (indexing)
5. Implementation of contiguous allocation techniques:
   i. Worst-Fit
   ii. Best- Fit
   iii. First- Fit
6. Calculation of external and internal fragmentation
   i. Free space list of blocks from system
   ii. List process file from the system
7. Implementation of compaction for the continually changing memory layout and calculate total movement of data
8. Implementation of resource allocation graph RAG)
9. Implementation of Banker's algorithm
10. Conversion of resource allocation graph (RAG) to wait for graph (WFG) for each type of method used for storing graph.
11. Implement the solution for Bounded Buffer (producer-consumer) problem using inter process communication techniques-Semaphores
12. Implement the solutions for Readers-Writers problem using inter process communication technique -Semaphore