**DEPARTMENT OF**

**COMPURER SCIENCE AND ENGINEERING**

**LAB MANUAL**

**R20[A]**

**OPERATING SYSTEMS & LINUX LAB**

**[III B.TECH, I-SEM]**

**Prepared by**

Asst.prof.G.Beulah Rani

****

|  |  |  |  |
| --- | --- | --- | --- |
| Document NO:  KITS/CSE/LAB MANUAL/Operating System & Linux lab | Date of issue: | Compiled by | Authorized by  Prof.R.RAMESH  HOD  DEPT. OF CSE |
|  | Date of revision | Verified by |

**VISION OF THE INSTITUTE**

To produce eminent and ethical Engineers and Managers in society by imparting quality professional education with emphasis on human values and holistic excellence.

**MISSION OF THE INSTITUTE**

* To incorporate benchmarked teaching and learning pedagogies in curriculum.
* To ensure all round development of students through judicious blend of curricular, co curricular and extracurricular activities.
* To support cross-cultural exchange of knowledge between industry and academy.
* To provide higher/continued education and research opportunities to the employees of the institution.

**DEPARTMENT VISION**

* To become a reputed center in computer Science and systems engineering for quality, competency and social responsibility

**DEPARTMENT MISSION**

* Strengthen the Core Competence with Vibrant Technological Education in a congenial Environment
* Promote innovative research and development for the Economic, Social and Environment.
* Inculcate professional behavior, strong ethical values to meet the challenges in collaboration and lifelong learning

**PROGRAM OUTCOMES**

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

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

**3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

**4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

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

**6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

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

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

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

**10. Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

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

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

**PROGRAM EDUCATIONAL OBJECTIVES OF CSE DEPARTMENT**

|  |
| --- |
| **PEO 1:**  Domain Knowledge: Have a strong foundation in areas like mathematics, science and engineering fundamentals so as to enable them to solve and analyze engineering problems and prepare them to careers, R&D and studies of higher level. |
| **PEO 2:**  Professional Employment: Have an ability to analyze and understand the requirements of software, technical specifications required and provide novel engineering solutions to the problems associated with hardware and software. |
| **PEO 3:**  Higher Degrees: Have exposure to cutting edge technologies thereby making them to achieve excellence in the areas of their studies. |
| **PEO 4:**  Engineering Citizenship: Work in teams on multi-disciplinary projects with effective communication skills and leadership qualities. |
| **PEO 5:**  Lifelong Learning: Have a successful career wherein they strike a balance between ethical values and commercial values. |

|  |
| --- |
| **PROGRAM SPECIFIC OUTCOME (PSO’S)** |
| **PSO1: Application Development**  Able to develop the business solutions through Latest Software   Techniques and tools for real time Applications. |
| **PSO2: Professional and Leadership**  Able to practice the profession with ethical leadership as an entrepreneur through participation in various events like Ideathon, Hackathon, project expos and workshops. |
| **PSO3: Computing Paradigms**  Ability to identify the evolutionary changes in computing   using Data Sciences, Apps, Cloud computing and IoT. |

**Operating Systems & Linux Programming Laboratory**

**List of Experiments**

**Linux Programming**

1. a) Study of Unix/Linux general purpose utility command list man,who,cat, cd, cp, ps, ls, mv, rm, mkdir, rmdir, echo, more, date, time, kill, history, chmod, chown, finger, pwd, cal, logout, shutdown.

b) Study of vi editor.

c) Study of Bash shell, Bourne shell and C shell in Unix/Linux operating system.

d) Study of Unix/Linux file system (tree structure).

e) Study of .bashrc, /etc/bashrc and Environment variables.

2. Write a C program that makes a copy of a file using standard I/O, and system call

3. Write a C program to emulate the UNIX ls –l command.

4. Write a C program that illustrates how to execute two commands concurrently

with a command pipe.

Ex: - ls –l | sort

5. Write a C program that illustrates two processes communicating using shared

memory

6. Write a C program to simulate producer and consumer problem using semaphores

7. Write C program to create a thread using p threads library and let it run its function.

8. Write a C program to illustrate concurrent execution of threads using p threads library.

**Operating system**

9. Simulate the following CPU scheduling algorithms

a) Round Robin b) SJF c) FCFS d) Priority

10. Multiprogramming-Memory management- Implementation of fork (), wait (), exec() and exit (), System calls

11. Simulate the following

Multiprogramming with a fixed number of tasks (MFT)

Multiprogramming with a variable number of tasks (MVT)

12. Simulate Bankers Algorithm for Dead Lock Avoidance

13. Simulate Bankers Algorithm for Dead Lock Prevention.

14. Simulate the following page replacement algorithms.

a) FIFO b) LRU c) LFU

15. Simulate the following File allocation strategies

a) Sequenced b) Indexed c) Linked

**Web References:**

* **<https://www.linux.com/topic/system-administration/>**
* **<https://www.geeksforgeeks.org/important-linux-commands-leave-diff-cal-ncal-locate-ln/?ref=lbp>**
* **<https://www.javatpoint.com/linux-directories>**
* **<https://www.w3schools.in/operating-system/intro>**
* **<https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/?ref=lbp>**

**COURSE OBJECTIVES:**

• To understand the design aspects of operating system

• To study the process management concepts & Techniques

• To study the storage management concepts

• To familiarize students with the Linux environment • To learn the fundamentals of shell scripting/programming

**Course Outcomes:**

CO-1: To use UNIX utilities and perform basic shell control of the utilities

CO-2: To use the UNIX file system and file access control

CO-3: To use of an operating system to develop software

CO-4: Students will be able to use Linux environment efficiently

CO-5: Solve problems using bash for shell scripting

**MAPPING OF CO’s and PO’s:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** |
| **CO3.1.7.1** |  |  | 2 |  | 3 |  |  |  |  |  |  |  |
| **CO3.1.7.2** |  | 2 |  | 3 |  |  |  |  |  |  |  |  |
| **CO3.1.7.3** |  | 2 | 3 |  |  |  |  |  |  |  |  |  |
| **CO3.1.7.4** |  |  |  | 2 | 3 |  |  |  |  |  |  |  |
| **CO3.1.7.5** |  | 3 | 2 |  |  |  |  |  |  |  |  |  |
| **CO3.1.7.6** | 3 |  |  |  | 2 |  |  |  |  |  |  |  |

**MAPPING OF CO’S AND PSO’S**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **PSO1** | **PSO2** | **PSO3** | **PSO4** | **PSO5** |
| **CO3.1.7.1** | 3 |  |  |  |  |
| **CO3.1.7.2** | 3 |  | 2 |  |  |
| **CO3.1.7.3** | 3 | 2 |  |  |  |
| **CO3.1.7.4** | 2 | 3 |  |  |  |
| **CO3.1.7.5** | 3 |  |  |  |  |
| **CO3.1.7.6** | 2 | 3 |  |  |  |

***Note:*** 1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High)

**Recommended System/Software Requirements:**

* Intel based desktop PC with minimum of 2.6GHZ or faster processor with at least 256 MB RAM and 40GB free disk space.
* Operating system: Flavor of any WINDOWS.
* Software: C/CPP/JAVA and Putty software

**List of Experiments**

|  |  |  |
| --- | --- | --- |
| **WEEK NO.** | **SUB**  **EXP.** | **NAME OF THE EXPERIMENT** |
| **LINUX PROGRAMMING LAB PROGRAMS** | | |
| 1 | a | Study of Unix/Linux general purpose utility command list: man,who,cat, cd, cp, ps, ls, mv, rm, mkdir, rmdir, echo, more, date, time, kill, history, Chmod, chown, finger, pwd, cal, logout, shutdown. |
| b | Study of vi editor |
| c | Study of Bash shell, Bourne shell and C shell in Unix/Linux operating system |
| d | Study of Unix/Linux files system (tree structure) |
| e | Study of .bashrc, /etc/bashrc and Environment variables. |
| 2 |  | Write a C program that makes a copy of a file using standard I/O, and system calls |
| 3 |  | Write a C program to emulate the UNIX ls –l command. |
| 4 |  | Write a C program that illustrates how to execute two commands concurrently with a command pipe. Ex: - ls –l | sort |
| 5 |  | To write a C program that illustrates two processes communicating using sharedmemory . |
| 6 |  | Write a C program to simulate producer and consumer problem using semaphores. |
| 7 |  | To create a thread using pthreads library and let it run its function. |
| 8 |  | **T**o illustrate concurrent execution of threads using pthreads library. |

|  |  |  |  |
| --- | --- | --- | --- |
| **OPERATING SYSTEM PROGRAMS** | | | |
| 9 | |  | Simulate the following CPU scheduling algorithms: |
| a | Round Robin |
| b | SJF |
| c | FCFS |
| d | Priority |
| 10 | |  | Multiprogramming-Memory management-Implementation of fork (), wait (), exec() and exit ()(), System calls |
| 11 | |  | Simulate the following: |
| a | M Multiprogramming with a fixed number of tasks (MFT) |
| b | Multiprogramming with a variable number of tasks (MVT) |
| 12 |  | | Simulate Bankers Algorithm for Dead Lock Avoidance | |
| 13 | |  | Simulate Bankers Algorithm for Dead Lock Prevention. |
| 14 | |  | Simulate the following page replacement algorithms: |
| a | FIFO |
| b | LRU |
| c | LFU |
| 15 | |  | 1. Simulate the following File allocation strategies |
|  | | a | Sequenced |
| b | Indexed |
| c | Linked |

**Additional Experiments**

|  |  |
| --- | --- |
| **S. No** | **Program Name** |
| 1 | Loading executable programs into memory |
| 2 | Execute System Call implementation- read(), write(), open () and close() |
| 3 | shell script that accepts a file name and displays all the lines between the given line numbers |
| 4 | shell script that displays a list of all files in the current directory |

**MAPPING OF CO’S with Lab Experiments**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EXPEREMENT | **CO1** | **CO2** | **CO3** | **CO4** | **CO5** | **CO6** |
|  | 3 |  |  |  |  |  |
|  |  |  |  |  | 2 |  |
|  | 3 |  |  |  |  |  |
|  |  | 3 |  |  |  |  |
|  |  | 3 |  |  |  |  |
|  |  |  | 3 |  |  |  |
|  | 3 |  |  |  |  |  |
|  |  |  |  | 3 | 3 |  |
|  |  |  |  |  | 2 |  |
|  |  |  |  | 2 |  |  |
|  |  |  |  |  | 3 |  |
|  |  |  |  |  |  | 3 |
|  |  |  |  |  | 2 |  |
|  |  |  |  |  |  | 2 |
|  |  |  |  |  |  | 2 |
|  | 2 |  |  |  |  |  |
|  | 2 |  |  |  |  |  |
|  |  |  |  |  | 3 |  |
|  |  |  |  |  | 3 |  |

Level of Mapping: 1 – Slightly 2 – Moderate 3 – Highly

**Syllabus:**

**UNIT I** **Operating Systems Overview**: Operating system functions, Operating system structure, Operating systems operations, Computing environments, Open-Source Operating Systems.

**System Structures**: Operating System Services, User and Operating-System Interface, systems calls, Types of System Calls, system programs, operating system structure, operating system debugging, System Boot.

**UNIT II** **Process Concept**: Process scheduling, Operations on processes, Inter-process communication, Communication in client server systems. Multithreaded Programming: Multithreading models, Thread libraries, Threading issues. Process Scheduling: Basic concepts, Scheduling criteria, Scheduling algorithms, Multiple processor scheduling, Thread scheduling.

**Inter-process Communication**: Race conditions, Critical Regions, Mutual exclusion with busy waiting, Sleep and wakeup, Semaphores, Mutexes, Monitors, Message passing, Barriers, Classical IPC Problems - Dining philosophers problem, Readers and writers problem.

**UNIT III Memory-Management Strategies**: Introduction, Swapping, Contiguous memory allocation, Paging, Segmentation. Virtual Memory Management: Introduction, Demand paging, Copy on-write, Page replacement, Frame allocation, Thrashing, Memory-mapped files, Kernel memory allocation.

**UNIT IV Deadlocks**: Resources, Conditions for resource deadlocks, Ostrich algorithm, Deadlock detection and recovery, Deadlock avoidance, Deadlock prevention. File Systems: Files, Directories, File system implementation, management and optimization. Secondary-Storage Structure: Overview of disk structure, and attachment, Disk scheduling, RAID structure, Stable storage implementation.

**UNIT V System Protection**: Goals of protection, Principles and domain of protection, Access matrix, Access control, Revocation of access rights. System Security: Introduction, Program threats, System and network threats, Cryptography for security, User authentication, implementing security defenses, Firewalling to protect systems and networks, Computer security classification.

**Case Studies**: Linux, Microsoft Windows.

**Text Books**:

Silberschatz A, Galvin P B, and Gagne G, Operating System Concepts, 9th edition, Wiley, 2013. 2) Tanenbaum A S, Modern Operating Systems, 3rd edition, Pearson Education, 2008. (for Inter process Communication and File systems.)

**Reference Books:**

1) Dhamdhere D M, Operating Systems A Concept Based Approach, 3rd edition, Tata McGraw-Hill, 2012.

2) Stallings W, Operating Systems -Internals and Design Principles, 6th edition, Pearson Education, 2009

3) Nutt G, Operating Systems, 3rd edition, Pearson Education, 2004

**E-Resources:**

**<https://nptel.ac.in/courses/106/105/106105214/>**

**Course Objectives**:

The objectives of this course is to

• Introduce to the internal operation of modern operating systems

• Define, explain, processes and threads, mutual exclusion, CPU scheduling, deadlock, memory management, and file systems

• Understand File Systems in Operating System like UNIX/Linux and Windows

• Understand Input Output Management and use of Device Driver and Secondary Storage (Disk) Mechanism

• Analyze Security and Protection Mechanism in Operating System

**Course Outcomes**:

At the end of the course, the students will be able to:

**CO-1**: Describe various generations of Operating System and functions of Operating System

**CO-2**: Describe the concept of program, process and thread and analyze various CPU Scheduling Algorithms and compare their performance

**CO-3:** Solve Inter Process Communication problems using Mathematical Equations by various methods

**CO-4**: Compare various Memory Management Schemes especially paging and Segmentation in Operating System and apply various Page Replacement Techniques

**CO-5**: Outline File Systems in Operating System like UNIX/Linux and Windows

**LINUX PROGRAMMING LAB PROGRAMS**

**Experiment no: 1**

**Objective:** to know the use of basic linux commands

**1. a) Study of Unix/Linux general purpose utility command list**

**man, who, cat, cd, cp, ps, ls, mv, rm, mkdir, rmdir, echo, more, date, time, kill, history, chmod,chown, finger, pwd, cal, logout, shutdown.**

**man**

Short for "manual," man allows a user to format and display the user manual built into Linux distributions, which documents commands and other aspects of the system.

**Syntax**

man [option(s)] keyword(s)

**Example**

man ls

**who:** identifies the users currently logged in

The "who" command lets you display the users that are currently logged into your UNIX computer system. The following information is displayed: login name, workstation name, date and time of login. Entering who am i or who am I displays your login name, workstation name,date and time you logged in.

**Synopsys**

who [OPTION]... [ FILE | ARG1 ARG2 ]

**Example**

who am i

**cat:** concatenate or display files

**Synopsys**

cat [- q] [- s] [- S] [- u] [- n[- b]] [- v [- [- t] ] [- | File ... ]

The cat command reads each File parameter in sequence and writes it to standard output. If you do not specify a file name, the cat command reads from standard input. You can also specify a file name of – (minus) for standard input.

*Exit Status*

This command returns the following exit values: 0 All input files were output successfully.

>0 An error occurred.

**Examples**

1. To display a file at the workstation, enter: cat notes
2. To concatenate several files, enter:

cat section1.1 section1.2 section1.3 >section1

1. To suppress error messages about files that do not exist, enter: cat -q section2.1 section2.2 section2.3 >section2
2. To append one file to the end of another, enter:

cat section1.4 >>section1

The >> appends a copy of section1.4 to the end of section1. If you want to replace the file, use the >.

1. To add text to the end of a file, enter: cat >>notes

Get milk on the way home Ctrl-D

1. To concatenate several files with text entered from the keyboard, cat section3.1 - section3.3 >section3
2. To concatenate several files with output from another command, li | cat section4.1 - >section4

**cd:**

The cd command, which stands for "change directory", changes the shell's current working directory.

**Syntax**

cd directory

**Example**

cd new

cd .

cd ..

**pwd***:*print name of current/working directory

|  |  |  |
| --- | --- | --- |
| Syntax: | pwd [OPTION]... | |
| example: | pwd -L |  |
| options:-L, --logical | | use PWD from environment, even if it contains symlinks |
| -P, --physical | | avoid all symlinks |
| --help | | display this help and exit |
| --version | | output version information and exit |

*clear -* clear the terminal screensyntax:clear

*history:* it displays commands executed by userssyntax: history

example:history

**kill** *-*send a signal to a process.The default signal for kill is SIGTERMsyntax:

kill [ -signal | -s signal ] pid ...

kill [ -L | -V, --version ] kill -l [ signal ]

EXAMPLES kill -9 -1

Kill all processes you can kill. kill -l 11

Translate number 11 into a signal name. kill -L

List the available signal choices in a nice table. kill 123 543 2341 3453

Send the default signal, SIGTERM, to all those processes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Num | Action | Description | |
| 0 | 0 | n/a | exit code indicates if a signal may be sent | |
| ALRM | 14 | exit |  |  |
| HUP | 1 | exit |  |  |
| INT | 2 | exit |  |  |
| KILL | 9 | exit | cannot be blocked | |
| PIPE | 13 | exit |  |  |
| POLL |  | exit |  |  |
| PROF |  | exit |  |  |
| TERM | 15 | exit | |  |
| USR1 |  | exit |  |  |
| USR2 |  | exit |  |  |
| Name | Num Action | | | Description |
| VTALRM | |  | exit |  |
| STKFLT | |  | exit | might not be implemented |
| PWR |  | ignore | | might exit on some systems |
| WINCH | | ignore | |  |
| CHLD |  | ignore | |  |
| URG |  | ignore | |  |
| TSTP |  | stop | | might interact with the shell |
| TTIN |  | stop | | might interact with the shell |
| TTOU |  | stop | | might interact with the shell |
| STOP |  | stop | | cannot be blocked |
| CONT |  | restart | | continue if stopped, otherwise ignore |
| ABRT | 6 |  | core |  |
| FPE | 8 | core | |  |
| ILL | 4 |  | core |  |
| QUIT | 3 |  | core |  |
| SEGV | 11 |  | core |  |
| TRAP | 5 |  | core |  |
| SYS |  |  | core | might not be implemented |
| EMT |  |  | core | might not be implemented |
| BUS |  |  | core | core dump might fail |
| XCPU |  |  | core | core dump might fail |
| XFSZ |  |  | core | core dump might fail |

**date** - print or set the system date and timesyntax:

date [OPTION]... [+FORMAT]

date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

options:

-d, --date=STRING display time described by STRING, not `now'

-f, --file=DATEFILE like --date once for each line of DATEFILE

-r, --reference=FILE display the last modification time of FILE

examples

$ date --date='@2147483647' :Convert seconds since the epoch (1970-01-01 UTC) to a date

$ TZ='America/Los\_Angeles' date :Show the time on the west coast of the US (use tzselect(1) to find TZ)

$ date --date='TZ="America/Los\_Angeles" 09:00 next Fri' :Show the local time for 9AM next Friday on the west coast of the US

**time -** run programs and summarize system resource usagesyntax: time [ -apqvV ] [ -f FORMAT ]

options:

[ -o FILE ]-o FILE, --output=FILE

Write the resource use statistics to FILE instead of to the standard error stream. By default, this overwrites the file, destroying the file's previous contents. This option is useful

for collecting information on interactive programs and programs that produce output on the standard error stream.

-a, --append

Append the resource use information to the output file instead of overwriting

it. This option is only useful with the `-o' or `--output' op‐ tion.

**more** *—*file perusal filter for crt viewing.more is a filter for paging through text onescreenful at a time. This version is especially primitive. Users should realize that less(1) provides more(1) emulation plus extensive enhancements.

syntax: more [-dlfpcsu] [-num] [+/pattern] [+linenum] [file ...]

|  |  |  |
| --- | --- | --- |
| options: | |  |
| -d | display help instead of ring bell | |
| -f | count logical, rather than screen lines | |
| -l | suppress pause after form feed | |
| -p | suppress scroll, clean screen and disblay text | |
| -c | suppress scroll, display text and clean line ends | |
| -u | suppress underlining | |
| -s | squeeze multiple blank lines into one | |
| -NUM | | specify the number of lines per screenful |
| +NUM | | display file beginning from line number NUM |

+/STRING display file beginning from search string match

-V output version information and exit

**echo** *-*display a line of text

Syntax:

echo [SHORT-OPTION]... [STRING]...

echo LONG-OPTION

options:

-n do not output the trailing newline

-e enable interpretation of backslash escapes

-E disable interpretation of backslash escapes (default)

--help display this help and exit

--version output version information and exit

**rmdir** *-*remove empty directories

Syntax: rmdir [OPTION]... DIRECTORY...

options:

-p, --parents

remove DIRECTORY and its ancestors; e.g., `rmdir -p a/b/c' is

similar to `rmdir a/b/c a/b a' -v, --verbose

output a diagnostic for every directory processed --help display this help and exit

**mkdir** - make directories

Syntax: mkdir [OPTION]... DIRECTORY...

options:

-m, --mode=MODE

set file mode (as in chmod), not a=rwx - umask -p, --parents

|  |  |  |
| --- | --- | --- |
|  |  | no error if existing, make parent directories as needed |
|  | -v, --verbose | |
|  |  | print a message for each created directory |
| **rm** - remove files or directories | | |
|  |  |  |
| Syntax: | | rm [OPTION]... FILE... |
| options: | |  |
| -f, --force | | |
|  |  | ignore nonexistent files, never prompt |
|  | -i prompt before every removal | |
|  | -I | prompt once before removing more than three files, or when |
|  |  |  |

removing recursively. Less intrusive than -i, while still giv‐ ing protection against most mistakes

**mv -** move (rename) files

syntax: mv [OPTION]... SOURCE... DIRECTORY options;

-b like --backup but does not accept an argument -f, --force

do not prompt before overwriting -i, --interactive

prompt before overwrite

**cal** *:*displays a calendar and the date of Easter. If arguments are not specified, thecurrent month is displayed.

syntax: cal [-hjy] [-A number] [-B number] [[month] year] examples: cal -hj 09 2015

cal -hjy

options:

-A number

Display the number of months after the current month. -B number

Display the number of months before the current month. -C Switch to cal mode.

-y Display a calendar for the specified year. -h Turns off highlighting of today.

-J Display Julian Calendar, if combined with the -e option, display date of Easter according to the Julian Calendar.

**login, logout:** - write utmp and wtmp entries

The utmp file records who is currently using the system. The wtmp file records all logins and logouts. See utmp(5).

The function login() takes the supplied struct utmp, ut, and writes it to both the utmp and the wtmp file.

The function logout() clears the entry in the utmp file again.

**shutdown:** bring the system down

syntax: shutdown [OPTION]... TIME [MESSAGE] example:

shutdown -h 17:25 shutdown -r +5

options:

-r Requests that the system be rebooted after it has been brought down.

-h Requests that the system be either halted or powered off after it has been brought down, with the choice as to which left up to

the system.

-c Cancels a running shutdown. TIME is not specified with this option, the first argument iWrite a C program that illustrates how to execute

two commands concurrently with a command pipe. s MESSAGE.

**1.b) Study of vi editor.**

The vi editor is available on almost all Unix systems. vi can be used from any type of terminal because it does not depend on arrow keys and function keys--it uses the standard alphabetic keys for commands.

vi (pronounced "vee-eye") is short for "vi"sual editor. It displays a window into the file being edited that shows 24 lines of text. vi is a text editor, not a "what you see is what you get" word processor. vi lets you add, change, and delete text, but does not provide such formatting capabilities as centering lines or indenting paragraphs.

This help note explains the basics of vi:

opening and closing a file

moving around in a file

elementary editing

**===== Starting vi =====**

You may use vi to open an already existing file by typing

vi filename

where "filename" is the name of the existing file. If the file is not in your current directory, you must use the full pathname.

Or you may create a new file by typing *vi newname*

where "newname" is the name you wish to give the new file.

To open a new file called "testvi," enter *vi testvi*

On-screen, you will see blank lines, each with a tilde (~) at the left, and a line at the bottom giving the name and status of the new file:

~

~

"testvi" [New file]

**===== vi Modes =====**

vi has two modes:

command mode

insert mode

In command mode, the letters of the keyboard perform editing functions (like moving the cursor,deleting text, etc.). To enter command mode, press the escape &<Esc> key.

In insert mode, the letters you type form words and sentences. Unlike many word processors, vi

starts up in command mode.

===== **Entering Text** =====

In order to begin entering text in this empty file, you must change from command mode to insert mode. To do this, type ‗*i'*

Nothing appears to change, but you are now in insert mode and can begin typing text. In

general, vi's commands do not display on the screen and do not require the Return key to be pressed.

Type a few short lines and press &<Return> at the end of each line. If you type a long line, you will notice the vi does not word wrap, it merely breaks the line unceremoniously at the edge of the screen. If you make a mistake, pressing <Backspace> or <Delete> may remove the error, depending on your terminal type.

===== **Moving the Cursor** =====

To move the cursor to another position, you must be in command mode. If you have just

finished typing text, you are still in insert mode. Go back to command mode by pressing <Esc>.

If you are not sure which mode you are in, press <Esc> once or twice until you hear a beep.

When you hear the beep, you are in command mode.

The cursor is controlled with four keys: h, j, k, l.

**Key Cursor Movement**

h left one space

j down one line

k up one line

l right one space

When you have gone as far as possible in one direction, the cursor stops moving and you hear a beep. For example, you cannot use l to move right and wrap around to the next line, you must use j to move down a line. See the section entitled "Moving Around in a File" for ways to move more quickly through a file.

**Basic Editing**

Editing commands require that you be command mode. Many of the editing commands have a different function depending on whether they are typed as upper- or lowercase. Often, editing commands can be preceded by a number to indicate a repetition of the command.

**Deleting Characters**

To delete a character from a file, move the cursor until it is on the incorrect letter, then type ‗*x’*

The character under the cursor disappears. To remove four characters (the one under the cursor and the next three) type 4x

To delete the character before the cursor, type X (uppercase)

**Deleting Words**

To delete a word, move the cursor to the first letter of the word, and type *dw*

This command deletes the word and the space following it. To delete three words type

*3dw*

**Deleting Lines**

To delete a whole line, type *dd*

The cursor does not have to be at the beginning of the line. Typing dd deletes the entire line containing the cursor and places the cursor at the start of the next line. To delete two lines, type *2dd*. To delete from the cursor position to the end of the line, type *D* (uppercase)

**Replacing Characters**

To replace one character with another:

1. Move the cursor to the character to be replaced.
2. Type r
3. Type the replacement character.

The new character will appear, and you will still be in command mode.

**Replacing Words**

To replace one word with another, move to the start of the incorrect word and type *cw*

The last letter of the word to be replaced will turn into a $. You are now in insert mode and may type the replacement. The new text does not need to be the same length as the original.

Press <Esc> to get back to command mode. To replace three words, type *3cw*

**Replacing Lines**

To change text from the cursor position to the end of the line:

1. Type C (uppercase).
2. Type the replacement text.
3. Press <Esc>.

**Inserting Text**

To insert text in a line:

1. Position the cursor where the new text should go.
2. Type i
3. Enter the new text. The text is inserted BEFORE the cursor.
4. Press <Esc> to get back to command mode.

**Appending Text**

To add text to the end of a line:

1. Position the cursor on the last letter of the line.
2. Type a
3. Enter the new text. This adds text AFTER the cursor.
4. Press <Esc> to get back to command mode.

**Opening a Blank Line**

To insert a blank line below the current line, type *o* (lowercase) To insert a blank line above the current line, type *O* (uppercase)

**Joining Lines**

To join two lines together:

1. Put the cursor on the first line to be joined.
2. Type J

To join three lines together:

1. Put the cursor on the first line to be joined.
2. Type 3J

===== **Undoing** =====

To undo your most recent edit, type *u*

To undo all the edits on a single line, type *U* (uppercase)

Undoing all edits on a single line only works as long as the cursor stays on that line. Once you move the cursor off a line, you cannot use U to restore the line.

===== **Moving Around in a File** =====

There are shortcuts to move more quickly though a file. All these work in command mode.

**Key Movement**

w forward word by word b backward word by word $ to end of line

0 (zero) to beginning of line H to top line of screen

M to middle line of screen L to last line of screen

G to last line of file 1G to first line of file

<Control>f scroll forward one screen <Control>b scroll backward one screen <Control>d scroll down one-half screen <Control>u scroll up one-half screen

===== **Moving by Searching** =====

To move quickly by searching for text, while in command mode:

1. Type / (slash).
2. Enter the text to search for.
3. Press <Return>.

The cursor moves to the first occurrence of that text.

To repeat the search in a forward direction, type *n*

To repeat the search in a backward direction, type *N*

===== **Closing and Saving a File** =====

With vi, you edit a copy of the file, rather than the original file. Changes are made to the original

only when you save your edits.

To save the file and quit vi, type **ZZ**

The vi editor is built on an earlier Unix text editor called ex. ex commands can be used within vi. ex commands begin with a : (colon) and end with a <Return>. The command is displayed on the status line as you type. Some ex commands are useful when saving and closing files.

To save the edits you have made, but leave vi running and your file open:

1. Press <Esc>.
2. Type :w
3. Press <Return>.

To quit vi, and discard any changes your have made since last saving:

1. Press <Esc>.
2. Type :q!
3. Press <Return>.

**1.c) Study of Bash shell, Bourne shell and C shell in Unix/Linux operating system.**

**Types of Shells in Linux**

In addition to graphical user interfaces like Gnome, KDE and MATE, the Linux operating system also offers several shells. These command-line interfaces provide powerful environments for software development and system maintenance. Though shells have many commands in common, each type has unique features. Over time, individual programmers come to prefer one type of shell over another; some develop new, enhanced shells based on previous ones. UNIX also has an ecosystem of different shells; Linux carries this practice into the open-source software arena.

**The Bourne shell**

The Bourne shell, called "sh," is one of the original shells, developed for Unix computers by Stephen Bourne at AT&T's Bell Labs in 1977. Its long history of use means many software developers are familiar with it. It offers features such as input and output redirection, shell scripting with string and integer variables, and condition testing and looping.

**The Bash shell**

The popularity of sh motivated programmers to develop a shell that was compatible with it, but with several enhancements. Linux systems still offer the sh shell, but "bash" -- the "Bourne-again Shell," based on sh -- has become the new default standard. One attractive feature of bash is its ability to run sh shell scripts unchanged. Shell scripts are complex sets of commands that automate programming and maintenance chores; being able to reuse these scripts saves programmers time. Conveniences not present with the original Bourne shell include command completion and a command history.

**C Shell**

Developers have written large parts of the Linux operating system in the C and C++ languages.

Using C syntax as a model, Bill Joy at Berkeley University developed the "C-shell," csh, in 1978. Ken Greer, working at Carnegie-Mellon University, took csh concepts a step forward with a new shell, tcsh, which Linux systems now offer. Tcsh fixed problems in csh and added command completion, in which the shell makes educated "guesses" as you type, based on your system's directory structure and files. Tcsh does not run bash scripts, as the two have substantial differences.

**The Korn shell**

David Korn developed the Korn shell, or ksh, about the time tcsh was introduced. Ksh is compatible with sh and bash. Ksh improves on the Bourne shell by adding floating-point arithmetic, job control, and command aliasing and command completion. AT&T held proprietary rights to ksh until 2000, when it became open source.

**1.d) Study of Unix/Linux file system (tree structure).**

A file system is a logical collection of files on a partition or diskUNIX uses a hierarchical file system structure, much like an upside-down tree, with root (/) at the base of the file system and all other directories spreading from there.

A UNIX filesystem is a collection of files and directories that has the following properties −

It has a root directory (/) that contains other files and directories.

Each file or directory is uniquely identified by its name, the directory in which it resides, and a unique identifier, typically called an inode.

By convention, the root directory has an inode number of 2 and the lost+found directory has an inode number of 3. Inode numbers 0 and 1 are not used. File inode numbers can be seen by specifying the -i option to ls command.

It is self contained. There are no dependencies between one filesystem and any other.

The directories have specific purposes and generally hold the same types of information for easily locating files. Following are the directories that exist on the major versions of

Unix −

Directory Description

/ This is the root directory which should contain only the directories needed at the top

level of the file structure.

/bin This is where the executable files are located. They are available to all user.

/dev These are device drivers.

/etc Supervisor directory commands, configuration files, disk configuration files, valid

user lists, groups, ethernet, hosts, where to send critical messages.

/lib Contains shared library files and sometimes other kernel-related files.

/boot Contains files for booting the system.

/home Contains the home directory for users and other accounts.

/mnt Used to mount other temporary file systems, such as cdrom and floppy for the CDROM drive and floppy diskette drive, respectively

/proc Contains all processes marked as a file by process number or other information that is dynamic to the system.

/tmp Holds temporary files used between system boots

/usr Used for miscellaneous purposes, or can be used by many users. Includes administrative commands, shared files, library files, and others

/var Typically contains variable-length files such as log and print files and any other type of file that may contain a variable amount of data

/sbin Contains binary (executable) files, usually for system administration. For example fdisk and ifconfig utlities.

/kernel Contains kernel files

**1.e) Study of .bashrc, /etc/bashrc and Environment variables.**

Following is the partial list of important environment variables.

**Variable Description**

**DISPLAY** Contains the identifier for the display that X11 programs should

use by default.

**HOME** Indicates the home directory of the current user: the default

argument for the cd built-in command.

**IFS** Indicates the Internal Field Separator that is used by the parser for

word splitting after expansion.

**LANG** LANG expands to the default system locale; LC\_ALL can be used to

override this. For example, if its value is pt\_BR, then the languageis set to (Brazilian) Portuguese and the locale to Brazil.

**LD\_LIBRARY\_PATH** On many Unix systems with a dynamic linker, contains acolonseparated list of directories that the dynamic linker should search for shared objects when building a process image after exec, before searching in any other directories.

**PATH** Indicates search path for commands. It is a colon-separated list of directories inwhich the shell looks for commands.

**PWD** Indicates the current working directory as set by the cd command.

**RANDOM** Generates a random integer between 0 and 32,767 each time it is

referenced.

**SHLVL** Increments by one each time an instance of bash is started. This

variable is useful for determining whether the built-in exit command ends the current session.

**TERM** Refers to the display type

**TZ** Refers to Time zone. It can take values like GMT, AST, etc.

**UID** Expands to the numeric user ID of the current user, initialized at

shell startup.

**Viva Questions:**

1.Create a read-only file in your home directory?

2.How do you find how many cpu are in your system and there details?

3.What is "chmod" command? What do you understand by this line “r-- -w- --x?

4.How do you find whether your system is 32 bit or 64 bit ?

5.How do you set environment variable which will be accessible form sub shell?

6.What different operating systems is vi available for?

.

7.What are some of the vi clones that are available?

8.How to quit without saving?

9.You have started up vi, but now want to be able to edit another file. How do you do this?

10.How do I search for a control sequence?

**EXPERIMENT: 2**

**Write a c program that makes a copy of a file using standard I/O and system calls**

**Aim:** copy a file using standard I/O and system calls

**Directions:**

1. open one existing file 1234.c in read mode using system call.
2. open one non-existing file zyx in write mode using system call.
3. read one character from existing file and store it in a buffer using file pointer .
4. write the same character to the new file if it is not a blank space .

**Algorithm:**

Step 1: start.

Step 2: open one existing file one.txt in read mode using file pointer.

fp=fopen(argv[1],"r");

Step 3: open one non-existing file two.txt in write mode using file descriptor.

fd=open(argv[2],O\_WRONLY);

Step 4: read one character from existing file and store it in a buffer using file pointer .

buf=fgetc(fp);

Step 5: write the same character to the new file if it is not a blank space two.txt using file descriptor.

write(fd,&buf,1);

Step 6: repeate step 3 and 4 upto end of the file.

while(!feof(fp))

Step 7: stop.

**Library functions and system calls:**

*fopen() :* a library function to open an existing file for read/write/append operation. Itreturns a pointer which points to the opened file. It is called file pointer.

*Open():* a system call to open an existing file and returns an intiger which is known asfile descriptor of that file.

*Fgetc():* read a character from the file specified in file pointer.

**Program**

#include<stdlib.h>

#include<fcntl.h>

#define Buff\_size 8192

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

{

int input\_fd,output\_fd;

ssize\_t ret\_in,ret\_out;

char buffer[Buff\_size];

if(argc!=3)

{

printf("\n Arguments must be 3");

return 1;

}

input\_fd=open(argv[1],O\_RDONLY);

if(input\_fd==-1)

{

perror("\n Some error in opening");

return 2;

}

output\_fd=open(argv[2],O\_WRONLY|O\_CREAT,0644);

if(output\_fd==-1)

{

perror("\n Some error in opening");

return 3;

}

while((ret\_in=read(input\_fd,&buffer,Buff\_size))>0)

{

ret\_out=write(output\_fd,&buffer,(ssize\_t)ret\_in);

if(ret\_out!=ret\_in)

{

perror("Some error in writing i,e copying");

return 4;

}

}

close(input\_fd);

close(output\_fd);

return (EXIT\_SUCCESS);

}

**EXPECTED OUTPUT:-**

[srinivas]$ vi copy.c

[srinivas]$ cc copy.c

[srinivas]$ ./a.out f1 f2

[srinivas]$ cat f2

hi

this is Srinivas

[srinivas]$

**Viva questions:**

1.What is the differenc between fopen() and open()

2.What is file pointer?

3.What is file descriptor

**EXPERIMENT NUMBER: 3**

**AIM: Write a C program to emulate the Unix ls-l command.**

**Description:**

The UNIX ls -l command can provide you with detailed information about each file and subdirectory in the current folder. Using this function is equivalent to entering "dir /V" at the Windows command prompt. The ls -l UNIX command reveals seven facts about each item in a directory. Here is an example of its

**Program:**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main()

{

int pid;

pid=fork();

if(fork<0)

{

printf("\n Fork failed");

exit(-1);

}

else if(pid==0)

{

execlp("/bin/ls","ls","-l",NULL);

}

else

{

printf("\n Child process complete");

exit(0);

}

}

**Output**

guest-glcbIs@ubuntu:~$gcc –o lsc.out lsc.c

guest-glcbIs@ubuntu:~$./lsc.out

total 100

-rwxrwx—x 1 guest-glcbls guest-glcbls 140 2012-07-06 14:55 f1

drwxrwxr-x 4 guest-glcbls guest-glcbls 140 2012-07-06 14:40 dir1 child complete

**Viva Questions:**

1.what is the task of getwd(str)?

2.what is the task of alphasort().

3.why scandir() is used?

4.dirent stands for \_\_\_\_\_\_\_\_?

5.what is the third argument in the scandir?

**EXPERIMENT NUMBER:4**

**AIM: To write a C program that illustrates how to execute two commands concurrently with a command pipe**

**OBJECTIVE**: To learn the function of pipe.

**DIRECTION:**

**1.**create a pipe using pipe system call2.create a child

3.in parent process execute a command using exec family 4.in child execute a command using exec family

**ALGORITHM:**

Step 1: start

Step 2:create a pipe using pipe system call k=pipe(fd);

Step 3:create a child using fork pid=fork();

Step 4:in parent process execute a command using exec family execlp(argv[1],argv[1],NULL);

Step 5:in child execute a command using exec family execlp(argv[2],argv[2],NULL);

Step 6:stop

**Program**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

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

{

int fd[2],pid,k;

k=pipe(fd);

if(k==-1)

{

perror("\n Pipe error");

exit(1);

}

pid=fork();

if(pid==0)

{

close(fd[0]);

dup2(fd[1],1);

close(fd[1]);

execlp(argv[1],argv[1],NULL);

perror("Exec error");

}

else

{

close(fd[1]);

dup2(fd[0],0);

close(fd[0]);

execlp(argv[2],argv[2],NULL);

perror("Exec error");

}

}

**Viva Questions**:

1.What is pipe()?

2.What is the return value of pipe()?

3.What is the task of exec family members?

4.Is it mandatory to close the reading and while writing and close the writing end while reading.

5.What is the header file for exit().

**EXPERIMENT NUMBER:5**

**AIM**: To write a C program that illustrates two processes communicating using sharedmemory .

**OBJECTIVE**:To learn the functions of shared memory.

**DIRECTIONS:**

1**.**create a shared memory using system call 2.write data in shared memory

3.read data from shared memory 4.remove shared memory segment

**ALGORITHM:**

Step 1:start

Step 2:create a shared memory using system call shmget()

shmid=shmget(key, SEGSIZE, IPC\_CREAT |IPC\_EXCL | 0666) Step 3: Write data to shared memory

strcpy(segptr,buff); Step 4: Read data from shared memory

segptr=shmat(shmid,0,0))==(char\*)-1) printf("DATA:-%s\n",segptr);

Step 5: Remove shared memory Segment shmctl(shmid,IPC\_RMID,0)== -1)

Step 6:stop

**Program**

**Code for server:**

#include<sys/shm.h>

#include<stdio.h>

#include<stdlib.h>

main()

{

int size=27;

int shmid;

char \*shm,\*s,c;

key\_t key;

key=12345678;

shmid=shmget(key,size,IPC\_CREAT|0666);

if(shmid==-1)

{

perror("Shmget error");

exit(0);

}

shm=shmat(shmid,NULL,0);

if(shm==(char\*)-1)

{

perror("Shm error");

exit(1);

}

s=shm;

for(c='a';c<='z';c++)

{

\*s++=c;

\*s='\0';

}

printf("\n Successful");

}

**Output:**

[srinivas]$ vi server.c

[srinivas]$ cc server.c

[srinivas]$ ./a.out

Successful[srinivas]$

**Code for client:**

#include<sys/shm.h>

#include<stdio.h>

#include<stdlib.h>

main()

{

int size=27;

int shmid;

char \*shm,\*s,c;

key\_t key;

key=12345678;

shmid=shmget(key,size,0666);

if(shmid==-1)

{

perror("Shmget error");

exit(0);

}

shm=shmat(shmid,NULL,0);

if(shm==(char\*)-1)

{

perror("Shm error");

exit(1);

}

s=shm;

for(s=shm;\*s!='\0';s++)

putchar(\*s);

}

**Output:**

[srinivas]$ cc client.c

[srinivas]$ ./a.out

abcdefghijklmnopqrstuvwxyz [srinivas]$

**Viva Questions:**

1.How shared memory are accessed if we have two procesors trying to acquire the same region.

2.What is shmget()?

3.What is the return of shmget()?

4.what is ftok()?

**EXPERIMENT NO: 6**

**Write a C program to simulate producer and consumer problem using semaphores**

**ALGORITHM:**

1. Start the process
2. Initialize buffer size
3. Consumer enters, before that producer buffer was not empty.
4. Producer enters, before check consumer consumes the buffer.
5. Stop the process.

**Program**

#include<stdio.h>

#include<stdlib.h>

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

main()

{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

printf("\n 1.Producer\n 2.Consumer\n 3.Exit\n");

while(1)

{

printf("\n Enter your choice");

scanf("%d",&n);

switch(n)

{

case 1:

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

producer();

else

printf("\n Bufferis full");

break;

case 2:

if((mutex==1)&&(full!=0))

consumer();

else

printf("\n Buffer is empty");

break;

case 3:

exit(0);

break;

}

}

}

int wait(int s)

{

return(--s);

}

int signal(int s)

{

return(++s);

}

void producer()

{

mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

printf("\n Producer produces item %d",x);

mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

printf("\n Consumer consumes the item %d",x);

x--;

mutex=signal(mutex);

}

**OUTPUT:**

[root@localhost ~]$ ./a.out 1.PRODUCER 2.CONSUMER

3.EXIT

ENTER YOURCHOICE 1

producer produces the item1

ENTER YOUR CHOICE 1

producer produces the item2

ENTER YOUR CHOICE

2

consumer consumes item2

ENTER YOUR CHOICE

2

consumer consumes item1

ENTER YOUR CHOICE

2

BUFFER IS EMPTY

ENTER YOUR CHOICE

3

**Viva Questions:**

1.What are pthread\_cond\_initializer, pthread\_cond\_initializer, pthread\_mutex\_initializer.

2.What is p thread \_create().

3.What is p thread\_join()?

4.What is producer consumer problem.

5.what is p thread\_t?

**Experiment no:7**

**Aim:** to create a thread using pthreads library and let it run its function

**Objective:** to start with thread

**Directions:**

1.create a thread by pthread\_creat().

2. use pthread\_join to wait until termination of thread.

**Algorithm:**

***Algorithm main()****:* step 1: start

step 2: Declare thread identifier pthread\_t pth;

step 3:create a thread and assign the identifier to pth pthread\_create(&pth,NULL,threadFunc,"I am running");

Step 4:repeate this step 5 times print ―control in main‖

step 5:main waits till the termination of the thread pthread\_join(pth,NULL);

step 6:print ―main is terminated‖ step 7: stop

***Algorithm threadFunc():***

step 1: start

step 2: read the 4th arg of ―pthread\_create(..);‖ and assign to a void pointer void \*threadFunc(void \*arg)

step 3:declare a pointer to character string char \*str;

step 4:assign that void pointer to a string pointer str=(char\*)arg;

step 5:print 10 times ―thread function says : 'passesd string'

step 6: stop

**Functions used:**

pthread\_create(): it creates a thread. It has four arguments.

Arg1: thread identifier

Arg2: NULL pointer

Arg3:function that has to be executed on that thread

Arg4: string passed to that function

pthread\_join(): it makes the calling function wait upto the termination of the thread whose id is mentioned as 1st argument. It has two arguments.

Arg1: the thread id for which calling function has to wait

Arg2: NULL pointer

**Program**

#include<stdio.h>

#include<pthread.h>

void \* mythread(void \* argv)

{

printf("Hello world");

return NULL;

}

int main()

{

pthread\_t tid;

pthread\_create(&tid,NULL,\* mythread,NULL);

pthread\_join(tid,NULL);

return 0;

}

**Output**

[srinivas]$ vi pthread.c

[srinivas]$ cc -pthread pthread.c

[srinivas]$ ./a.out

Hello world[srinivas]$

**Viva Questions:**

1.What is pthread\_create().

2.What is pthread\_join()?

3.what is pthread\_t?

4.Why thread is used?

5.what is the command to compile thread program?

**Experiment no:8**

**Aim:** to illustrate concurrent execution of threads using pthreads library

**Objective:** to observe the executions of threads

**Directions:**

1.create two threads in main

2wait the main() to finish up the child threads

3.main terminated.

**Algorithm:**

**Algorithm main():**

step 1: Start

step 2: Declare thread identifier pthread\_t pth;

step 3:

step 3.1:create a thread and assign the identifier to pth1

pthread\_create(&pth1,NULL,threadFunc,"Thread one

running");

step 3.2:create a thread and assign the identifier to pth2

pthread\_create(&pth2,NULL,threadFunc,"Thread two

running");

Step 4:repeate this step 5 times print ―control in main‖

step 5:main waits till the termination of the threads pthread\_join(pth1,NULL);pthread\_join(pth2,NULL);

step 6:print ―main is terminated‖ step 7: stop

**Algorithm threadFunc():**

step 1: start

step 2: read the 4th arg of ―pthread\_create(..);‖ and assign to a void pointer

void \*threadFunc(void \*arg)

step 3:declare a pointer to character string

char \*str;

step 4:assign that void pointer to a string pointer

str=(char\*)arg;

step 5:print 10 times ―thread function says : 'passesd string' ‖

step 6: stop

**Functions used:**

pthread\_create(): it creates a thread. It has four arguments.

Arg1: thread identifier

Arg2: NULL pointer

Arg3:function that has to be executed on that thread

Arg4: string passed to that function

pthread\_join(): it makes the calling function wait upto the termination of the thread whose id is mentioned as 1st argument. It has two arguments.

Arg1: the thread id for which calling function has to wait

Arg2: NULL pointer

**Program**

#include<stdio.h>

#include<pthread.h>

void \*mythread1(void \*argv)

{

int i;

for(i=0;i<5;i++)

printf("Hello");

return NULL;

}

void \*mythread2(void \*argv)

{

int i;

for(i=0;i<5;i++)

printf("World");

return NULL;

}

int main()

{

pthread\_t tid1;

pthread\_t tid2;

pthread\_create(&tid1,NULL,\*mythread1,NULL);

pthread\_create(&tid2,NULL,\*mythread2,NULL);

pthread\_join(tid1,NULL);

pthread\_join(tid2,NULL);

return 0;

}

Output:

[srinivas]$ vi pthread.c

[srinivas]$ cc -pthread pthread.c

[srinivas]$ ./a.out

Hello world[srinivas]$ vi pthreadcon.c

[srinivas]$ cc -pthread pthreadcon.c

[srinivas]$ ./a.out

HelloHelloHelloHelloHelloWorldWorldWorldWorldWorld[srinivas]$

**Viva Questions**:

1.why it shows different output in different computer?

2.What is pthread\_join()?

3.what is pthread\_t?

4.Why thread is used?

5.what is the command to compile thread program?

**EXPERIMENT NO.9**

**CPU SCHEDULING ALGORITHMS**

**A).FIRST COME FIRST SERVE:**

**AIM**: To write a c program to simulate the cpu scheduling algorithm First Come First Serve (FCFS)

**DESCRIPTION:**

To calculate the average waiting time using the FCFS algorithm first the waiting time of the first process is cept zero and the waiting time of the second process is the burst time of the first process and the waiting time of the third process is the sum of the burst times of the first and the second process and so on . after calculating all the waiting times the average waiting time is calculated as the average of all the waiting times. FCFS mainly says first come first serve the algorithm which came first will be served first.

**ALGORITHM:**

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process name and the burst time

Step 4: Set the waiting of the first process as ‗0‘and its burst time as its turnaround time Step 5: for each process in the Ready Q calculate

a). Waiting timess(n)= waiting time (n-1) + Burst time (n-1)

b). Turnaround time (n)= waiting time(n)+Burst time(n)

Step 6: Calculate

1. Average waiting time = Total waiting Time / Number of process
2. Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

**Program:**

#include<stdio.h>

#include<conio.h>

#include<process.h> void main()

{

char p[10][5];

int tot=0,wt[10],i,n;

float avg=0; clrscr();

printf("enter no of processes:"); scanf("%d",&n); for(i=0;i<n;i++)

{

printf("enter process%d name:**\n**",i+1); scanf("%s",&p[i]);

printf("enter process time"); scanf("%d",&pt[i]);

}

wt[0]=0;

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

{

wt[i]=wt[i-1]+et[i-1]; tot=tot+wt[i];

}

avg=(float)tot/n;

printf("p\_name**\t** P\_time**\t** w\_time**\n**"); for(i=0;i<n;i++) printf("%s**\t**%d**\t**%d**\n**",p[i],et[i],wt[i]);

printf("total waiting time=%d**\n** avg waiting time=%f",tot,avg); getch();

}

**Expected input/output:**

enter no of processes: 5

enter process1 name: aaa

enter process time: 4

enter process2 name: bbb

enter process time: 3

enter process3 name: ccc

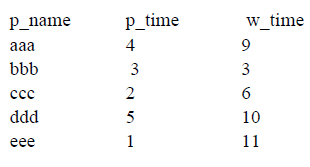
enter process time: 2

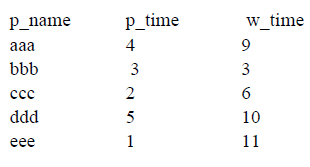
enter process4 name: ddd

enter process time: 5

enter process5 name: eee

enter process time: 1





total waiting time=34

avg waiting time=6.80

**Viva-voce Questions**

1. Define CPU scheduling.

1. What is a Dispatcher?

.

1. What is turnaround time?

1. Define dispatch latency?
2. What is job scheduling?

**B).SHORTEST JOB FIRST:**

**AIM**: To write a C program to simulate the CPU scheduling algorithm,

shortest job first with pre-emption.

**DESCRIPTION:**

To find the average waiting time of the process in the SJF preemption, there will be the arrival times for each process after the arrival. We will check for the burst times of all the previously arrived processes and will execute the shortest job and the process will enter. After the arrival of all the processes, the average waiting time can be calculated.

**ALGORITHM:**

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time and process arrival time.

|  |  |
| --- | --- |
| Step 4: Start the Ready Q according the shortest Burst time by sorting according to |  |
| lowest to highest burst time. |  |
| Step 5: Set the waiting time of the first process as ‗0‘ and its turnaround time as |  |

burst time.

Step 6: Sort the processes names based on their Burt time and Arrival time

Step 7: For each process in the ready queue, calculate

a). Waiting timess(n)= waitingtime (n-1) + Burst time (n-1)-arrivaltime(n) b). Turnaround time (n)= waiting time(n)+Burst time(n)

Step 8: Calculate and print the results

1. Average waiting time = Total waiting Time / Number of process
2. Average Turnaround time = Total Turnaround Time / Number of process

Step 9: Stop the process

**Program:**

#include<stdio.h>

#include<conio.h>

#include<process.h> void main()

{

char p[10][5],temp[5];

int tot=0,wt[10],pt[10],i,j,n,temp1; float avg=0;

clrscr();

printf("enter no of processes:"); scanf("%d",&n); for(i=0;i<n;i++)

{

printf("enter process%d name:**\n**",i+1); scanf("%s",&p[i]);

printf("enter process time"); scanf("%d",&pt[i]);

}

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

{

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

{

if(pt[i]>pt[j])

{

temp1=pt[i];

pt[i]=pt[j];

pt[j]=temp1;

strcpy(temp,p[i]);

strcpy(p[i],p[j]);

strcpy(p[j],temp);

}

}

}

wt[0]=0;

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

{

wt[i]=wt[i-1]+et[i-1]; tot=tot+wt[i];

}

avg=(float)tot/n;

printf("p\_name**\t** P\_time**\t** w\_time**\n**");

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

printf("%s**\t**%d**\t**%d**\n**",p[i],et[i],wt[i]);

printf("total waiting time=%d**\n** avg waiting time=%f",tot,avg);

getch();

}

**Expected input/output:**

enter no of processes: 5

enter process1 name: aaa

enter process time: 4

enter process2 name: bbb

enter process time: 3

enter process3 name: ccc

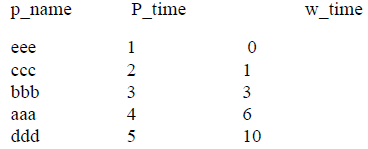
enter process time: 2

enter process4 name: ddd

enter process time: 5

enter process5 name: eee

enter process time: 1



total waiting time=20

avg waiting time=4.00

**Viva-voce Questions**

1. What is symmetric multiprocessing?

2. List out the types in mainframe systems

3. What is file-server systems?

4.What is job scheduling?

**C) ROUND ROBIN**

**AIM**: To simulate the CPU scheduling algorithm round-robin.

**DESCRIPTION:**

To aim is to calculate the average waiting time. There will be a time slice, each process should be executed within that time-slice and if not it will go to the waiting state so first check whether the burst time is less than the time-slice. If it is less than it assign the waiting time to the sum of the total times. If it is greater than the burst-time then subtract the time slot from the actual burst time and increment it by time-slot and the loop continues until all the processes are completed.

**ALGORITHM:**

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue and time quantum

(or) time slice

Step 3: For each process in the ready Q, assign the process id and accept the

CPU burst time

Step 4: Calculate the no. of time slices for each process where

No. of time slice for process (n) = burst time process(n)/time slice

Step 5: If the burst time is less than the time slice then the no. of time slices =1.

Step 6: Consider the ready queue is a circular Q, calculate

1. Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1 ) + the time difference in getting the CPU from process(n-1)
2. Turnaround time for process(n) = waiting time of process(n) + burst time of

process(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

1. Average waiting time = Total waiting Time / Number of process
2. Average Turnaround time = Total Turnaround Time / Number of process Step 8: Stop the process

**Program:**

#include<stdio.h>

#include<conio.h>

#include<process.h>

#include<string.h>

void main{

char p[10][5];

int et[10],wt[10],timer=3,count,pt[10],rt,i,j,totwt=0,t,n=5,found=0,m; float avgwt;

clrscr();

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

{

printf("enter the process name : "); scanf("%s",&p[i]);

printf("enter the processing time : "); scanf("%d",&pt[i]);

}

m=n;

wt[0]=0;

i=0;

do

{

if(pt[i]>timer)

{

rt=pt[i]-timer; strcpy(p[n],p[i]); pt[n]=rt; et[i]=timer; n++;

}

else

{

et[i]=pt[i];

}

i++; wt[i]=wt[i-1]+et[i-1]; }while(i<n);

count=0;

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

{

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

{

if(strcmp(p[i],p[j])==0)

{

count++;

found=j;

}

}

if(found!=0)

{

wt[i]=wt[found]-(count\*timer); count=0;

found=0;

}

}

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

{

totwt+=wt[i];

}

avgwt=(float)totwt/m;

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

{

printf("**\n**%s**\t**%d**\t**%d",p[i],pt[i],wt[i]);

}

printf("**\n**total waiting time %d**\n**",totwt); printf("total avgtime %f",avgwt);

}

**Expected Input/Output**

INPUT :

enter the process name : aaa

enter the processing time : 4

enter the process name : bbb

enter the processing time : 3

enter the process name : ccc

enter the processing time : 2

enter the process name : ddd

enter the processing time : 5

enter the process name : eee

enter the processing time : 1

|  |  |  |
| --- | --- | --- |
| OUTPUT |  |  |
| p\_name | p\_time | w\_time |
| aaa | 4 | 9 |
| bbb | 3 | 3 |
| ccc | 2 | 6 |
| ddd | 5 | 10 |
| eee | 1 | 11 |

total waiting time : 39

average waiting time : 7.8000

**Viva-voce Questions**

1. Define CPU scheduling.
2. What is a Dispatcher?
3. What is turnaround time?

4. Define dispatch latency?

**D) PRIORITY:**

**AIM**: To write a c program to simulate the cpu scheduling priority algorithm.

**DESCRIPTION:**

To calculate the average waiting time in the priority algorithm, sort the burst times according to their priorities and then calculate the average waiting time of the processes. The waiting time of each process is obtained by summing up the burst times of all the previous processes.

**ALGORITHM:**

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Sort the ready queue according to the priority number.

Step 5: Set the waiting of the first process as ‗0‘ and its burst time as its turnaround time

Step 6: Arrange the processes based on process priority Step 7:For each process in the Ready Q calculate

Step 8: for each process in the Ready Q calculate

a). Waiting timess(n)= waiting time (n-1) + Burst time (n-1)

b). Turnaround time (n)= waiting time(n)+Burst time(n)

Step 9: Calculate

1. Average waiting time = Total waiting Time / Number of process
2. Average Turnaround time = Total Turnaround Time / Number of process Print the results in an order.

Step 10: Stop the process

**Program:**

#include<stdio.h>

#include<conio.h> void main()

{

char p[10][5],temp[5];

int i,j,pt[10],wt[10],totwt=0,pr[10],temp1,n; float avgwt;

clrscr();

printf("enter no of processes:"); scanf("%d",&n);

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

{

printf("enter process%d name:",i+1); scanf("%s",&p[i]);

printf("enter process time:"); scanf("%d",&pt[i]);

printf("enter-priority:"); scanf("%d",&pr[i]);

}

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

{

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

{

if(pr[i]>pr[j])

{

temp1=pr[i];

pr[i]=pr[j];

pr[j]=temp1;

temp1=pt[i];

pt[i]=pt[j];

pt[j]=temp1;

strcpy(temp,p[i]);

strcpy(p[i],p[j]);

strcpy(p[j],temp);

}

}

}

wt[0]=0;

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

{

wt[i]=wt[i-1]+et[i-1]; totwt=totwt+wt[i];

}

avgwt=(float)totwt/n;

printf("p\_name**\t** p\_time**\t** priority**\t** w\_time**\n**"); for(i=0;i<n;i++)

{

printf(" %s**\t** %d**\t** %d**\t** %d**\n**" ,p[i],pt[i],pr[i],wt[i]);

}

printf("total waiting time=%d**\n** avg waiting time=%f",tot,avg); getch();

}

**Expected input/output:**

enter no of processes: 5

enter process1 name: aaa enter process time: 4 enter priority:5

enter process2 name: bbb enter process time: 3 enter priority:4

enter process3 name: ccc enter process time: 2 enter priority:3

enter process4 name: ddd

enter process time: 5

enter priority:2

enter process5 name:eee

enter process time: 1

enter priority:1

|  |  |  |
| --- | --- | --- |
| p\_name | P\_time priority | w\_time |
| eee | 1 | 1 0 |
| ddd | 5 | 2 1 |
| ccc | 2 | 3 6 |
| bbb | 3 | 4 8 |
| aaa | 4 | 115 |

total waiting time=26

avg waiting time=5.20

**Viva-voce Questions**

1. Define CPU scheduling.

1. What is a Dispatcher?
2. What is turnaround time?
3. Define dispatch latency?
4. What is job scheduling?

**Experiment no: 10**

**Question**: Multiprogramming-Memory management- Implementation of Fork(), Wait(),Exec() and Exit() System calls

**Objective**:A simulation program for “*ls”*command and using command line arguments.

**Algorithm for main(integer argumentCount, String argvalue[])**

Step 1: Start 2: Set pid:=fork() //creating child processed 3: If pid=0 Then 4: Call execv(“ls”,argValue[1]) 5: Call exit(<exitcode>) //return error code to OS 6: Else 7: Print “Waiting to child process to finish” 8: Call wait(null) // wait for child process to finish [End If] 9: Print “Sucess message” 10: Stop

Step 1: Start

2: Set pid:=fork() //creating child processed

3: If pid=0 Then

4: Call execv(“ls”,argValue[1])

5: Call exit(<exitcode>) //return error code to OS

6: Else

7: Print “Waiting to child process to finish”

8: Call wait(null) // wait for child process to finish [End If]

9: Print “Sucess message”

10: Stop

**Program**

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

#include <unistd.h> /\* for fork \*/ #include <sys/types.h> /\* for pid\_t \*/ #include <sys/wait.h> /\* for wait \*/

int main(int argc,char\*\* argv)

{

/\*Spawn a child to run the program.\*/ pid\_t pid=fork();

if (pid==0)

{ /\* child process \*/

execv("/bin/ls",argv);

exit(127); /\* only if execv fails \*/

}

else

{ /\* pid!=0; parent process \*/

printf("\nWaiting Child process to finish"); //waitpid(pid,0,0); /\* wait for child to exit \*/ wait(NULL);

}

printf("\nExiting main process\n"); return 0;

}

**Expected Input/Output:**

$os509 $cc multiprogramming.c

$os509 $./a.out –l

Total 28

-rwxrwxr-x 1 staff2 staff2 7317 aug14 09:22 a.out

-rwxrwxr-x 1 staff2 staff2 7317 aug14 09:16 multiprogramming

-rwxrwxr-x 1 staff2 staff2 361 aug16 09:20 multiprogramming.c

-rwxrwxr-x 1 staff2 staff2 359 aug14 09:16 multiprogramming.c

-rwxrwxr-x 1 staff2 staff2 1320 aug14 09:18 multiprogramming.o

Waiting until child process is finished End of main() process

**Viva-voce Questions**

1. What is symmetric multiprocessing?

2. List out the types in mainframe systems

3. What is file-server systems?

4. What is job scheduling?

**Experiment no:11 a. MEMORY MANAGEMENT WITH FIXED PARTITIONING TECHNIQUE (MFT)**

**AIM**: To implement and simulate the MFT algorithm.

**DESCRIPTION:**

In this the memory is divided in to parts and process is fit into it. The process which is best suited in to it is placed in the particular memory where it suits. We have to check memory partition. If it suits, its status should be changed.

**ALGORITHM:**

Step1: start the process.

Step2: Declare variables.

Step3: Enter total memory size ms. Step4: Allocate memory for os.

Ms=ms-os

Step5: Read the no partition to be divided n Partition size=ms/n.

Step6: Read the process no and process size.

Step 7: If proceaa size is less than partition size allot alse blocke the process. While allocating update memory wastage-external fragmentation.

if(pn[i]==pn[j])

f=1;

if(f==0){

if(ps[i]<=size)

{

extft=extft+size-ps[i]; avail[i]=1;

count++;

}

}

Step 8: Print the results

Step 9 :Stop the process.

**Program**

#include<stdio.h>

#include<conio.h>

main()

{

int ms,i,ps[20],n,size,p[20],s,intr=0;

clrscr();

printf("Enter size of memory:");

scanf("%d",&ms);

printf("Enter memory for OS:");

scanf("%d",&s); ms-=s;

printf("Enter no.of partitions to be divided:"); scanf("%d",&n);

size=ms/n;

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

{

printf("Enter process and process size"); scanf("%d%d",&p[i],&ps[i]); if(ps[i]<=size)

{

intr=intr+size-ps[i]; printf("process%dis allocated\n",p[i]);

}

else

printf("process%d is blocked",p[i]);

}

printf("total fragmentation is %d",intr); getch();

}

**Expected input/output:**

Enter total memory size : 50

Enter memory for OS :10

Enter no.of partitions to be divided:4

Enter size of page : 10

Enter size of page : 9

Enter size of page : 9

Enter size of page : 8

Internal Fragmentation is = 4

**Experiment no: 11b. MEMORY VARIABLE PARTIONING TYPE (MVT)**

**AIM**: To write a program to simulate the MVT algorithm

**ALGORITHM:**

Step1: start the process.

Step2: Declare variables.

Step3: Enter total memory size ms. Step4: Allocate memory for os.

Ms=ms-os

Step5: Read the no partition to be divided n Partition size=ms/n.

Step6: Read the process no and process size.

Step 7: If proceaa size is less than partition size allot alse blocke the process. While allocating update memory wastage-external fragmentation.

if(pn[i]==pn[j])

f=1;

if(f==0){

if(ps[i]<=size)

{

extft=extft+size-ps[i]; avail[i]=1;

count++;

}

}

Step 8: Print the results

Step 9 :Stop the process.

**Program**

#include<stdio.h>

#include<conio.h>

main()

{

int i,m,n,tot,s[20];

clrscr();

printf("Enter total memory size:");

scanf("%d",&tot);

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

scanf("%d",&n);

printf("Enter memory forOS:");

scanf("%d",&m);

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

{

printf("Enter size ofPage%d:",i+1);

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

}

tot=tot-m;

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

{

if(tot>=s[i])

{

printf("Allocate page%d\n",i+1);

tot=tot-s[i];

}

else

printf("process p%d is blocked\n",i+1);

}

printf("External Fragmentation is=%d",tot); getch();

**Expected Input/Output:**

Enter total memory size : 50

Enter memory for OS :10

Enter no.of partitions to be divided:4

Enter size of page : 10

Enter size of page : 9

Enter size of page : 9

Enter size of page : 8

Internal Fragmentation is = 4

**Viva-voce Questions**

1. What is logical address space and physical address space?

2. What is the main function of the memory-management unit?

3. Define swapping.

4. What do you mean by best fit?

5. What do you mean by first fit?

**Experiment no: 12**

**Question:** Simulate Bankers Algorithm for Dead Lock Avoidance

**Objective:** Creating Bankers program for deadlock avoiding by check the system is safe orunsafe state

**Algorithm for main() function: //for Banker's deadlock avoidance**

Step 1: Start

1. Declare \*\*Max, \*\*need, \*\*alloc, \*avail, \*completed, \*safeSequence
2. Declare p,r,i,j,count:=0
3. Input p //no.of processes
4. Initialize all Completed[] array to 0 //all are false

|  |  |  |  |
| --- | --- | --- | --- |
| 6: | Input r | //no.of resources |  |
| 7: | Set Max:=ProcessRequirement(Max,p,r) | | //allocate 2-D array and input |
| 8: | Set alloc:=ProcessRequirement(alloc,p,r) | | //allocate 2-D array and input |

1. Allocate Avail[] array dynamically
2. Allocate safeSequence[] array dynamically
3. Repeat Step 12 For I:=0 to r step 1

|  |  |
| --- | --- |
| 12: | Input avail[i] |
|  | [End For] |

1. Allocate need[][] 2-D array using for loop

|  |  |
| --- | --- |
| 14: | Repeat step 15,16 for I:=0 to p step 1 |
| 15: | Repeat step 16 for j:=0 to r step 1 |
| 16: | Set need[i][j]=Max[i][j]-alloc[i][j] |
|  | [End For] |
|  | [End For] |

1. Call printMatrix(Max,alloc,need,p,r) //for printing of 3-arrays

|  |  |
| --- | --- |
| 18: | Repeat steps 19-26 For I:=0 to p step 1 |
| 19: | Repeat step 20-26 for j:=0 to r step 1 |
| 20: | if(avail[j]>=need[i][j]) then |
| 21: | Set count++ |
|  | [End If] |
| 22: | If(count==r and completed[i]==0) Then |
| 23: | Set completed[i]=1 //set true |
| 24: | Set safeSequence[i]=i+1 |
| 25: | Set avail[j]+=alloc[i][j] |
| 26: | Set count:=0 |
|  | [End If] |
|  | [End for] |
|  | [End for] |

1. Repeat Step 28-30 for I:=0 to p step 1

|  |  |
| --- | --- |
| 28: | If(Completed[i]==0) Then |
| 29: | Print “System is unsafe state” |

30: Return 1

[End If]

[End For]

1. Print “System is in safe state”
2. Repeat step 33 for I:=0 to p step 1

33: Print safeSequence[i] [End For]

34:Stop

**Program**

#include <stdio.h> #include <stdlib.h> int main()

{

int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10]; int p, r, i, j, process, count;

count = 0;

printf("Enter the no of processes : "); scanf("%d", &p);

for(i = 0; i< p; i++) completed[i] = 0;

printf("\n\nEnter the no of resources : "); scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : "); for(i = 0; i < p; i++)

{

printf("\nFor process %d : ", i + 1); for(j = 0; j < r; j++)

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

}

printf("\n\nEnter the allocation for each process : "); for(i = 0; i < p; i++)

{

printf("\nFor process %d : ",i + 1); for(j = 0; j < r; j++)

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

}

printf("\n\nEnter the Available Resources : "); for(i = 0; i < r; i++)

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

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

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

need[i][j] = Max[i][j] - alloc[i][j];

do

{

printf("\n Max matrix:\tAllocation matrix:\n");

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

{

for( j = 0; j < r; j++) printf("%d ", Max[i][j]);

printf("\t\t");

for( j = 0; j < r; j++) printf("%d ", alloc[i][j]);

printf("\n");

}

process = -1;

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

{

if(completed[i] == 0)//if not completed

{

process = i ;

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

{

if(avail[j] < need[i][j])

{

process = -1; break;

}

}

}

if(process != -1) break;

}

if(process != -1)

{

printf("\nProcess %d runs to completion!", process + 1); safeSequence[count] = process + 1;

count++;

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

{

avail[j] += alloc[process][j]; alloc[process][j] = 0; Max[process][j] = 0;

completed[process] = 1;

}

}

}

while(count != p && process != -1);

if(count == p)

{

printf("\nThe system is in a safe state!!\n"); printf("Safe Sequence : < ");

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

printf("%d ", safeSequence[i]); printf(">\n");

}

else

printf("\nThe system is in an unsafe state!!");

}

**Expected Input/Output:**

Enter the no of processes : 5

Enter the no of resources : 3

Enter the Max Matrix for each process: For process 1 : 7 5 3

For process 2 : 3 2 2

For process 3 : 7 0 2

For process 4 : 2 2 2

For process 5 : 4 3 3

Enter the allocation for each process: For process 1 : 0 1

For process 2 : 2 0 0

For process 3 : 3 0 2

For process 4 : 2 1 1

For process 5 : 0 0 2

Enter the Available Resources : 3 3 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Max matrix: | | Allocation matrix: | | |
| 7 5 3 | 0 | 1 0 | | |
| 3 2 2 | 2 | 0 0 | | |
| 7 0 2 | 3 | 0 2 | | |
| 2 2 2 | 2 | 1 1 | | |
| 4 3 3 | 0 | 0 2 | | |
| Process 2 runs to completion! | | | | |
| Max matrix: | | Allocation matrix: | | |
| 7 5 3 | 0 | 1 0 | | |
| 0 0 0 | 0 | 0 0 | | |
| 7 0 2 | 3 | 0 2 | | |
| 2 2 2 | 2 | 1 1 | | |
| 4 3 3 | 0 | 0 2 | | |
| Process 3 runs to completion! | | | | |
| Max matrix: | | Allocation matrix: | | |
| 7 5 3 | 0 | 1 0 | | |
| 0 0 0 | 0 | 0 0 | | |
| 0 0 0 | 0 | 0 0 | | |
| 2 2 2 | 2 | 1 1 | | |
| 4 3 3 | 0 | 0 2 | | |
| Process 4 runs to completion! | | | | |
| Max matrix: | | Allocation matrix: | | |
| 7 5 3 | 0 | 1 0 | | |
| 0 0 0 | 0 | 0 0 | | |
| 0 0 0 | 0 0 0 | |
| 0 0 0 | 0 0 0 | |
| 4 3 3 | 0 0 2 | |
| Process 5 runs to completion!  The system is in a safe state!!  Safe Sequence : < 2 3 4 1 5 > | | | |  | |
|  |  | |
|  |  | |

**Viva-voce Questions**

1.What is meant by deadlock?

2. What are Necessary and Sufficient Deadlock Conditions?

3. .What is meant by Deadlock Avoidance?

**Experiment no: 13**

**Question:** Simulate Bankers Algorithm for Dead Lock Prevention

**Objective:** Creating Bankers program for deadlock prevention by check the system is safe orunsafe state

**Algorithm for main() function: //for Banker's deadlock prevention**

Step 1: Start

1. Declare \*\*Max, \*\*need, \*\*alloc, \*avail, \*completed, \*safeSequence
2. Declare p,r,i,j,count:=0
3. Input p //no.of processes
4. Initialize all Completed[] array to 0 //all are false

|  |  |  |  |
| --- | --- | --- | --- |
| 6: | Input r | //no.of resources |  |
| 7: | Set Max:=ProcessRequirement(Max,p,r) | | //allocate 2-D array and input |
| 8: | Set alloc:=ProcessRequirement(alloc,p,r) | | //allocate 2-D array and input |

1. Allocate Avail[] array dynamically
2. Allocate safeSequence[] array dynamically
3. Repeat Step 12 For I:=0 to r step 1

|  |  |
| --- | --- |
| 12: | Input avail[i] |
|  | [End For] |

1. Allocate need[][] 2-D array using for loop

|  |  |
| --- | --- |
| 14: | Repeat step 15,16 for I:=0 to p step 1 |
| 15: | Repeat step 16 for j:=0 to r step 1 |
| 16: | Set need[i][j]=Max[i][j]-alloc[i][j] |
|  | [End For] |
|  | [End For] |

1. Call printMatrix(Max,alloc,need,p,r) //for printing of 3-arrays

|  |  |
| --- | --- |
| 18: | Repeat steps 19-26 For I:=0 to p step 1 |
| 19: | Repeat step 20-26 for j:=0 to r step 1 |
| 20: | if(avail[j]>=need[i][j]) then |
| 21: | Set count++ |
|  | [End If] |
| 22: | If(count==r and completed[i]==0) Then |
| 23: | Set completed[i]=1 //set true |
| 24: | Set safeSequence[i]=i+1 |
| 25: | Set avail[j]+=alloc[i][j] |
| 26: | Set count:=0 |
|  | [End If] |

[End for] [End for]

1. Repeat Step 28-30 for I:=0 to p step 1

28: If(Completed[i]==0) Then

|  |  |
| --- | --- |
| 29: | Print “System is unsafe state” |
| 30: | Return 1 |
|  | [End If] |
|  | [End For] |

1. Print “System is in safe state”
2. Repeat step 33 for I:=0 to p step 1

33: Stop

**Program**

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

void main()

{

int allocated[15][15],max[15][15],need[15][15],avail[15],tres[15],work[15],flag[15]; int pno,rno,i,j,prc,count,t,total;

count=0;

clrscr();

printf("\n Enter number of process:"); scanf("%d",&pno);

printf("\n Enter number of resources:"); scanf("%d",&rno);

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

{

flag[i]=0;

}

printf("\n Enter total numbers of each resources:");

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

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

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

{

printf("\n for process %d:",i);

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

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

}

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

{

printf("\n for process %d:",i);

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

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

}

printf("\n available resources:\n"); for(j=1;j<= rno;j++)

{

avail[j]=0;

total=0;

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

{

total+=allocated[i][j];

}

avail[j]=tres[j]-total; work[j]=avail[j];

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

}

do

{

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

{

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

{

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

}

}

printf("\n Allocated matrix Max need");

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

{

printf("\n");

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

{

printf("%4d",allocated[i][j]);

}

printf("|");

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

{

printf("%4d",max[i][j]);

}

printf("|");

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

{

printf("%4d",need[i][j]);

}

}

prc=0;

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

{

if(flag[i]==0)

{

prc=i;

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

{

if(work[j]< need[i][j])

{

prc=0;

break;

}

}

}

if(prc!=0)

break;

}

if(prc!=0)

{

printf("\n Process %d completed",i); count++;

printf("\n Available matrix:"); for(j=1;j<= rno;j++)

{

work[j]+=allocated[prc][j];

allocated[prc][j]=0;

max[prc][j]=0;

flag[prc]=1;

printf(" %d",work[j]);

}

}

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

if(count==pno){

printf("\nThe system is in a safe state!!"); else

printf("\nThe system is in an unsafe state!!");

getch();

}

**Output**

Enter number of process:5

Enter number of resources:3

Enter total numbers of each resources:10 5 7

Enter Max resources for each process: for process 1:7 5 3

for process 2:3 2 2

for process 3:9 0 2

for process 4:2 2 2

for process 5:4 3 3

Enter allocated resources for each process: for process 1:0 1 0

for process 2:3 0 2

for process 3:3 0 2

for process 4:2 1 1

for process 5:0 0 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| available resources: | | | | | | | | | | | | | |  | |  | |  | | | |
| 2 | |  | 3 | | | |  | | 0 |  | | |  | |  | |  | | | |
| Allocated matrix | | | | | | | | | | Max | | |  | | | | need | | | |
| 0 | | 1 | 0| | | | | 7 | | 5 | 3| | | | 7 | | 4 | | 3 | | | |
| 3 | | 0 | 2| | | | | 3 | | 2 | 2| | | | 0 | | 2 | | 0 | | | |
| 3 | | 0 | 2| | | | | 9 | | 0 | 2| | | | 6 | | 0 | | 0 | | | |
| 2 | | 1 | 1| | | | | 2 | | 2 | 2| | | | 0 | | 1 | | 1 | | | |
| 0 | | 0 | 2| | | | | 4 | | 3 | 3| | | | 4 | | 3 | | 1 | | | |
| Process 2 completed | | | | | | | | | | | | | |  | | | |  |  | | |
| Available matrix: 5 3 2  Allocated matrix Max need 0 1 0| 7 5 3| 7 4 3  0 0 0| 0 0 0| 0 0 0 3 0 2| 9 0 2| 6 0 0 2 1 1| 2 2 2| 0 1 1 0 0 2| 4 3 3| 4 3 1 | | | | | | | | | | | | | |  | | | |  |  | | |
| Process 4 completed | | | | | | | | | |  |  |  | | | | | | | |
| Available matrix: | | | | | | | | 7 | | 4 | 3 |  | | | | | | | |
| Allocated matrix | | | | | | MaX | |  | | | | need | | | | | | | |
| 0 | 1 | | 0| | 7 | 5 | 3| | | 7 | |  | 4 | 3 | | | | | | | |
| 0 | 0 | | 0| | 0 | 0 | 0| | | 0 | |  | 0 | 0 | | | | | | | |
| 3 | 0 | | 2| | 9 | 0 | 2| | | 6 | |  | 0 | 0 | | | | | | | |
| 0 | 0 | | 0| | 0 | 0 | 0| | | 0 | |  | 0 | 0 | | | | | | | |
| 0 | 0 | | 2| | 4 | 3 | 3| | | 4 | |  | 3 | 1 | | | | | | | |
| Process 1 completed | | | | | | | | | |  |  |  | | | | | | | |
| Available matrix: | | | | | | | | 7 | | 5 | 3 |  | | | | | | | |
| Allocated matrix | | | | | | Max | |  | | | | need | | | | | | | |
| 0 | 0 | | 0| | 0 | 0 | 0| | | 0 | |  | 0 | 0 | | | | | | | |
| 0 | 0 | | 0| | 0 | 0 | 0| | | 0 | |  | 0 | 0 | | | | | | | |
| 3 | 0 | | 2| | 9 | 0 | 2| | | 6 | |  | 0 | 0 | | | | | | | |
| 0 | 0 | | 0| | 0 | 0 | 0| | | 0 | |  | 0 | 0 | | | | | | | |
| 0 | 0 | | 2| | 4 | 3 | 3| | | 4 | |  | 3 | 1 | | | | | | | |
| Process 3 completed | | | | | | | | | |  |  |  | | | | | | | |
| Available matrix: | | | | | | | | 10 | |  | 5 | 5 | | | | | | | |
| Allocated matrix | | | | | | Max | |  | | | | need | | | | | | | |
| 0 | 0 | | 0| | 0 | 0 | 0| | | 0 | |  | 0 | 0 | | | | | | | |
| 0 | 0 | | 0| | 0 | 0 | 0| | | 0 | |  | 0 | 0 | | | | | | | |
| 0 | 0 | | 0| | 0 | 0 | 0| | | 0 | |  | 0 | 0 | | | | | | | |
| 0 | 0 | | 0| | 0 | 0 | 0| | | 0 | |  | 0 | 0 | | | | | | | |
| 0 | 0 | | 2| | 4 | 3 | 3| | | 4 | |  | 3 | 1 | | | | | | | |
| Process 5 completed | | | | | | | | | |  |  |  | | | | | | | |
| Available matrix: | | | | | | | | 10 | |  | 5 | 7 | | | | | | | |

The system is in a safe state!!

**Viva-voce Questions**

1 .What is meant by preemptable and non\_preemptable resource?

2.What are strategies Dealing with Deadlock Problem?

3. What is meant by Deadlock Detection?

**Experiment no: 14(a)**

**Question:** Simulate the page replacement algorithm FIFO

**Algorithm**

Step 1: Start

Step 2: Declare frame, available ,count=0, n //n for number of pages

Step 3: Enter pages

Step 4: Enter page numbers

Step 5: Enter number of frames

Step 6: Print pages

Step 7: Print page numbers

Step 8: Print number of frames

Step 9: Print reference string, page frames

Step 10: count no of page faults

Step 11: print pagefaults

Step 12: stop

**Program**

include<stdio.h>

int main()

{

int i,j,n,a[50],frame[10],no,k,avail,count=0; printf("\n ENTER THE NUMBER OF PAGES:\n"); scanf("%d",&n);

printf("\n ENTER THE PAGE NUMBER :\n"); for(i=1;i<=n;i++)

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

printf("\n ENTER THE NUMBER OF FRAMES :"); scanf("%d",&no);

for(i=0;i<no;i++) frame[i]= -1;

j=0;

printf("\tref string\t page frames\n");

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

{

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

avail=0;

for(k=0;k<no;k++)

if(frame[k]==a[i])

avail=1;

if (avail==0)

{

frame[j]=a[i];

j=(j+1)%no;

count++;

for(k=0;k<no;k++)

printf("%d\t",frame[k]);

}

printf("\n");

}

printf("Page Fault Is %d",count); return 0;

}

**Output**

ENTER THE NUMBER OF PAGES: 20

ENTER THE PAGE NUMBER : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 ENTER THE NUMBER OF FRAMES :3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ref string |  | | page frames | |
| 7 | 7 |  | -1 | -1 |
| 0 | 7 |  | 0 | -1 |
| 1 | 7 |  | 0 | 1 |
| 2 | 2 |  | 0 | 1 |
| 0 |  |  |  |  |
| 3 | 2 |  | 3 | 1 |
| 0 | 2 |  | 3 | 0 |
| 4 | 4 |  | 3 | 0 |
| 2 | 4 |  | 2 | 0 |
| 3 | 4 |  | 2 | 3 |
| 0 | 0 |  | 2 | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 |  |  |  |
| 2 |  |  |  |
| 1 | 0 | 1 | 3 |
| 2 | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 |  |  |  |
| 7 | 7 | 1 | 2 |
| 0 | 7 | 0 | 2 |
| 1 | 7 | 0 | 1 |

Page Fault Is 15

**Viva-voce Questions**

1.What is the purpose of page replacement algorithms ?

2.What is meant by paging?

3.What is meant by fifo?

4.What is meant by second-chance algorithm

5.What is the advantage of the LRU algorithm

**Experiment no: 14(b)**

**Question:** Simulate the page replacement algorithm LRU

**Algorithm**

Step 1: Start

Step 2: Declare frame, available ,count=0, n //n for number of pages

Step 3: Enter pages

Step 4: Enter page numbers

Step 5: Enter number of frames

Step 6: Print pages

Step 7: Print page numbers

Step 8: Print number of frames

Step 9: Print reference string, page frames

Step 10: count no of page faults

Step 11: print pagefaults

Step 12: stop

**Program**

#include<stdio.h>

main()

{

int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20]; printf("Enter no of pages:");

scanf("%d",&n);

printf("Enter the reference string:"); for(i=0;i<n;i++) scanf("%d",&p[i]);

printf("Enter no of frames:"); scanf("%d",&f);

q[k]=p[k];

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

c++;

k++;

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

{

c1=0;

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

{

if(p[i]!=q[j])

c1++;

}

if(c1==f)

{

c++;

if(k<f)

{

q[k]=p[i];

k++;

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

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

printf("\n");

}

else

{

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

{

c2[r]=0; for(j=i-1;j<n;j--)

{

if(q[r]!=p[j])

c2[r]++; else break;

}

}

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

b[r]=c2[r];

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

{

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

{

if(b[r]<b[j])

{

t=b[r];

b[r]=b[j];

b[j]=t;

}

}

}

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

{

if(c2[r]==b[0])

q[r]=p[i];

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

}

printf("\n");

}

}

}

printf("\nThe no of page faults is %d",c);

}

**OUTPUT:**

Enter no of pages:10

Enter the reference string:7 5 9 4 3 7 9 6 2 1

Enter no of frames:3

|  |  |  |
| --- | --- | --- |
| 7 |  |  |
| 7 | 5 |  |
| 7 | 5 | 9 |
| 4 | 5 | 9 |
| 4 | 3 | 9 |
| 4 | 3 | 7 |
| 9 | 3 | 7 |
| 9 | 6 | 7 |
| 9 | 6 | 2 |

1 6 2

The no of page faults is 10

**Viva-voce Questions**

1. What is the purpose of page replacement algorithms ?

2.What is meant by paging?

3.What is meant by fifo?

4.What is meant by second-chance algorithm?

5.What is the advantage of the LRU algorithm

**Experiment no: 14(c)**

**Question:** Simulate the page replacement algorithm LFU

**Algorithm**

Step 1: Start

Step 2: Declare frame, available ,count=0, n //n for number of pages

Step 3: Enter pages

Step 4: Enter page numbers

Step 5: Enter number of frames

Step 6: Print pages

Step 7: Print page numbers

Step 8: Print number of frames

Step 9: Print reference string, page frames

Step 10: count no of page faults

Step 11: print pagefaults

Step 12: stop

**Program**

#include<stdio.h>

#include<conio.h>

main()

{

int fr[5],i,j,k,t[5],p=1,flag=0,page[25],psz,nf,t1,u[5]; clrscr();

printf("enter the number of frames:"); scanf("%d",&nf);

printf("\n enter the page size"); scanf("%d",&psz);

printf("\nenter the page sequence:"); for(i=1; i<=psz; i++)

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

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

fr[i]=-1;

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

{

if(full(fr,nf)==1)

break; else

{

flag=0;

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

{

if(page[i]==fr[j])

{

flag=1;

printf(" \t%d:\t",page[i]); break;

}

}

if(flag==0)

{

fr[p]=page[i];

printf(" \t%d:\t",page[i]); p++;

}

for(j=1; j<=nf; j++) printf(" %d ",fr[j]);

printf("\n");

}

}

p=0;

for(; i<=psz; i++)

{

flag=0;

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

{

if(page[i]==fr[j])

{

flag=1;

break;

}

}

if(flag==0)

{

p++;

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

{

for(k=i+1; k<=psz; k++)

{

if(fr[j]==page[k])

{

u[j]=k;

break;

}

else u[j]=21;

}

}

for(j=1; j<=nf; j++) t[j]=u[j];

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

{

for(k=j+1; k<=nf; k++)

{

if(t[j]<t[k])

{

t1=t[j];

t[j]=t[k];

t[k]=t1;

}

}

}

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

{

if(t[1]==u[j])

{

fr[j]=page[i];

u[j]=i;

}

}

printf("page fault\t");

}

else

printf(" \t"); printf("%d:\t",page[i]); for(j=1; j<=nf; j++)

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

}

printf("\ntotal page faults: %d",p+3); // getch();

}

int full(int a[],int n)

{

int k;

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

{

if(a[k]==-1) return 0;

}

return 1;

}

**OUTPUT:**

enter the number of frames:5 enter the page size2

enter the page sequence:1 2

1: 1 -1 -1 -1 -1

1. 1 2 -1 -1 -1

total page faults: 3

**Viva-voce Questions**

1.What is the purpose of page replacement algorithms ?

2.What is meant by paging?

3.What is meant by fifo?

.4.What is meant by second-chance algorithm?

5.What is the advantage of the LRU algorithm

**EXPERIMENT NO.15**

**FILE ALLOCATION STRATEGIES**

**A) SEQUENTIAL:**

The most common form of file structure is the sequential file in this type of file, a fixed format is used for records. All records (of the system) have the same length, consisting of the same number of fixed length fields in a particular order because the length and position of each field are known, only the values of fields need to be stored, the field name and length for each field are attributes of the file structure.

**Algorithm:**

Step 1: Start the program. Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations to each in sequential order a). Randomly select a location from available location

s1= random(100);

b). Check whether the required locations are free from the selected location. if(b[s1].flag==0)

{

for(j=s1;j<s1+p[i];j++)

{

if((b[j].flag)==0)

count++;

}

if(count==p[i])

break;

}

c). Allocate and set flag=1 to the allocated locations. for(s=s1;s<(s1+p[i]);s++)

{

k[i][j]=s;

j=j+1;

b[s].bno=s;

b[s].flag=1;

}

Step 5: Print the results fileno, lenth ,Blocks allocated.

Step 6: Stop the program

**Program**

#include<stdio.h>

#include<conio.h>

main()

{

int n,i,j,b[20],sb[20],t[20],x,c[20][20];

clrscr();

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

scanf("%d",&n);

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

{

printf("Enter no. of blocks occupied by file%d",i+1);

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

printf("Enter the starting block of file%d",i+1); scanf("%d",&sb[i]);

t[i]=sb[i];

for(j=0;j<b[i];j++)

}

printf("Filename\tStart block\tlength\n"); for(i=0;i<n;i++)

printf("%d\t %d \t%d\n",i+1,t[i],b[i]); printf("Enter file name:"); scanf("%d",&x);

printf("File name is:%d",x); printf("length is:%d",b[x-1]); printf("blocks occupied:"); for(i=0;i<b[x-1];i++) printf("%4d",c[x-1][i]); getch();

}

**Expected Input/Output:**

Enter no.of files: 2

Enter no. of blocks occupied by file1 4 Enter the starting block of file1 2

Enter no. of blocks occupied by file2 10

Enter the starting block of file2 5

Filename Start block length

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 4 |
| 2 | 5 | 10 |

Enter file name: rajesh

File name is:12803

length is:0 blocks occupied

**B) INDEXED:**

**Aim**:To implement allocation method using chained method

**Description:**

In the chained method file allocation table contains a field which points to starting block of memory. From it for each bloc a pointer is kept to next successive block.Hence, there is no external fragmentation.

**Algorithm:**

Step 1: Start the program. Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly q= random(100);

a). Check whether the selected location is free .

b). If the location is free allocate and set flag=1 to the allocated locations. q=random(100);

{

if(b[q].flag==0)

b[q].flag=1;

b[q].fno=j;

r[i][j]=q;

Step 5: Print the results fileno, lenth ,Blocks allocated.

Step 6: Stop the program

**Program**

#include<stdio.h>

#include<conio.h>

main()

{

int m[20],i,j,sb[20],s[20],b[20][20],x,n; clrscr();

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

scanf("%d",&n); for(i=0;i<n;i++)

{

printf("Enter starting block and size of file%d:",i+1); scanf("%d%d",&sb[i],&s[i]);

printf("Enter blocks occupied by file%d:",i+1);

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

printf("enter blocks of file%d:",i+1);

for(j=0;j<m[i];j++)

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

}

printf("\nFile\t index\tlength\n");

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

{

printf("%d\t%d\t%d\n",i+1,sb[i],m[i]); }printf("\nEnter file name:"); scanf("%d",&x);

printf("file name is:%d\n",x); i=x-1;

printf("Index is:%d",sb[i]); printf("Block occupied are:"); for(j=0;j<m[i];j++) printf("%3d",b[i][j]);

getch();

}

**Expected Input/Ouput:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Enter no. of files:2 | | | |  |
| Enter starting block and size of file1: 2 | | | | 5 |
| Enter blocks occupied by file1:10 | | | |  |
| enter blocks of file1:3 | | | |  |
| 2 5 4 6 7 2 6 4 7 |  |  |  |  |
| Enter starting block and size of file2: 3 | | | | 4 |
| Enter blocks occupied by file2:5 | | | |  |

enter blocks of file2: 2 3 4 5 6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| File | index | | | length |
| 1 | 2 | | 10 | | |
| 2 | 3 | 5 | |  |

Enter file name: venkat

file name is:12803 Index is:0

**C) LINKED:**

**AIM**: To implement linked file allocation technique.

**DESCRIPTION:**

In the chained method file allocation table contains a field which points to starting block of memory. From it for each bloc a pointer is kept to next successive block.Hence, there is no external fragmentation

**ALGORTHIM:**

Step 1: Start the program. Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly q= random(100);

a). Check whether the selected location is free .

b). If the location is free allocate and set flag=1 to the allocated locations. While allocating next location address to attach it to previous location

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

{

for(j=0;j<s[i];j++)

{

q=random(100);

if(b[q].flag==0)

b[q].flag=1;

b[q].fno=j;

r[i][j]=q;

if(j>0)

{

p=r[i][j-1]; b[p].next=q;

}

}

}

Step 5: Print the results fileno, lenth ,Blocks allocated.

Step 6: Stop the program

**Program**

#include<stdio.h>

#include<conio.h>

Struct file

{

Char fname[10];

Int start, size, block[10];

}f[20];

Main()

{

Int i,j,n;

Clrscr();

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

scanf("%d",&n);

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

{

printf("Enter file name:");

scanf("%s",&f[i].fname);

printf("Enter starting block:");

scanf("%d",&f[i].start);

f[i].block[0]=f[i].start;

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

scanf("%d",&f[i].size);

printf("Enter block numbers:");

for(j=1;j<=f[i].size;j++)

{

scanf("%d",&f[i].block[j]);

}

}

printf("File\tstart\tsize\tblock\n");

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

{

printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);

for(j=1;j<=f[i].size-1;j++)

printf("%d",f[i].block[j]);

printf("%d--->",f[i].block[j]);

printf("\n");

}

getch();

}

**Expected output**

Enter no. of files:2

Enter file name: venkat

Enter starting block:20

Enter no.of blocks:6

Enter block numbers: 4 12 15 45 32 25

Enter file name:rajesh

Enter starting block:12

Enter no.of blocks:5

Enter block numbers:6

5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4 |  |  | |  | | |
| 3 |  |  | |  | | |
| 2 |  |  | |  | | |
| File | | | start | | size | block | |
| venkat 20 | | | | | 6 | 4--->12--->15--->45--->32--->25 | |
| Rajesh | | | 12 | | 5 | 6--->5--->4--->3--->2 | |

**Viva-voce Questions**

1.Define seek time and latency time.

2.What are the allocation methods of a disk space?

3.What are the advantages of Contiguous allocation?

4. What are the drawbacks of contiguous allocation of disk space?

**Add on programs**

**Experiment#1**

**Question:** Loading executable programs into memory

**Objective:** Create program to load any executable programs like *“ls”* or *”clear”* or any *“a.out”*.

**Requirement Analysis:**

i) Create *main()* function with command line arguments ii) Use function *“execvp()”* to execute any command passed as arguments iii) If failed, print a message

**Algorithm for main(integer argcount, string argvalue[])**

Step 1: Start

2: Include required header files for spl. Functions

3: call execvp(argv[1],<address>argv[1])

4: If failed print “error message”

5: Stop

**Program**

/\* using execvp to execute the contents of argv \*/

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

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

{

if(argc!=2)

printf("exec failure, usage: <programfile><command>\n");

else

execvp(argv[1], &argv[1]);

return 0;

}

**Expected input/output:**

$./a.out clear

$./a.out ls

a.out loading.c loading.c~ loading.o

**Orginal input/output:**

****

**Viva-voce Questions**

1.What symmetric multiprocessing?

2. List out the types in mainframe systems

3. What is file-server systems?

4. What is job scheduling?

.

**Experiment-2**

**Question:** Execute System Call implementation- read(), write(), open () and close()

**Objective:**Creating a system call simulator for unix command **“cp <sourcefile> <destfile>”**

**Requirement Analysis:**

i) Create *main()* function with command line arguments

ii) Declare file descriptor to handle file addresses

iii) Declare *buffer[]* for character content of file

iv) If condition to check no.of arguments passed

v) Open files for *read()* and *write()* operations

vi) Required a loop for copying processed

**Algorithm for main(integer argcount, string argvalue[])**

Step 1: Start

2: Declare input\_fd,output\_fd //for file descriptors

3: Declare ret\_in,ret\_out for no.of character return by *read()* or *write()*

4: If argc!=3 Then

5: print “Usage: <programfile> <src> <dest>

6: return 1 //error code 1 [End if]

7: Set input\_fd=open(argv[1],O\_RDONLY)

8: If(input\_fd=-1) //read file opening failed

9: Print “error”

10: Return 2 //error code 2 [End if]

11:Set output\_fd=open(argv[2],O\_WRONLY|O\_CREAT, 0644)

12: If(output\_fd=-1)

13: Print “error”

14: Return 3 //error code 3

15: Repeat While ret\_in > 0 //reading character is available

16: write to file using *write()* functions

17: If ret\_out!=ret\_in then //if writing is failed

18: Print “error”

19: Return 4 //error code 4

20: Close all files using *close()* functions

21: Stop#include <stdio.h>

**Program**

#include <stdlib.h>

#include <fcntl.h>

#include <errno.h>

#include <sys/types.h>

#include <unistd.h>

#define BUF\_SIZE 8192

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

{

int input\_fd, output\_fd; /\* Input and output file descriptors \*/

ssize\_t ret\_in, ret\_out; /\* Number of bytes returned by read() and write() \*/

char buffer[BUF\_SIZE]; /\* Character buffer \*/

/\* Are src and dest file name arguments missing \*/

if(argc != 3)

{

printf ("Usage: <programfile><sourcefile><destfile>\n");

return 1;

}

\* Create input file descriptor \*/ input\_fd = open (argv [1], O\_RDONLY);

if (input\_fd == -1)

{

perror ("open");

return 2;

}

/\* Create output file descriptor \*/

output\_fd = open(argv[2], O\_WRONLY | O\_CREAT, 0644);

if(output\_fd == -1)

{

perror("open"); return 3;

}/\* Copy process \*/

while((ret\_in = read (input\_fd, &buffer, BUF\_SIZE)) > 0)

{

ret\_out = write (output\_fd, &buffer, (ssize\_t) ret\_in);

if(ret\_out != ret\_in)

{

/\* Write error \*/ perror("write"); return 4;

}

}

/\* Close file descriptors \*/ close (input\_fd);

close (output\_fd);

printf("File Copied successfully\n"); return (EXIT\_SUCCESS);

}

**Expected Input/Output:**

$cc execute.c

$cat o.txt

My name is uma

$./a.out o.text n.text

File copy is successful

$cat n.txt

My name is uma

if permission denied

$sudo chmod 777 n.txt

**Orginal Input/Output:**



**Viva-voce Questions**

1.What symmetric multiprocessing?

2. List out the types in mainframe systems

3. What is file-server systems?

4.What is job scheduling?

**Experiment-3:**

**Write a shell script that accepts a file name, starting and ending numbers as arguments and displays all the lines between the given line numbers.**

**Algorithm**

Step 1: enter file name

Step 2: read file

Step 3: take first argument as starting line

Step 4: take second argument as last line

Step 5: first Line displays

Step 6: displays all lines in a given file

Step 7: last Line displays

**Program**

lastline=` wc -l <$1 `

if [ $2 -lt $lastline -a $3 -le $lastline ]

then

nline=` expr $3 - $2 + 1 `

echo ` head -$3 $1|tail -$nline `

else

echo "Invalid range specification"

fi

**Expected output**

Enter file name

Sample

First line displays

#!/bin/bash

Displays all lines in a given file

#/bin/bash

echo‖hello world‖

echo‖knowledge is power‖

**Viva questions:**

1.What is the differenc between fopen() and open().

2.What is file pointer?

3.What is file descriptor.

4.What are the modes used in open().

5.What is FILE?

**Experiment-4:**

**Write a shell script that displays a list of all files in the current directory to which the user has read, write and execute permissions.**

**Algorithm**

Step 1: start

Step 2: "List of Files which have Read, Write and Execute Permissions in Current Directory"

Step 3:stop

**Program**

for i in `ls`

do

if [ -r $i -a -w $i -a -x $i ]

then

echo $i

fi

Done

**Expected output**

list of files which have read,write and execute permissions in current directory desktop

Downloads

Music

pictures

**Viva questions:**

1.What is the differenc between fopen() and open().

2.What is file pointer?

3.What is file descriptor.

4.What are the modes used in open().

5.What is FILE?