**Department of Computer Science and Engineering**

**LAB MANUAL**

**R16**

**Operating Systems & Linux Programming Lab**

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



**KKR & KSR INSTITUTE OF TECHNOLOGY AND SCIENCES**

**Vinjanampadu, Guntur District- 522017 (A. P.)**

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

**Department of Computer Science Engineering**

**LAB MANUAL**

**R16**

**OPERATING SYSTEMS & LINUX LAB**

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

|  |  |  |
| --- | --- | --- |
| **INDEX** | | |
| **S.No** | **Contents** | **Page. No** |
| 1 | Institute Vision & Mission | 4 |
| 2 | Department Vision & Mission | 4 |
| 3 | Program Educational Objectives & Program Outcomes | 5-7 |
| 4 | Program Specific Outcomes | 7 |
| 5 | Syllabus | 8-11 |
| 6 | Course Outcomes | 12 |
| 7 | List of Experiments | 13-14 |
| 8 | Course Outcomes of associated course | 15 |
| 9 | Experiment Mapping with Course Outcomes | 16 |
|  | **Experiments** |  |
| 10 | Simulation of the CPU scheduling algorithms | 17-29 |
| 11 | Implementation of fork (), wait (), exec() and exit (), System calls | 30-32 |
| 12 | Simulation of MVT and MFT | 33-36 |
| 13 | Simulation of Bankers Algorithm for Dead Lock Avoidance | 37-41 |
| 14 | Simulation of Bankers Algorithm for Dead Lock Prevention | 42-47 |
| 15 | Simulation of page replacement algorithms | 48-57 |
| 16 | Simulation of File allocation strategies | 58-66 |
| 17 | Additional programs | 67-71 |
| 18 | Study of Unix/Linux general purpose utility command list, vi editor, file system, .bashrc, /etc/bashrc and Environment variables | 73-88 |
| 19 | C program that makes a copy of a file using standard I/O, and system calls | 89-91 |
| 20 | C program to emulate the UNIX ls –l command | 92-93 |
| 21 | C program illustrates how to execute two commands concurrently with a command pipe | 94-95 |
| 22 | C program of two processes communication using shared memory | 96-98 |
| 23 | C program to simulate producer and consumer problem using semaphores | 99-101 |
| 24 | C program to create a thread using pthreads library and let it run its function | 102-103 |
| 25 | C program to illustrate concurrent execution of threads using pthreads library | 104-106 |
| 26 | Additional Experiments | 107-109 |

**INSTITUTE VISION**

|  |
| --- |
| To become a knowledge centre for technical education and also to become the top engineering college in the sunrise state of Andhra Pradesh.  **INSTITUTE MISSION** |
| 1. To incorporate benchmarked teaching and learning pedagogies in curriculum. 2. To ensure all round development of students through judicious blend of curricular, co-curricular and extra-curricular activities. 3. To support cross-cultural exchange of knowledge between industry and academy. 4. 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**

* Providing a strong theoretical and practical education in a congenial environment.
* Providing additional skills and training to meet the current needs of the industry.
* Inculcating ethical values to meet the challenges of life with courage and confidence.

**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 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 SPECIFIC OUTCOMES:**

**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

****

**JAWARHARLAL NEHRU TECHNOLOGICAL UNIVERSITY**

**KAKINADA-533003 , A.P. INDIA**

**III Year – I SEMESTER T P C**

**0 3 2**

**Operating Systems & Linux Programming Laboratory**

**Operating Systems**

1. Simulate the following CPU scheduling algorithms

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

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

3. Simulate the following

Multiprogramming with a fixed number of tasks (MFT)

Multiprogramming with a variable number of tasks (MVT)

4. Simulate Bankers Algorithm for Dead Lock Avoidance

5. Simulate Bankers Algorithm for Dead Lock Prevention.

6. Simulate the following page replacement algorithms.

a) FIFO b) LRU c) LFU

7. Simulate the following File allocation strategies

a) Sequenced b) Indexed c) Linked

**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 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. 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.

**Associated Course Syllabus:**

**III Year – I Semester L T P C**

**4 0 0 3**

**OPERATING SYSTEMS**

**OBJECTIVES:**

• Study the basic concepts and functions of operating systems.

• Understand the structure and functions of OS.

• Learn about Processes, Threads and Scheduling algorithms.

• Understand the principles of concurrency and Deadlocks.

• Learn various memory management schemes.

• Study I/O management and File systems.

• Learn the basics of Linux system and perform administrative tasks on Linux Servers.

**UNIT I**

**Introduction to Operating System Concept:** Types of operating systems, operating systems

concepts, operating systems services, Introduction to System call, System call types.

**UNIT-II:**

**Process Management** – Process concept, The process, Process State Diagram , Process control

block, Process Scheduling- Scheduling Queues, Schedulers, Operations on Processes,

Interprocess Communication, Threading Issues, Scheduling-Basic Concepts, Scheduling Criteria,

Scheduling Algorithms.

**UNIT-III:**

**Memory Management:** Swapping, Contiguous Memory Allocation, Paging, structure of the

Page Table, Segmentation

**Virtual Memory Management:**

Virtual Memory, Demand Paging, Page-Replacement Algorithms, Thrashing

**UNIT-IV:**

**Concurrency:** Process Synchronization, The Critical- Section Problem, Synchronization

Hardware, Semaphores, Classic Problems of Synchronization, Monitors, Synchronization

Examples

**Principles of deadlock** – System Model, Deadlock Characterization, Deadlock Prevention,

Detection and Avoidance, Recovery form Deadlock

**UNIT-V:**

**File system Interface-** the concept of a file, Access Methods, Directory structure, File system

Mounting, file sharing, protection.

**File System implementation-** File system structure, allocation methods, free-space management

**Mass-storage structure** overview of Mass-storage structure, Disk scheduling, Device drivers,

**UNIT VI:**

**Linux System**: Components of LINUX, Interprocess Communication, Synchronization,

Interrupt, Exception and System Call.

**Android Software Platform**: Android Architecture, Operating System Services, Android

Runtime Application Development, Application Structure, Application Process management

**OUTCOMES:**

• Design various Scheduling algorithms.

• Apply the principles of concurrency.

• Design deadlock, prevention and avoidance algorithms.

• Compare and contrast various memory management schemes.

• Design and Implement a prototype file systems.

• Perform administrative tasks on Linux Servers

• Introduction to Android Operating System Internals

**TEXT BOOK:**

1. Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin and Greg Gagne

9th Edition, John Wiley and Sons Inc., 2012.

2.Operating Systems – Internals and Design Principles, William Stallings, 7th Edition,

Prentice Hall, 2011.

3. Operating Systems-S Halder, Alex A Aravind Pearson Education Second Edition 2016 .

**REFERENCES:**

1. Modern Operating Systems, Andrew S. Tanenbaum, Second Edition, Addison Wesley,

2001.

2. Operating Systems: A Design-Oriented Approach, Charles Crowley, Tata Mc Graw Hill

Education”, 1996.

3. Operating Systems: A Concept-Based Approach, D M Dhamdhere, Second Edition, Tata

Mc Graw-Hill Education, 2007.

**III Year – I Semester L T P C**

**4 0 0 3**

**UNIX PROGRAMMING**

**OBJECTIVES:**

• Written technical communication and effective use of concepts and terminology.

• Facility with UNIX command syntax and semantics.

• Ability to read and understand specifications, scripts and programs.

• Individual capability in problem solving using the tools presented within the class.

Students will demonstrate a mastery of the course materials and concepts within in class

discussions.

**UNIT-I**

Introduction to unix-Brief History-What is Unix-Unix Components-Using Unix-Commands in Unix-Some Basic Commands-Command Substitution-Giving Multiple Commands.

**UNIT-II**

The File system –The Basics of Files-What’s in a File-Directories and File Names-Permissions-I Nodes-The Directory Hierarchy, File Attributes and Permissions-The File Command knowing the File Type-The Chmod Command Changing File Permissions-The Chown Command Changing the Owner of a File-The Chgrp Command Changing the Group of a File.

**UNIT-III**

Using the Shell-Command Line Structure-Met characters-Creating New Commands-Command Arguments and Parameters-Program Output as Arguments-Shell Variables- -More on I/O Redirection-Looping in Shell Programs.

**UNIT-IV**

Filters-The Grep Family-Other Filters-The Stream Editor Sed-The AWK Pattern Scanning and processing Language-Good Files and Good Filters.

**UNIT-V**

Shell Programming-Shell Variables-The Export Command-The Profile File a Script Run During Starting-The First Shell Script-The read Command-Positional parameters-The $? Variable knowing the exit Status-More about the Set Command-The Exit Command-Branching Control Structures-Loop Control Structures-The Continue and Break Statement-The Expr Command: Performing Integer Arithmetic-Real Arithmetic in Shell Programs-The here Document(<<)-The Sleep Command-Debugging Scripts-The Script Command-The Eval Command-The Exec Command.

**UNIT-VI**

The Process-The Meaning-Parent and Child Processes-Types of Processes-More about

Foreground and Background processes-Internal and External Commands-Process Creation-The Trap Command-The Stty Command-The Kill Command-Job Control.

**OUTCOMES:**

• Documentation will demonstrate good organization and readability.

• File processing projects will require data organization, problem solving and research.

• Scripts and programs will demonstrate simple effective user interfaces.

• Scripts and programs will demonstrate effective use of structured programming.

• Scripts and programs will be accompanied by printed output demonstrating completion of

a test plan.

• Testing will demonstrate both black and glass box testing strategies.

• Project work will involve group participation.

**TEXT BOOKS:**

1. The Unix programming Environment by Brain W. Kernighan & Rob Pike, Pearson.

2. Introduction to Unix Shell Programming by M.G.Venkateshmurthy, Pearson.

**REFERENCE BOOKS:**

1. Unix and shell programmingby B.M. Harwani, OXFORD university press.

**Course Outcomes: (Operating Systems)**

|  |
| --- |
| 1. Memorize the concepts of operating systems such as types, services and system calls. |
| 1. Implement the process concepts on different processes by using scheduling algorithms. |
| 1. Analyze different memory management techniques and gives optimal solutions. |
| 1. Identify the principals of concurrency and apply algorithms for deadlock problems. |
| 1. Implement file system concepts and storage structures. |
| 1. Recall the basic concepts of Linux system and Android system platform. |

**Course Outcomes: (Unix Programming)**

|  |
| --- |
| 1. Describe the architecture and features of UNIX Operating System and distinguish it from other Operating System |
| 1. To gain an understanding of important aspects related to the SHELL and the process |
| 1. Write Regular expressions for pattern matching and apply them to various filters for a specific task |
| 1. Understand the concepts of process, threads, and file structure. |

**List of Experiments**

|  |  |  |
| --- | --- | --- |
| **WEEK NO.** | **SUB**  **EXP.** | **NAME OF THE EXPERIMENT** |
| **OPERATING SYSTEMS LAB PROGRAMS** | | |
| 1 | a | Simulate the following CPU scheduling algorithms  Round Robin |
| b | SJF |
| c | FCFS |
| d | Priority |
| 2 |  | Implementation of Fork(), Wait(), Exec() and Exit() System calls |
| 3 | a | Simulate MVT and MFT  Multiprogramming Variable Tasking |
| b | Multiprogramming Fixed Tasking |
| 4 |  | Simulate Bankers Algorithm for Dead Lock Avoidance |
| 5 |  | Simulate Bankers Algorithm for Dead Lock Prevention. |
| 6 | a | Simulate all page replacement algorithms.  FIFO |
| b | LRU |
| c | LFU |
| 7 | a | Simulate the following File allocation strategies  Sequential |
| b | Indexed |
| c | Linked |
| **LINUX PROGRAMMING LAB PROGRAMS** | | |
| 1 | a | Study of Unix/Linux general purpose utility command  list |
| b | Study of vi editor |
| **WEEK NO.** | **SUB**  **EXP.** | **NAME OF THE EXPERIMENT** |
|  | 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. |
| 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 pthreads library  and let it run its function |
| 8 |  | Write a C program to illustrate concurrent execution of  threads using pthreads library. |

**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 |

**Course Outcomes of Associated Course:**

Two courses associated with this laboratory. They are Operating systems and Unix Programming. Here, the list of outcomes is given below:

**Course Outcomes of Operating Systems:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Course Code** | **Course Outcome** | **Level** | **Bloom’s Taxonomy** |
| C315.1 | Memorize the concepts of operating systems such as types, services and system calls. | L1 | Remember |
| C315.2 | Implement the process concepts on different processes by using scheduling algorithms. | L3 | Apply |
| C315.3 | Analyze different memory management techniques and gives optimal solutions. | L4 | Analyze |
| C315.4 | Identify the principals of concurrency and apply algorithms for deadlock problems. | L2 & L3 | Understand & Apply |
| C315.5 | Implement file system concepts and storage structures. | L3 | Apply |
| C315.6 | Recall the basic concepts of Linux system and Android system platform. | L1 | Remember |

**Course Outcomes of Unix Programming:**

|  |  |  |
| --- | --- | --- |
| **Course code** | **Course outcome** | **Taxonomy Level** |
| C312.1 | Describe the architecture and features of UNIX Operating System and distinguish it from other Operating System | L1:Remembering |
| C312.2 | To gain an understanding of important aspects related to the SHELL and the process | L3:Applying |
| C312.4 | Understand the concepts of process, threads, and file structure. | L2:Understanding |

**Experiment Mapping with Course Outcomes:**

**Operating Systems Experiments:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Experiment | | C315.1 | C315.2 | C315.3 | C315.4 | C315.5 | C315.6 |
| Ex 1 | a |  | 3 |  |  |  |  |
| b |  | 3 |  |  |  |  |
| c |  | 3 |  |  |  |  |
| d |  | 3 |  |  |  |  |
| Ex 2 | | 3 |  |  |  |  | 2 |
| Ex 3 | a |  |  | 3 |  |  |  |
| b |  |  | 3 |  |  |  |
| Ex 4 | |  |  |  | 3 |  |  |
| Ex 5 | |  |  |  | 3 |  |  |
| Ex 6 | a |  |  | 3 |  |  |  |
| b |  |  | 3 |  |  |  |
| c |  |  | 3 |  |  |  |
| Ex 7 |  |  |  |  |  | 3 |  |
|  |  |  |  |  | 3 |  |
|  |  |  |  |  | 3 |  |

**Linux Programming Experiments:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Experiment | | C312.1 | C312.2 | C312.4 |
| Ex 1 | a | 2 |  |  |
| b | 2 |  |  |
| c | 2 |  |  |
| d | 2 |  |  |
| e | 2 |  |  |
| Ex 2 | | 3 |  |  |
| Ex 3 | |  | 3 |  |
| Ex 4 | |  | 3 |  |
| Ex 5 | |  |  | 3 |
| Ex 6 | |  |  | 3 |
| Ex 7 | |  |  | 3 |
| Ex 8 | |  |  | 3 |

**Experiment 1(a):**

**Aim:** Simulate CPU Scheduling algorithm***Round Robin***

**Objective:** We create*n*no.of processes, each process one has*burst time*and*arrival time.*

Based on input of *time quantum,* processes are scheduled.

**Algorithm**

Step 1:Start

2: Declare exeCount,i,rounds:=1,finish=0,totalwaiting=0 3:Repeat steps 4 to 10 While(not isAllFinished(p,n))

4: Print “After Round #”,rounds++

5: Repeat steps 6 to 10 For I:=0 to n step 1

1. Repeat steps 7 to 9 For exeCount:=1 to exeCount<=q AND p[i].remainingTime!=0
2. Set p[i].remainingTime--
3. Set finish:=finish+1
4. if(p[i].remainingTime=0) Then → Set p[i].finishTime:=finish

[End For – exeCount]

10: Print “Remaining time ',p[i].remainingTime

[End For – I]

[End While]

11: Repeat steps 12,13 For I:=0 to n step 1

1. Print p[i].finishTime, p[i].finishTime-p[i].burstTime //this is waiting time

subtraction

1. Set totalwaiting+=p[i].finishTime-p[i].burstTime //adding total waiting

[End For]

14: Print totalwaiting, totalwaiting/n //n for no.of processes

15: Stop

**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

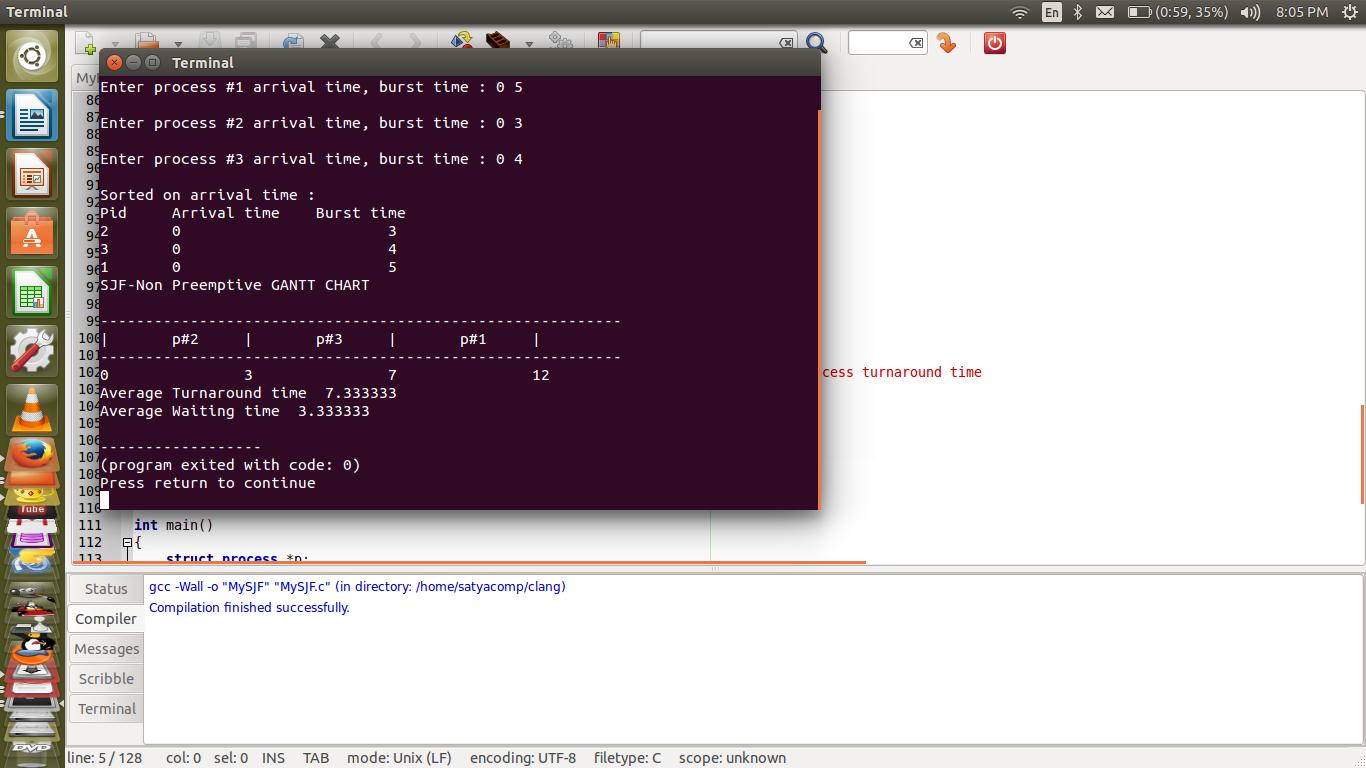
enter the processing time : 1

|  |  |  |
| --- | --- | --- |
| **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

**Original input/output:**



**Viva-voice Questions**

**1. Define CPU scheduling.**

CPU scheduling is the process of switching the CPU among various processes. CPU scheduling is the basis of multi programmed operating systems. By switching the CPU among processes, the operating system can make the computer more productive.

1. **What is a Dispatcher?**

The dispatcher is the module that gives control of the CPU to theprocess selected by the short-term scheduler. This function involves: • Switching context • Switching to user mode • Jumping to the proper location in the user program to restart that program.

1. **What is turnaround time?**

Turnaround time is the interval from the time of submission tothe time of completion of a process. It is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

1. **Define dispatch latency?**

The time taken for the dispatcher to stop one process and start another running process is known as dispatch latency

**5. What is job scheduling?**

If several jobs are ready to be brought in to memory, and if there is not enough room for all of them, then the system must choose among them. Making this decision is job scheduling

**Experiment 1(b):**

**Aim:** Simulate CPU Scheduling algorithm***Shortest Job First***

**Objective:** Create*n*no. of processes, each process has*arrival time*and*burst time.*Sort on*arrival time*then by*burst time.* Process least burst time first and highest burst time next.

**Algorithm**

|  |  |  |
| --- | --- | --- |
| Step 1: Start |  |  |
| 2: Declare | wt:=0,tt:=0,i | //wt-for waiting time, tt-turnaround time, i -index repeating |
| 3: Repeat | step 4 For I:=0 to n step 1 | |
| 4: Print p[i].pid | | //printing each process id in Gantt chart |
| [End for] |  |  |
| 5: Repeat Steps 6 to 10 | | For i=0 to n step 1 |
| 6: print wt | //priting waiting time | |

1. Set p[i].watitingtime:=wt
2. Repeat Steps 9,10 while p[i].RemainingTime>0 //each process remainingtime is initially burst time
3. Set p[i].RemainingTime:=p[i].RemaminingTime-1

10: Set Wt:=Wt+1 //increase waiting time by 1

[End While]

[End For]

11: print Wt //last process waiting time

1. Set p[i].waitingTime:=Wt
2. Repeat steps 14 to 16 For I:=0 to n step 1
3. set p[i].turnAroundTime:=(p[i].waitingTime+p[i].burstTime)-p[i].ArrivalTime //Each process turnaround time

15: Set tt:=tt+p[i].TurnaroundTime //calculating total turnaround time

1. Set wt:=wt+p[i].waitingTime //calculating total waiting time [End For]
2. Print “Avg turnaround time : “,tt/n
3. Print “Avg waiting time : “,wt/n
4. Stop

**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

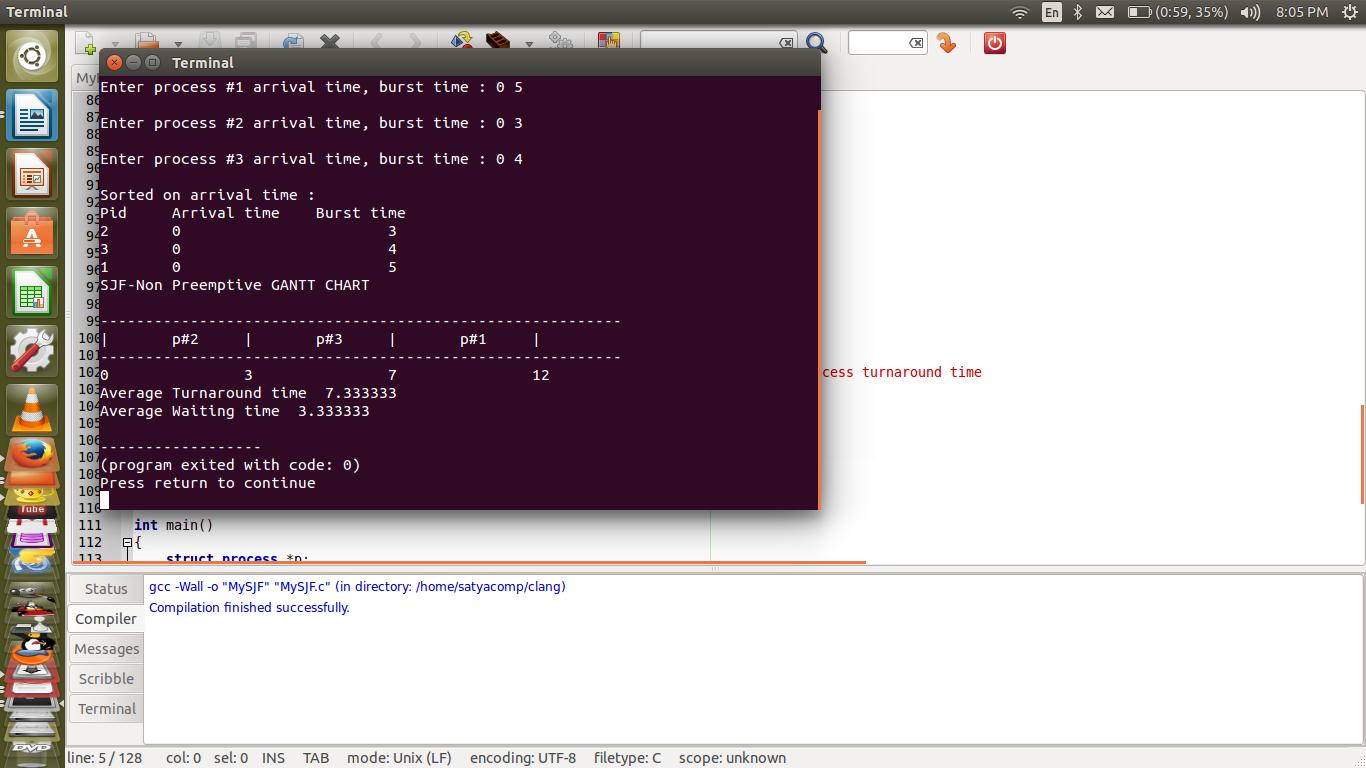
enter process time: 5

enter process5 name: eee enter process time: 1

|  |  |  |
| --- | --- | --- |
| p\_name | P\_time | w\_time |
| eee | 1 | 0 |
| ccc | 2 | 1 |
| bbb | 3 | 3 |
| aaa | 4 | 6 |
| ddd | 5 | 10 |

total waiting time=20 avg waiting time=4.00

**Original input/output:**



**Viva-voice Questions**

**1. What is symmetric multiprocessing?**

Each processor runs an identical copy of the operating system, and these copies communicate with one another as needed.

**2. List out the types in mainframe systems**

1. Batch system
2. Multiprogrammed systems
3. Time-sharing system

**3. What are file-server systems?**

File-server system provides a file system interface where clients can create, update, read, and delete files

**4. What is job scheduling?**

If several jobs are ready to be brought in to memory, and if there is not enough room for all of them, then the system must choose among them. Making this decision is job scheduling

**Experiment 1(c):**

**Aim:** Simulate CPU Scheduling algorithm***First Come First Serve***

**Objective:** Create*n*no.of processes, each process has*arrivaltime*and*bursttime.*Sort on*arrivaltime.* First entered process is processed first.

**Algorithm**

Step 1: Start

1. Declare wt:=0, I //Waiting Time and i for index
2. Repeat step 4 for I:=0 to n step 1
3. Print p[i].pid //printing each process in Gantt chart [End for]
4. Repeat steps 6 to 10 For I:=0 to n step 1

|  |  |  |
| --- | --- | --- |
| 6: | Print wt | // process each waiting time |
| 7: | Set p[i].waitingTime:=wt-p[i].arrivalTime //update each process waiting | |

time

1. Repeat Step 9,10 while p[i].burstTime>0 loop
2. p[i].bursttime-- //process each bursttime until reaches to 0
3. Set wt:=wt+1

[End While]

[End For]

11: Print wt //last proccess waiting time

12: Set p[i].waitingTime:=wt

13:Repeat step 14 for I:=0 to n step 1

1. Set wt:=wt+p[i].waitingTime //sum of all waiting times [End for]
2. print “Average waiting time : “,wt/n
3. Stop

**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

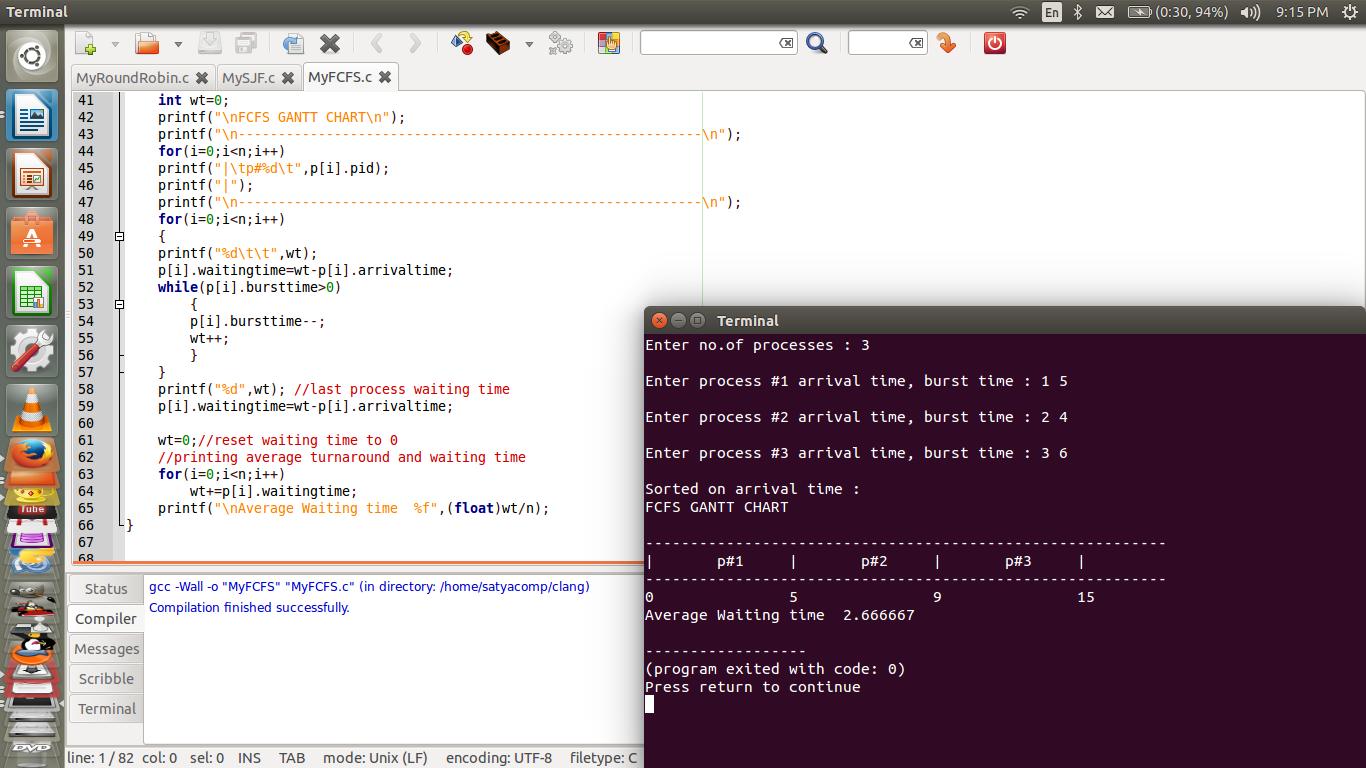
enter process5 name: eee

enter process time: 1

|  |  |  |
| --- | --- | --- |
| p\_name | P\_time | W\_time |
| aaa | 4 | 0 |
| bbb | 3 | 4 |
| ccc | 2 | 7 |
| ddd | 5 | 9 |
| eee | 1 | 14 |

total waiting time=34 avg waiting time=6.80

**Original output:**



**Viva-voice Questions**

**1. Define CPU scheduling.**

CPU scheduling is the process of switching the CPU among various processes. CPU scheduling is the basis of multi programmed operating systems. By switching the CPU among processes, the operating system can make the computer more productive.

1. **What is a Dispatcher?** The dispatcher is the module that gives control of the CPU to theprocess selected by the short-term scheduler. This function involves: • Switching context • Switching to user mode • Jumping to the proper location in the user program to restart that program.
2. **What is turnaround time?** Turnaround time is the interval from the time of submission tothe time of completion of a process. It is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.
3. **Define dispatch latency?**

The time taken for the dispatcher to stop one process and start another running process is known as dispatch latency.

**Experiment 1(d):**

**Aim:** Simulate CPU Scheduling algorithm***Priority Scheduling***

**Objective:** Create*n*no. of processes, each process has*priority no*and*burst time.*Sort on*priorityno.* Lowest priority number indicates that is processes first. If multiple process havesame priority number, then FCFS is used.

Algorithm for Priority Scheduling (Process \*p,integer n)

Step 1: Start

1. Declare wt:=0, I //Waiting Time and i for index
2. Repeat step 4 for I:=0 to n step 1
3. Print p[i].pid //printing each process in Gantt chart [End for]
4. Repeat steps 6 to 10 For I:=0 to n step 1

|  |  |  |
| --- | --- | --- |
| 6: | Print wt | // process each waiting time |
| 7: | Set p[i].waitingTime:=wt-p[i].arrivalTime //update each process waiting | |

time

1. Repeat Step 9,10 while p[i].burstTime>0 loop
2. p[i].bursttime-- //process each bursttime until reaches to 0
3. Set wt:=wt+1

[End While]

[End For]

11: Print wt //last proccess waiting time

12: Set p[i].waitingTime:=wt

13:Repeat step 14 for I:=0 to n step 1

1. Set wt:=wt+p[i].waitingTime //sum of all waiting times [End for]
2. print “Average waiting time : “,wt/n
3. Stop

**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

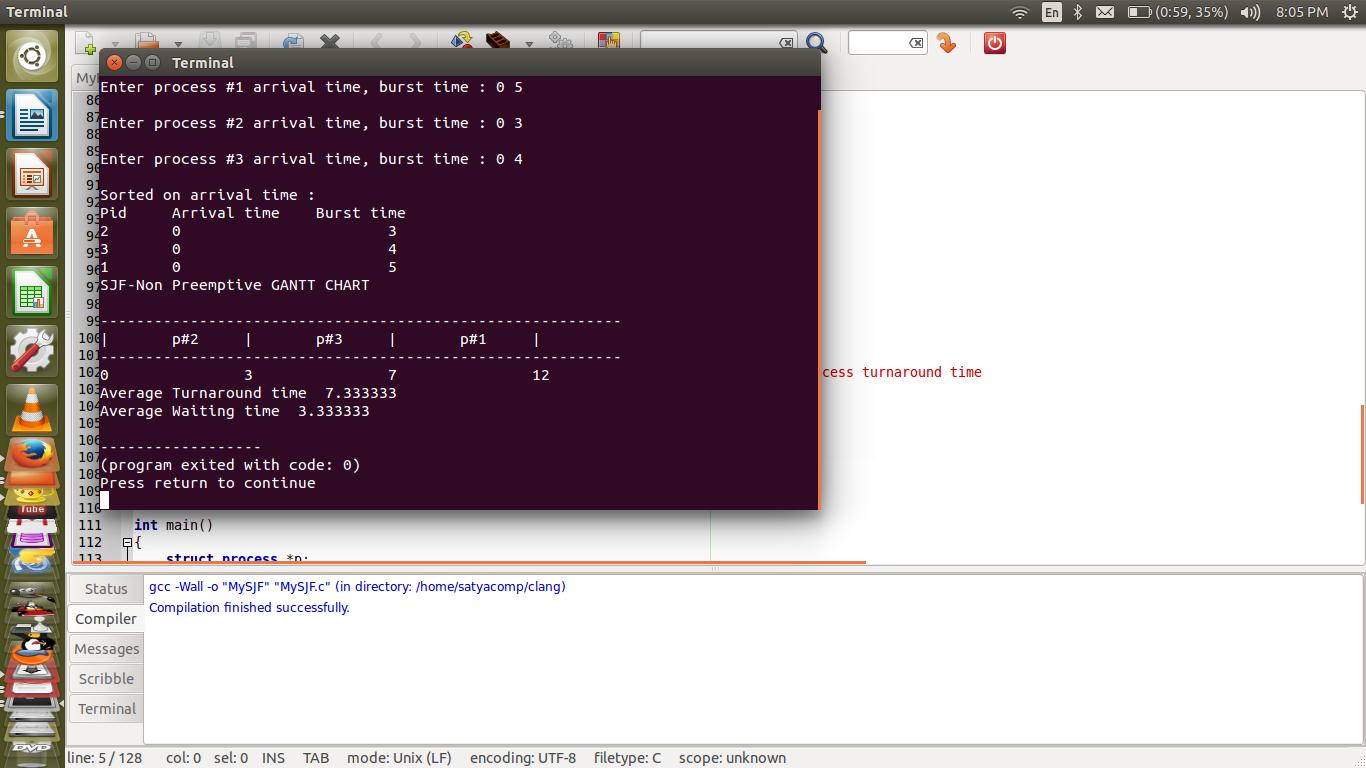
enter process4 name: ddd enter process time: 5 enter priority:2

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

**Output:**



**Viva-voice Questions**

**1. Define CPU scheduling.**

CPU scheduling is the process of switching the CPU among various processes. CPU scheduling is the basis of multi programmed operating systems. By switching the CPU among processes, the operating system can make the computer more productive.

**2. What is a Dispatcher?**

The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves: • Switching context • Switching to user mode Jumping to the proper location in the user program to restart that program.

**3. What is turnaround time?**

Turnaround time is the interval from the time of submission to the time of completion of a process. It is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

**4. Define dispatch latency?**

The time taken for the dispatcher to stop one process and start another running process is known as dispatch latency.

**Experiment 2:**

**Aim**: 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 |

1. If pid=0 Then
2. Call execv(“ls”,argValue[1])
3. Call exit(<exitcode>) //return error code to OS
4. Else
5. Print “Waiting to child process to finish”
6. Call wait(null) // wait for child process to finish [End If]
7. Print “Sucess message”
8. 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 \*/

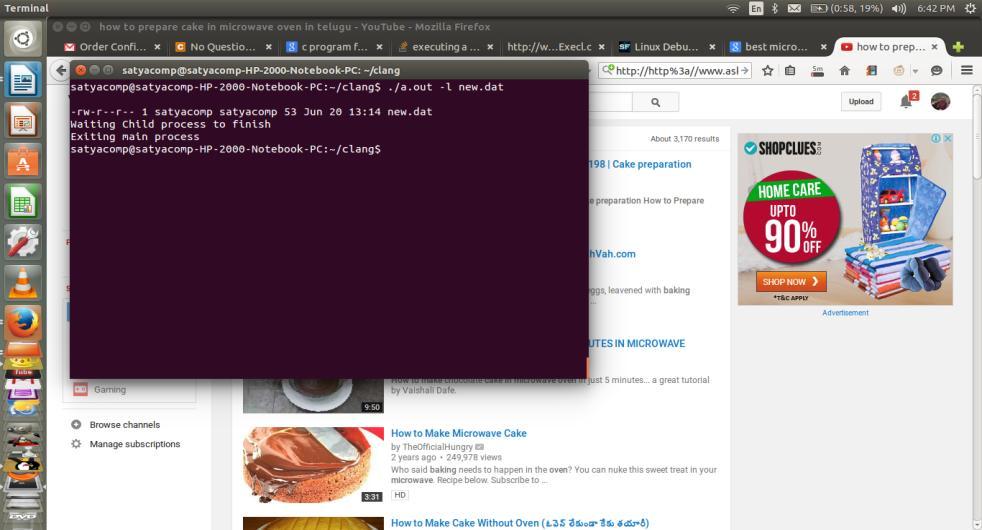
printf("\nWaiting Child process to finish"); //waitpid(pid,0,0); /\* wait for child to exit \*/ wait(NULL);

}

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

}

**Output:**



**Viva-voice:**

**1. What is process?**

Ans: Process is an active program currently running.

1. **Where is the context of process is saved?**

Ans: Process Control Block(PCB).

1. **How to create new child process?**

Ans: Using “fork()” function unix OS creates a new child process

**4. What is return value of fork()?**

Ans: It returns >0 value for parent process running. It returns 0 if child process is executing. Returns <0 if process creation is failed.

**5. What information is available in PCB?**

Ans: Process state, Process number, Program counter, registers, memory limits and list of open file.

**6. What is process state?**

Ans: Process state contains *new, ready, running, waiting and suspended* states of process. **7. What is program counter?**

Ans: It is next instruction to be executed in the process.

**8. What is registers?**

Ans: Registers are different from system to system. They include index register, stack registers and accumulators etc.,.

**9. What are memory limits?**

Ans: The minimum and maximum memory boundaries of process is defined here.

**10. What is list of open files?**

Ans: A running process may open any data files as background. This information is called opened files.

**Experiment 3(a):**

**Aim:** Simulate all File allocation strategy MVT

**Objective:** Creating Mutliprogramming Variable Tasking with First Fit

**Algorithm**

Step 1: Start

2: Declare blocks,processes,fragments as pointers

3: Declare i,j,totalfragments:=0,np,nb //i,j for index and np-no.of processes,nb-

no.of blocks

4: Input np,nb //no.of processes and no.of blocks

1. Set blocks:=call allocateMemoryBlocks()
2. Set processes:=call allocateProcesses()
3. Set fragments:=allocate with size of no.of processes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 7: Repeat Steps 8 to 13 | For I:=0 to np step 1 | //no.of process times | | |  |
| 8: Repeat Steps 9 to 13 | For j:=0 to nb step 1 | //no.of blocks times | | |  |
| 9: If processes[i]<=blocks[j] Then | |  |  |  |  |
| 10: Set fragments[i]:=blocks[j]-processes[i] //allocation | | | and | subtraction | for |

remaining space

1. Set totalfragments+=fragments[i]
2. Print “Fragment #”,i+1,”Size is : “,fragments[i] //Each fragment and its size

13: Exit loop // inner loop only [End If]

[End For j] [End For i]

1. Print “Total fragmentation : “,totalfragments
2. Stop

**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];

}

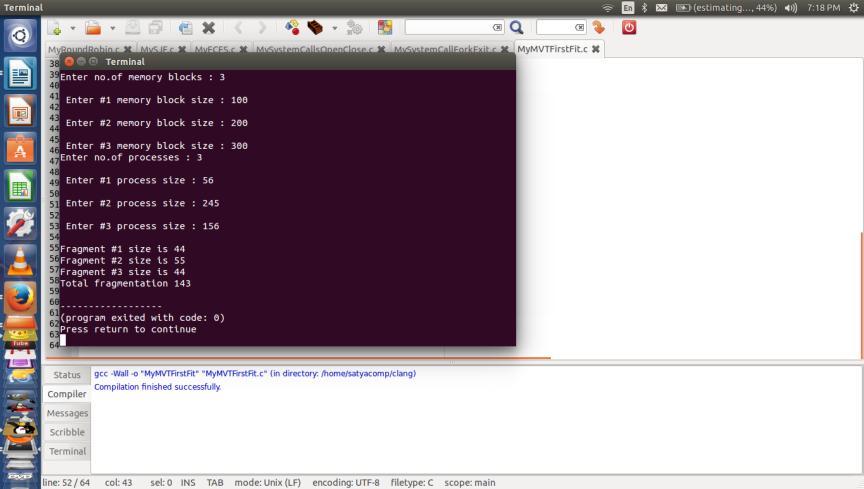
else

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

}

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

**Original output**



**Viva-voice Questions**

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

The set of all logical addresses generated by a program is called a logical address space; the set of all physical addresses corresponding to these logical addresses is a physical address space.

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

The runtime mapping from virtual to physical addresses is done by a hardware device called a memory management unit (MMU).

**3. Define swapping.**

A process needs to be in memory to be executed. However a process can be swapped temporarily out of memory to a backing tore and then brought back into memory for continued execution. This process is called swapping.

1. **What do you mean by best fit?**

Best fit allocates the smallest hole that is big enough. His entire list has to be searched, unless it is sorted by size. His strategy produces the smallest leftover hole.

1. **What do you mean by first fit?**

First fit allocates the first hole that is big enough. Searching can either start at the beginning of the set of holes or where the previous first-fit search ended. Searching can be stopped as soon as a free hole that is big enough is found.

**Experiment 3(b)**

**Aim:** Simulate all File allocation strategy MFT

**Objective:** Creating Mutliprogramming Fixed Tasking

**Algorithm**

Step 1: Start

2: Declare \*processes,\*blocks, totalfrags, i, ,j, np,memory,nb 3: Input np //no.of processes

1. Set processes=readProcesses(processes,np)
2. Input memory

6: Input nb //no.of blocks

1. if(nb<np) Then
2. Print “Error message”
3. Return 1 [End If]
4. Set blocks=allocateMemoryBlocks(blocks,nb,memory/nb)

11: Repeat steps 12 to 17 for i=0,j=0 to nb,np step 1

1. Set fragment:=blocks[i]-processes[i]
2. Print processes[i], blocks[i]
3. If(fragment<0) Then : print “Failed”
4. Else: Print fragment [End If]
5. If(fragment>0) Then: totalfrags+=fragment [End For]
6. Print “Total fragmentation”,totalfrags
7. Stop

**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

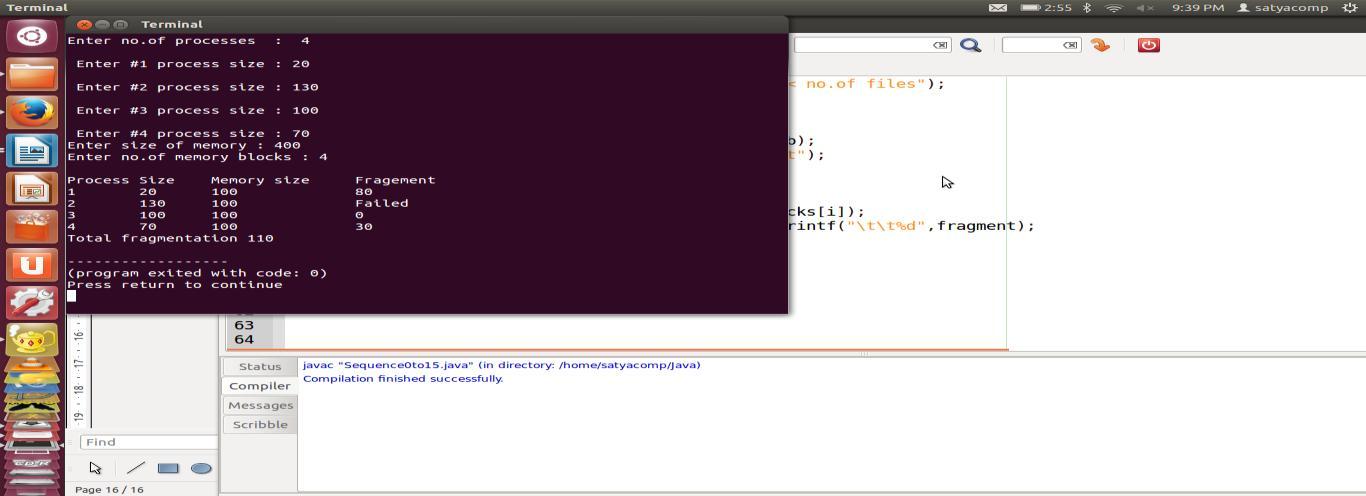
printf("process%d is blocked",p[i]);

}

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

}

**Original output**



**Viva-voice Questions**

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

The set of all logical addresses generated by a program is called a logical address space; the set of all physical addresses corresponding to these logical addresses is a physical address space.

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

The runtime mapping from virtual to physical addresses is done by a hardware device called a memory management unit (MMU).

**3. Define swapping.**

A process needs to be in memory to be executed. However a process can be swapped temporarily out of memory to a backing tore and then brought back into memory for continued execution. This process is called swapping.

**4. What do you mean by best fit?**

Best fit allocates the smallest hole that is big enough. His entire list has to be searched, unless it is sorted by size. His strategy produces the smallest leftover hole.

**5. What do you mean by first fit?**

First fit allocates the first hole that is big enough. Searching can either start at the beginning of the set of holes or where the previous first-fit search ended. Searching can be stopped as soon as a free hole that is big enough is found.

**Experiment: 4**

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

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

**Algorithm**

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]

1. 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!!");

}

**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 0

For process 2 : 2 0 0

For process 3 : 3 0 2

For process 4 : 2 1 1

For process 5 : 0 0 2

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 1 runs to completion!

|  |  |  |
| --- | --- | --- |
| Max matrix: | | Allocation matrix: |
| 0 0 0 | 0 | 0 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-voice Questions**

**1. What is meant by deadlock?**

A set of process is in a deadlock state if each process in the set is waiting for an event that can be caused by only another process in the set Preemptable and Nonpreemptable Resources

**2. What are Necessary and Sufficient Deadlock Conditions?**

1.Mutual Exclusion Condition

2.Hold and Wait Condition

3.No-Preemptive

4.Circular Wait Condition

**3. What is meant by Deadlock Avoidance?**

This approach to the deadlock problem anticipates deadlock before it actually occurs. This approach employs an algorithm to access the possibility that deadlock could occur and acting accordingly. This method differs from deadlock prevention, which guarantees that deadlock cannot occur by denying one of the necessary conditions of deadlock.

**Experiment: 5**

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

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

**Algorithm**

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]

1. 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-voice Questions**

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

A preemptable resource is one that can be taken away from the process with no ill effects. Memory is an example of a preemptable resource.

A non-preemptable resource is one that cannot be taken away from process (without causing ill effect).

For example, CD resources are not preemptable at an arbitrary moment. Reallocating resources can resolve deadlocks that involve preemptable resources.

**2. What are strategies Dealing with Deadlock Problem?**

In general, there are four strategies of dealing with deadlock problem: 1.The Ostrich Approach

Just ignore the deadlock problem altogether.

2.Deadlock Detection and Recovery

Detect deadlock and, when it occurs, take steps to recover. 3.Deadlock Avoidance

Avoid deadlock by careful resource scheduling. 4.Deadlock Prevention

Prevent deadlock by resource scheduling so as to negate at least one of the four conditions.

**3.What is meant by Deadlock Detection?**

Deadlock detection is the process of actually determining that a deadlock exists and identifying the processes and resources involved in the deadlock.

**Experiment: 6(a)**

**Aim:** 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

**Experiment: 6(b)**

**Aim:** 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

**Experiment: 6(c)**

**Aim:** 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-voice Questions**

1**. What is the purpose of page replacement algorithms ?**

**page replacement algorithms** decide which memory pages to page out (swap out, write todisk) when a page of memory needs to be allocated.

**2.What is meant by paging?**

[**Paging**](https://en.wikipedia.org/wiki/Paging)happenswhen a[page fault](https://en.wikipedia.org/wiki/Page_fault)occursand a free page cannot be used to satisfy theallocation, either because there are none, or because the number of free pages is lower than some threshold.

**3.What is meant by fifo?**

The simplest page-replacement algorithm is a FIFO algorithm. The first-in, first-out (FIFO) page replacement algorithm is a low-overhead algorithm that requires little bookkeeping on the part of the [operating system](https://en.wikipedia.org/wiki/Operating_system).

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

A modified form of the FIFO page replacement algorithm, known as the Second-chance page replacement algorithm, fares relatively better than FIFO at little cost for the improvement. It works by looking at the front of the queue as FIFO does, but instead of immediately paging out that page,

**5.What is the advantage of the LRU algorithm**

One important advantage of the LRU algorithm is that it is amenable to full statistical analysis. It has been proven, for example, that LRU can never result in more than N-times more page faults than OPT algorithm, where N is proportional to the number of pages in the managed pool.On the other hand, LRU's weakness is that its performance tends to degenerate under many quite common reference pattern

**Experiment7(a)**

**Aim:** Simulate Sequential file allocation strategy

**Algorithm**

Step 1: Start

1. Declare I
2. Repeat step 4 to 10 for i=0 to n step 1
3. Input files[i].filename
4. Repeat 6-8 While True

6: Input files[i].startBlock

7: If(!isBlockOccupied(files,i,files[i].startBlock) break;

1. print “Sorry!starting block is already occupied, try again\n”

[End While]

1. Repeat 10 to 13 While True
2. Input files[i].size
3. Set files[i].endBlock=files[i].startBlock+files[i].size
4. if(!isBlockOccupied(files,i,files[i].endBlock)) break;
5. Print “Sorry! Ending block range is collaged, try again\n”

[End While]

[End For]

1. Return files
2. Stop

**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

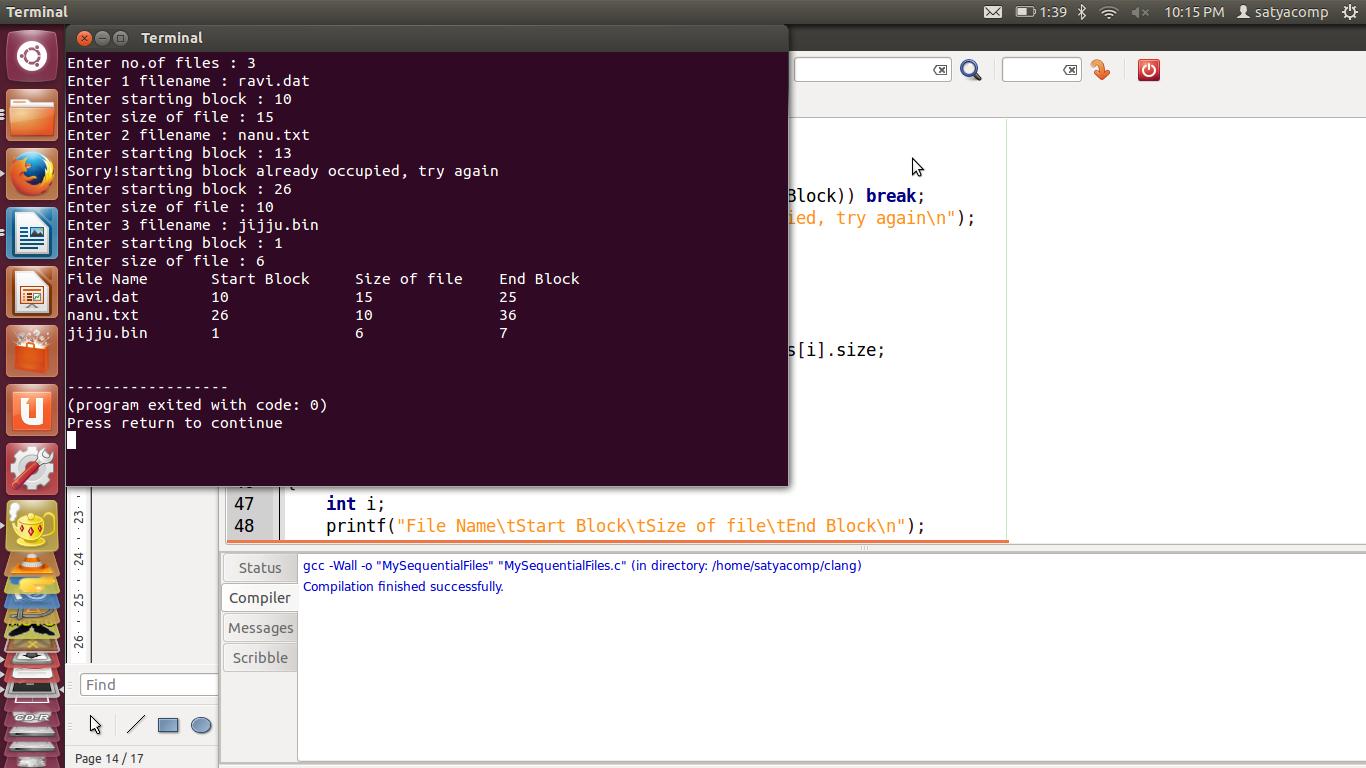
|  |  |  |
| --- | --- | --- |
| 1 | 2 | 4 |
| 2 | 5 | 10 |

Enter file name: rajesh

File name is:12803

length is:0 blocks occupied

**Original output:**



**Experiment7(b)**

**Aim:** Simulate Indexed file allocation strategy

**Algorithm:**

Step 1: Start

1. Declare i
2. Repeat 4-10 for i:=0 to n step 1
3. Input files[i].filename
4. Repeat 6-8 While(true)

|  |  |  |
| --- | --- | --- |
| 6: | Input files[i].indexBlock |  |
| 7: | if(!isIndexBlockOccupied(files,i,files[i].indexBlock) | AND |
| !isChildBlockOccupied(files,i,files[i].indexBlock)) break; | |  |
| 8: | Print “Error” |  |
|  | [End While] |  |

1. Input files[i].noofChildBlocks
2. files[i].childBlocks=allocateChildBlocks(files,i,files[i].noOfChildBlocks) [End For]
3. Return files
4. Stop

**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/Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | | | |  |

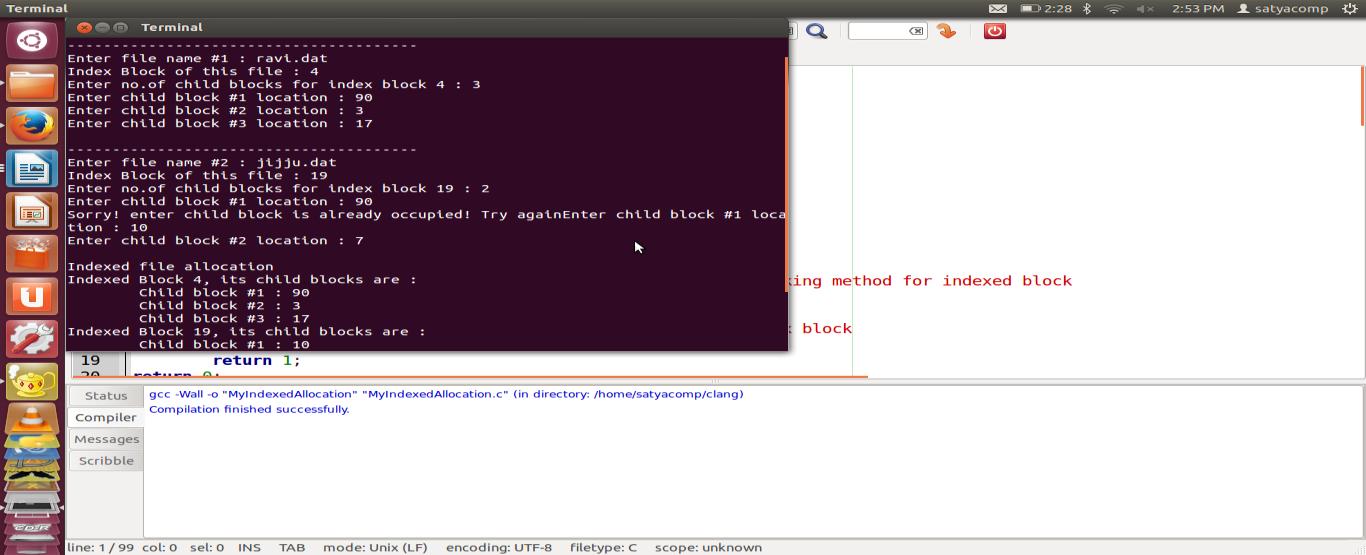
enter blocks of file2: 2 3 4 5 6

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

Enter file name: venkat

file name is:12803 Index is:0

**Original Output**



**Experiment7(c)**

**Aim:** Simulate Linked file allocation strategy

**Algorithm**

1: Start

1. Declare I
2. Repeat step 4 to 10 for i=0 to n step 1
3. Input files[i].filename
4. Repeat 6-8 While True

6: Input files[i].startBlock

7: If(!isBlockOccupied(files,i,files[i].startBlock) break;

1. print “Sorry!starting block is already occupied, try again\n”

[End While]

1. Repeat 10 to 13 While True
2. Input files[i].size
3. Set files[i].endBlock=files[i].startBlock+files[i].size
4. if(!isBlockOccupied(files,i,files[i].endBlock)) break;
5. Print “Sorry! Ending block range is collaged, try again\n”

[End While]

[End For]

1. Return files
2. Stop

**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

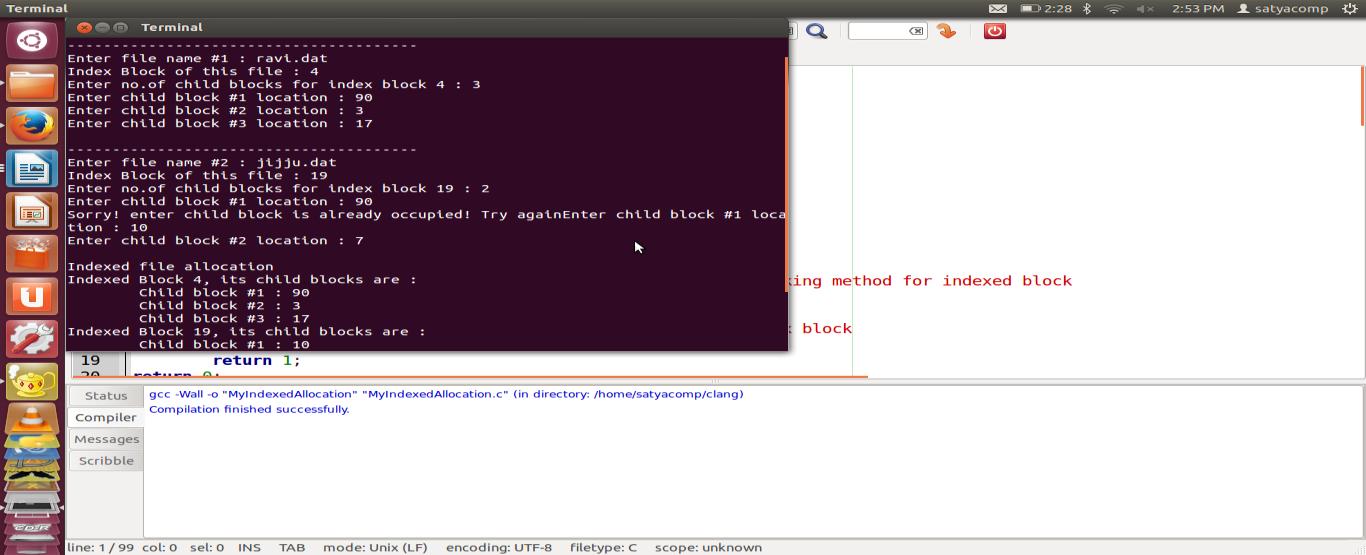
Enter starting block:12

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 |

**Original output**



**Viva-voice Questions**

**1.Define seek time and latency time.**

The time taken by the head to move to the appropriate cylinder or track is called seek time. Once the head is at right track, it must wait until the desired block rotates under the read-write head. This delay is latency time.

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

Three major methods of allocating disk space which are widely in use are

* 1. Contiguous allocation b. Linked allocation c. Indexed allocation

1. **What are the advantages of Contiguous allocation?**

The advantages are

* 1. Supports direct access

b. Supports sequential access

* 1. Number of disk seeks is minimal.

1. **What are the drawbacks of contiguous allocation of disk space?**

The disadvantages are

1. Suffers from external fragmentation b. Suffers from internal fragmentation
2. Difficulty in finding space for a new file
   1. File cannot be extended
   2. Size of the file is to be declared in advance

5. **What are the advantages of Linked allocation?**

The advantages are: a. No external fragmentation

b.Size of the file does not need to be declared

**Additional programs**

**Experiment 1**

**Aim:** Loading executable programs into memory

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

**Requirement Analysis:**

1. Create *main()* function with command line arguments
2. Use function *“execvp()”* to execute any command passed as arguments
3. If failed, print a message

**Algorithm**

Step 1: Start

1. Include required header files for spl. Functions
2. call execvp(argv[1],<address>argv[1])
3. If failed print “error message”
4. 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;

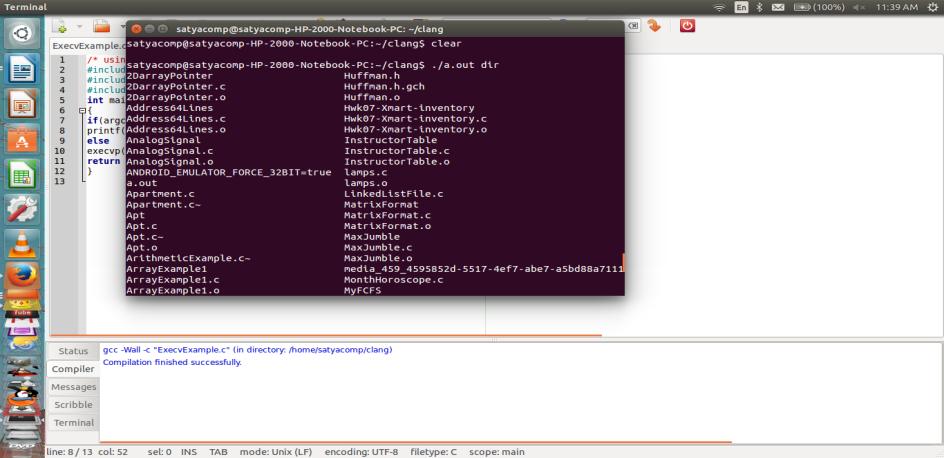
}

**Expected input/output:**

$./a.out clear $./a.out ls

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

**Original input/output:**



**Viva-voice Questions**

**1. What is symmetric multiprocessing?**

Each processor runs an identical copy of the operating system, and these copies communicate with one another as needed.

1. **List out the types in mainframe systems** i) Batch system

ii) Multiprogrammed systems iii) Time-sharing system

1. **What is a file-server system?**

File-server system provides a file system interface where clients can create, update, read, and delete files

**4. What is job scheduling?**

If several jobs are ready to be brought in to memory, and if there is not enough room for all of them, then the system must choose among them. Making this decision is job scheduling.

**Experiment 2**

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

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

**Requirement Analysis:**

1. Create *main()* function with command line arguments
2. Declare file descriptor to handle file addresses
3. Declare *buffer[]* for character content of file
4. If condition to check no.of arguments passed
5. Open files for *read()* and *write()* operations
6. Required a loop for copying processed

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

Step 1: Start

1. Declare input\_fd,output\_fd //for file descriptors
2. Declare ret\_in,ret\_out for no.of character return by *read()* or *write()*
3. If argc!=3 Then
4. print “Usage: <programfile><src><dest>
5. return 1 //error code 1

[End if]

1. Set input\_fd=open(argv[1],O\_RDONLY)
2. If(input\_fd=-1) //read file opening failed
3. Print “error”
4. Return 2 //error code 2

[End if]

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

1. If(output\_fd=-1)
2. Print “error”
3. Return 3 //error code 3

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

1. write to file using *write()* functions
2. If ret\_out!=ret\_in then //if writing is failed
3. Print “error”
4. Return 4 //error code 4
5. Close all files using *close()* functions
6. 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

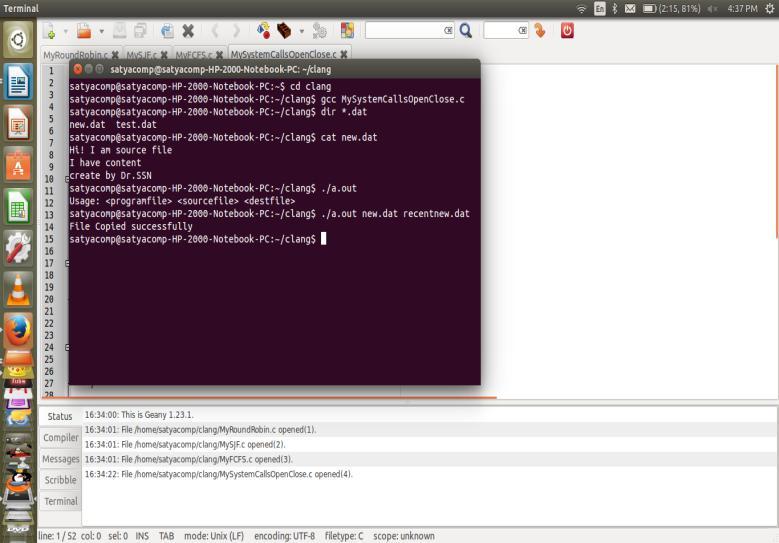
My name is uma

$./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

**Original Input/Output:**



**Viva-voice Questions**

**1. What is symmetric multiprocessing?**

Each processor runs an identical copy of the operating system, and these copies communicate with one another as needed.

1. **List out the types in mainframe systems** i) Batch system

ii) Multiprogrammed systems iii) Time-sharing system

1. **What are file-server systems?**

File-server system provides a file system interface where clients can create, update, read, and delete files

**4. What is job scheduling?**

If several jobs are ready to be brought in to memory, and if there is not enough room for all of them, then the system must choose among them. Making this decision is job scheduling.

**LINUX PROGRAMMING LAB PROGRAMS**

**Experiment: 1a**

**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 youdo not specify a file name, the cat command reads from standard input. You can also specify afile name of – (minus) for standard input.

**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

**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'](mailto:--date%3D'@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 executetwo 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:

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

**Appending Text**

To add text to the end of a line:

* Position the cursor on the last letter of the line.
* Type a
* Enter the new text. This adds text AFTER the cursor.
* 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:

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

To join three lines together:

* Put the cursor on the first line to be joined.
* 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:

* Type / (slash).
* Enter the text to search for.
* 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:

* Press <Esc>.
* Type :w
* Press <Return>.

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

* Press <Esc>.
* Type :q!
* 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?

This is a UNIX command where you need to create a file and change its parameter to read-only by using chmod command you can also change your umask to create read only file.

$touch file

$touch 400 file

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

By looking into file /etc/cpuinfo for example you can use below command:

cat /proc/cpuinfo

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

chmod command is used to change permission of a file or directory in UNIX. The line you see shows the permission for three different set of people : user, group and others. User is the currently logged in user, while group is for all other member which are part of certain group and others means anyone other than user and group member. Each group has three permissions rwx stands for read, write and execute and they are written as user\_group\_others. So in above line, user has only read permission, group members has write permissions and other people has only execute permission. If it is a directory then you need execute permission to go inside that directory.

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

Either by using "uname -a" command or by using "arch" command.

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

By using export command, for example export count=1 will be available on all sub shell.

**6.What different operating systems is vi available for?**

Unix. That's it.However, there are many, many clones of vi that are available for different operating systems.

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

STvi (STevie), elvis, lemmy, vile, vim, and nvi, xvi. elvis is available for: Amiga, DOS, OS/2, Unix, VMS, Atari,Psion3a Handheld.

**8.How to quit without saving?**

:q! will do it.

**9.You have started up vi, but now want to be able to edit another**

**file. How do you do this?**

:n <filename> will load that file into vi. You'll need to save your current file.

**10.How do I search for a control sequence?**

/<ctrl-v><ctrl-<seq>>

<ctrl-v> will tell vi to take the next character literally, and not to take it as a command.

**EXPERIMENT 2**

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

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

**Directions:**

open one existing file 1234.c in read mode using system call.

open one non-existing file zyx in write mode using system call.

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

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);

}

**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().**

Ans: fopen() is the library function which internally calls the system callopen().

**2.What is file pointer?**

Ans: a pointer declared with data type FILE. It is used to store the base address of a file.

**3. What is file descriptor.**

Ans: It is the integer number assigned by the OS while opening the file with the system call open().

**4. What are the modes used in open().**

Ans: O\_WRONLY, O\_RDONLY, O\_RDWR

**5. What is FILE?**

Ans: it is a pre-defined structure.

**EXPERIMENT 3**

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

**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:

[srinivas]$ vi ls.c

[srinivas]$ cc ls.c

[srinivas]$ ./a.out

Child process complete[srinivas]$ total 120

-rw-rw-r-- 1 14JR1A0509 14JR1A0509 0 2016-07-28 21:47 1.c

-rw-rw-r-- 1 14JR1A0509 14JR1A0509 0 2016-08-11 21:44 2

-rw-rw-r-- 1 14JR1A0509 14JR1A0509 0 2016-08-23 22:29 509.c

-rw-rw-r-- 1 14JR1A0509 14JR1A0509 0 2016-09-15 22:00 9a.sh

-rw-rw-r-- 1 14JR1A0509 14JR1A0509 46 2016-06-23 21:58 aa

-rw-rw-r-- 1 14JR1A0509 14JR1A0509 1 2016-08-11 21:10 a.c

**Viva Questions:**

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

Ans: It reads the path of current working directory and stores that path as a string in str. **2.what is the task of alphasort().**

Ans: it will sort the file names stored in the ―file‖.

**3.why scandir() is used?**

Ans: To scann the filedirectory names in a specific directory. **4.dirent stands for \_\_\_\_\_\_\_\_?**

Ans: directory entry.

**5.what is the third argument in the scandir?**

Ans: during scanning if we want to give some conditions then we write the name of the function where condition is defined. Ex: select files if file size more than 100

**EXPERIMENT 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 call

2. 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");

}

}

**Output**

[srinivas]$ vi concurrent.c

[srinivas]$ cc concurrent.c

[srinivas]$ ./a.out ls sort

1.c

2

509.c

9a.sh

aa

a.c

fork.c

k

k.c

linux

linux\_1

ls.c

[srinivas]$

**Viva Questions**

**1. What is pipe()?**

Ans: it is a system call which will create a pipe with two file descrptors. fd[0] and fd[1]. One is for reading and other for writing.

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

Ans:It will return zero on success and -1 on failure.

**3. What is the task of exec family members?**

Ans: They will execute the command given in the area located to that process.

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

Ans: it is a good habbit for synchronization between reading and writing.

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

Ans: stdlib.h

**EXPERIMENT 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.**

Ans: Synchronization should be done to protect the data corruption by simultaneous writes from two processes running on two different processors.

Synchronization between two processors is best done with the help of spinlocks. When a process on one processor had locked the shared memory the other process running on the second processor shall be doing a busy spin until the first process released the lock. This is the funda of Spin locks

**2. What is shmget()?**

Ans: it is a system call to get memory allocation for a shared memory.

**3.What is the return of shmget()?**

Ans:A valid segment identifier, shmid, is returned on success, -1 on error.

**4**.**what is ftok()?**

Ans: it is the system call which convert a pathname and a project identifier to a System V IPC key.

**5.what is the task of shmctl()?**

Ans:shmctl() performs the control operation specified by cmd(2nd argument) on the shared memory segment whose identifier is given in shmid(1st argument).

**EXPERIMENT 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**

[srinivas]$ cc semaphore.c

[srinivas]$ ./a.out

1.Producer

2.Consumer

3.Exit

Enter your choice1

Producer produces item 1

Enter your choice1

Producer produces item 2

Enter your choice2

Consumer consumes the item 2

Enter your choice2

Consumer consumes the item 1

Enter your choice3

[srinivas]$

**Viva Questions:**

**1. What are PTHREAD\_COND\_INITIALIZER, THREAD\_COND\_INITIALIZER, PTHREAD\_MUTEX\_INITIALIZER.**

Ans:Pre-defined constants.

**2. What is pthread\_create().**

Ans: to create a thread to run its own function.

**3. What is pthread\_join()?**

Ans: a system call to make the parent process wait till the death of the child process.

**4. What is producer consumer problem.**

Ans:Suppose a producer produces 50 units per day but the consumer is having capacity to consume 100 units per day or vice-versa.Then it is called the producer consumer problem.

**5. What is pthread\_t?**

Ans:It is the data type for the thread id.

**EXPERIMENT 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().**

Ans: to create a thread to run its own functions.

**2. What is pthread\_join()?**

Ans: a system call to make the parent process wait till the death of the child process.

**3. What is pthread\_t?**

Ans:It is the data type for the thread id.

**4. Why thread is used?**

Ans: thread is used to exeute functions on it independently.

**5. What is the command to compile thread program?**

Ans: gcc filename.c -lpthread

**EXPERIMENT 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

2. Wait the main() to finish up the child threads 3.main terminated.

**Algorithm:**

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?**

Ans: Sharing of CPU time among the threads depends upon the CPU scheduling.

**2. What is pthread\_join ()?**

Ans: a system call to make the parent process wait till the death of the child process.

**3. What is pthread\_t?**

Ans: It is the data type for the thread id.

**4. Why thread is used?**

Ans: thread is used to exeute functions on it independently.

**5. What is the command to compile thread program?**

Ans: gcc filename.c –lpthread

**Additional programs**

**1.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.**

**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

**Output**

[srinivas]$ vi between.sh

[srinivas]$ sh between.sh caps 1 4

A B C D

[srinivas]$

**2. 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.**

**Program**

for i in `ls`

do

if [ -r $i -a -w $i -a -x $i ]

then

echo $i

fi

Done

**Output**

[srinivas]$ vi rwe.sh

[srinivas]$ sh rwe.sh

a.out

[srinivas]$

**3. Write a shell script to take input from user as file name and word which finds the how many occurances of word in a file.**

**Program**

echo "Enter filename"

read file

echo "Enter word to search in a file"

read word

grep $word $file|wc –l

**Output**

[srinivas]$ vi word.sh

[srinivas]$ sh word.sh

Enter filename

z

Enter word to search in a file

lab

2

[srinivas]$