

|  |
| --- |
| **Operating System** |
| Lab Manual |
| **Department of Computer Science and Engineering**  **The NorthCap University, Gurugram** |

**Operating System**

**CSL303**

**Author Name**

Dr Priyanka Vashisht

Dr Anuradha Dhull

Ms Monika Yadav

Ms Jyoti Yadav

Ms Ishita

Ms Akansha Singh



Department of Computer Science and Engineering

NorthCap University, Gurugram- 122001, India

Session 2021-22

*Published by:*

**School of Engineering and Technology**

**Department of Computer Science & Engineering**

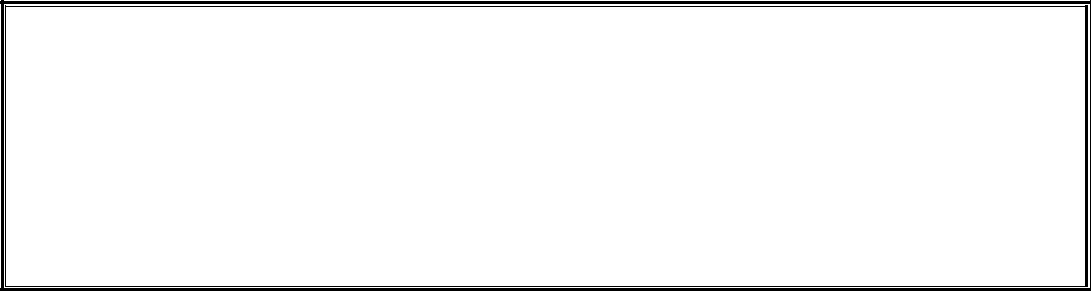
**The NorthCap University Gurugram**

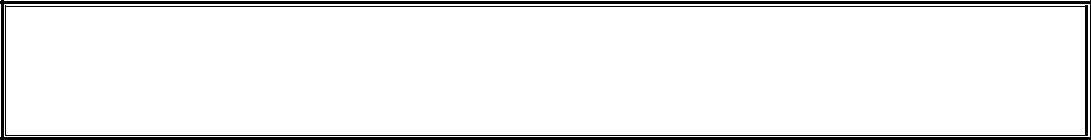
**• Laboratory Manual is for Internal Circulation only**

**© Copyright Reserved**

*No part of this Practical Record Book may be*

*reproduced, used, stored without prior permission of The NorthCap University*



Copying or facilitating copying of lab work comes under cheating and is considered as use of unfair means. Students indulging in copying or facilitating copying shall be awarded zero marks for that particular experiment. Frequent cases of copying may lead to disciplinary action. Attendance in lab classes is mandatory.

Labs are open up to 7 PM upon request. Students are encouraged to make full use of labs beyond normal lab hours.

**PREFACE**

Operating System Lab Manual is designed to meet the course and program requirements of NCU curriculum for B.Tech III year students of CSE branch. The concept of the lab work is to give brief practical experience for basic lab skills to students. It provides the space and scope for self-study so that students can come up with new and creative ideas.

The Lab manual is written on the basis of “teach yourself pattern” and expected that students who come with proper preparation should be able to perform the experiments without any difficulty. Brief introduction to each experiment with information about self-study material is provided. The laboratory exercises will include familiarization with LINUX system calls for process management and inter-process communication; Experiments on process scheduling and other operating system tasks through simulation/implementation. Students would require design process synchronization, CPU scheduling algorithms, memory management and disc management algorithms in high level languages like c, c++, python. Finally, the students would require applying the operating system concepts by experimenting on either xv6/minix operating systems. At the start of each experiment a question bank for preparation and practice is suggested which may be used to test the basic understanding of the students about the experiment. Students are expected to come thoroughly prepared for the lab. General disciplines, safety guidelines and report writing are also discussed.

The lab manual is a part of curriculum for the TheNorthCap University, Gurugram. Teacher’s copy of the experimental results and answer for the questions are available as sample guidelines.

We hope that lab manual would be useful to students of CSE, IT, ECE and BSc branches and author requests the readers to kindly forward their suggestions / constructive criticism for further improvement of the work book.

Author expresses deep gratitude to Members, Governing Body-NCU for encouragement and motivation.

**Authors**

**The NorthCap University**

**Gurugram, India**

|  |  |  |  |
| --- | --- | --- | --- |
| **CONTENTS** | | | |
| S.N. | **Details** | Page No. | |
|  | **Syllabus** |  | |
| 1 | **Introduction** |  | |
| 2 | **Lab Requirement** |  |
| 3 | **General Instructions** |  |
| 4 | **List of Experiments** |  |
| 5 | **List of Flip Assignment** |  |
| 6 | **List of Projects** |  |
| 7 | **Rubrics** |  |
| 8 | **Annexure 1 (Format of Lab Report)** |  |
| 9 | **Annexure 2 (Format of Lab Certificate)** |  |

**SYLLABUS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. **Department:** | | **Department of CSE** | | |
| 1. **Course Name: Operating Systems** | | 1. **Course Code :** | 1. **L- P** | 1. **Credits** |
| **Code: CSL 303** | 3-2 | 4 |
| 1. **Type of Course (Check one):** | | Programme Core Programme Elective Open Elective | | |
| 1. **Frequency of offering (check one):** Odd Even Either Sem. Every Sem. | | | | |
| 1. **Brief Syllabus:** This is an introductory course which briefs LINUX Operating System Concepts that forms an integral part of computer science engineering in development of software applications in many diverse areas, including Web Development, Windows Applications, Research, Analytics and Processing. It lays the foundation of Process Management & Scheduling, Memory Management, Deadlocks and other Operating system Concepts. | | | | |
| 1. **Total lecture and Practical Hours for this course: 30 Hours**   The class size is maximum 30 learners. | | | | |
| 1. **Course Outcomes (COs)**   Possible usefulness of this course after its completion i.e. how this course will be practically useful to him once it is completed | | | | |
| **CO 1** | The students will be able to understand the basic architecture of Linux. | | | |
| **CO 2** | The students will be able to understand the process management & scheduling of Linux. | | | |
| **CO 3** | The students will be able to understand the memory management of Linux. | | | |
| **CO 4** | The students will be able to understand the inter process communication of Linux. | | | |
| **CO 5** | They will understand the main principles and techniques to handle the deadlocks. | | | |
| **CO6** | They will understand the I/O device management & the VFS of Linux. | | | |
| 1. **UNIT WISE DETAILS No. of Units: -05** | | | | |
| **Unit 1: Introduction to Linux OS Hours: 6**  Introduction & overview: functions of operating systems, Overview of various Operating Systems, Linux architecture, Boot strap loader of Linux, Tasks of the kernel, implementation strategies of kernel, System Calls. | | | | |
| **Unit II: Process Management & Scheduling Hours: 6**  Process priorities, process life cycle of Linux, process representation: process types, process identification numbers, process management system calls, kernel thread, overview of different scheduling algorithms, Linux scheduler: priority and completely fair share scheduling algorithm. | | | | |
| **Unit III: Process Synchronization and Memory Management Hours: 8**  Implementation of Producer- Consumer problem, implementation of semaphores, Page-Replacement Algorithms. | | | | |
| **Unit IV: Deadlocks Hours: 6**  Implementation of Banker’s Algorithm, | | | | |
| **Unit V: Virtual File System Hours: 4**  Disk scheduling algorithms, Introduction to VFS File System types, Common File model, Structure of the VFS | | | | |
| 1. **Guided Project (No. of Hours):** Case Study on Windows OS 2. **Unguided Project (No. of Hours):** Case Study ofLinux, Window, MAC OS | | | | |
| 1. **Brief Description of Self-learning component by students (through books/resource material etc.): Topics:** Linux syntax for shell scripting, revise c/c++/Python and data structure concepts from previous semesters | | | | |
| 1. **Suggested Readings**   GNU/Linux Command−Line Tools Summary [eBook]  <http://www.tldp.org/LDP/Bash-Beginners-Guide/Bash-Beginners-Guide.pdf>  **websites:**   * <https://www.linuxjournal.com/> * <https://www.omgubuntu.co.uk/> | | | | |

1. **INTRODUCTION**

That ‘learning is a continuous process’ cannot be over emphasized. The theoretical knowledge gained during lecture sessions need to be strengthened through practical experimentation. Thus practical makes an integral part of a learning process.

The purpose of conducting experiments can be stated as follows:

* To familiarize the students with the basic concepts, programming skill development and the take home laboratory assignments mainly implementation-oriented which have to be coded in high level language. The lab sessions will be based on exploring the concepts discussed in class.
* Observing basic structure and characteristics of Operating Systems
* Reporting and analyzing the complexities.
* Hands on experience on the experimental setup and software tools

1. **LAB REQUIREMENTS**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Requirements** | **Details** |
| **1** | **Software Requirements** | Linux’s Shell, Python/c/c++ |
| **2** | **Operating System** | Linux Operating System |
| **3** | **Hardware Requirements** | Windows and Linux: Intel 64/32 or AMD Athlon 64/32, or AMD Opteron processor  2 GB RAM  80 GB hard disk space |
| **4** | **Required Bandwidth** | NA |

1. **GENERAL INSTRUCTIONS** 
   1. **General discipline in the lab**
   * Students must turn up in time and contact concerned faculty for the experiment they are supposed to perform.
   * Students will not be allowed to enter late in the lab.
   * Students will not leave the class till the period is over.
   * Students should come prepared for their experiment.
   * Experimental results should be entered in the lab report format and certified/signed by concerned faculty/ lab Instructor.
   * Students must get the connection of the hardware setup verified before switching on the power supply.
   * Students should maintain silence while performing the experiments. If any necessity arises for discussion amongst them, they should discuss with a very low pitch without disturbing the adjacent groups.
   * Violating the above code of conduct may attract disciplinary action.
   * Damaging lab equipment or removing any component from the lab may invite penalties and strict disciplinary action.
   1. **Attendance**

* Attendance in the lab class is compulsory.
* Students should not attend a different lab group/section other than the one assigned at the beginning of the session.
* On account of illness or some family problems, if a student misses his/her lab classes, he/she may be assigned a different group to make up the losses in consultation with the concerned faculty / lab instructor. Or he/she may work in the lab during spare/extra hours to complete the experiment. No attendance will be granted for such case**.**
  1. **Preparation and Performance**
* Students should come to the lab thoroughly prepared on the experiments they are assigned to perform on that day. Brief introduction to each experiment with information about self study reference is provided on LMS.
* Students must bring the lab report during each practical class with written records of the last experiments performed complete in all respect.
* Each student is required to write a complete report of the experiment he has performed and bring to lab class for evaluation in the next working lab. Sufficient space in work book is provided for independent writing of theory, observation, calculation and conclusion.
* Students should follow the Zero tolerance policy for copying / plagiarism. Zero marks will be awarded if found copied. If caught further, it will lead to disciplinary action.
* Refer **Annexure 1** for Lab Report Format

1. **LIST OF EXPERIMENTS**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Exp. No.** | **Division of Experiments** | **List of Experiments** | **Software Used** | **Unit Covered** | **CO Covered** | **Time Required** |
| 1 | Basics of Linux | Explain the structure of Linux Operating System | **Cygwin** | 1 | CO1 | 2 Hrs |
|  | Installation of Ubuntu Operating system | Cygwin | 1 | CO1 | 2 Hrs |
| 1 | Shell Programs | Write a shell program to find factorial of a number. |  | 1 | CO1 | 2 Hrs |
| 2 | Write a shell program to find gross salary of an employee. | Cygwin | 1 | CO1 | 2 Hrs |
| 3 | Write a shell program to display the menu and execute instructions accordingly  (i)List of file (ii)Process Status (iii) Date (iv) users in program (v) Quit | Cygwin | 1 | CO1 | 2 Hrs |
| 4 | Write a shell program to find Fibonacci series. | Cygwin | 1 | CO1 | 2 Hrs |
| 5 | Write a shell program to find largest of three numbers. | Cygwin | 1 | CO1 | 2 Hrs |
| 6 | Write a shell program to find average of N numbers | Cygwin | 1 | CO1 | 2 Hrs |
| 7 | CPU Scheduling Algorithms | Write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time.  a) FCFS b) SJF c) Round Robin (pre-emptive) d) Priority | C++/ Python/ Java | 2 | CO2 | 2 Hrs |
| 8 | \*Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue. | C++/ Python/ Java | 2 | CO2 | 2 Hrs |
| Implement the following CPU scheduling Algorithms.  i) Round Robin  ii) Priority Based | C++/ Python/ Java | 2 | CO2 | 2 Hrs |
| 9 | Deadlock Management  Technique | Write a program to simulate Bankers algorithm for the purpose of deadlock avoidance | C++/ Python/ Java | 3 | CO3 | 2 Hrs |
| 10 | Page Replacement  Algorithms | Write a C program to simulate page replacement algorithms  a) FIFO b) LRU c) LFU | C++/ Python/ Java | 4 | CO4 | 2 Hrs |
| 11 | Write a C program to simulate page replacement algorithms  a) Optimal | C++/ Python/ Java | 4 | CO4 | 2 Hrs |

1. **LIST OF FLIP EXPERIMENTS**
2. Execute the **who** command written in a file to instruct the shell to read input from a file called "myfile1" instead of from the keyboard. Use the **more** command to see the contents of myfile1.
3. Use the date and who commands in sequence (in one line) such that the output of date will display on the screen and the output of who will be redirected to a file called myfile2. Use the more command to check the contents of myfile2
4. Write a sed command that swaps the first and second words in each line in a file.
5. Write a shell script that takes a command –line argument and reports on whether it is directory, a file, or something else.
6. Write a shell script that accepts one or more file name as arguments and converts all of them to uppercase, provided they exist in the current directory.
7. Write a shell script that determines the period for which a specified user is working on the system.
8. Write a shell script that accepts a file name, starting and ending line numbers as arguments and displays all the lines between the given line numbers.
9. Write a shell script that deletes all lines containing a specified word in one or more files supplied as arguments to it.
10. Write a shell script to perform the following string operations:
    * 1. To extract a sub-string from a given string
      2. To find the length of a given string
11. **LIST OF PROJECTS**
    * + 1. Case Study of Window OS
        2. Case Study of Linux OS
        3. Case Study of MAC OS
12. **RUBRICS**

|  |  |
| --- | --- |
| **Marks Distribution** | |
| **Continuous Evaluation(50 Marks)** | **End Semester Exam (20 Marks)** |
| Each experiment shall be evaluated for 10 marks and at the end of the semester proportional marks shall be awarded out of 50. | End semester practical evaluation including Mini project (if any) carries 20 marks. |
| Following is the breakup of 10 marks for each  **4 Marks**: Observation & conduct of experiment. Teacher may ask questions about experiment.  **3 Marks:** For report writing  **3 Marks:** For the 15 minutes quiz to be conducted in every lab. |

**Annexure 1**

**Operating System**

**(CSL 303)**

**Lab Practical Report**

Faculty name Dr. Priyanka Vashisht Student name : Namit Kumar

Roll No.:19CSU185

Semester:V

Group:A1



**Department of Computer Science and Engineering**

**NorthCap University, Gurugram- 122001, India**

**Session 2021-22**

**INDEX**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| S.No | Experiment | Page No. | Date of Experiment | Date of Submission | Marks | CO Covered | Signature |
| 1 | Implement the following things:   * Cygwin Installation * Basic Linux commands | 20 | 6th August 2021 | 6th August 2021 |  | CO1 |  |
| 2 | a) Write A Shell Program of Hello World  b) Write a shell program to find factorial of a number.  c) Write a shell program to find gross salary of an employee.  d) Write a shell program to display the menu and execute instructions accordingly  **(i)**List of files **(ii)**Process Status **(iii)** Date **(iv)** users in program **(v)** Quit | 27 | 10th August 2021 | 10th August 2021 |  | CO1 |  |
| 3 | a) Write a shell program to find Fibonacci series.  b) Write a shell program to find largest of three numbers.  c) Write a shell program to find average of N numbers | 34 | 17th August | 17th August |  | CO1 |  |
| 4 | a) Write a shell program to check whether a number is even or odd  b) Write a shell program to find whether a number is prime or not.  c) Write a shell program to find whether a number is palindrome or not.  d) Write a shell program to type number 1 to 7 and then print its corresponding day of week | 39 | 8th September | 8th September |  | CO1 |  |
| 5 | Implement the following CPU scheduling Algorithms.   1. FCFS with Arrival time 2. FCFS without Arrival time | 46 | 8th September | 8th September |  | CO2 |  |
| 6 | Implement the following CPU scheduling Algorithms.   * SJF (Non-Preemptive) * SJTF (shortest remaining time first -Preemptive SJF) | 51 | 8th September | 8th September |  | CO2 |  |
| 7 | Implement the priority scheduling. | 60 | 6th October | 6th October |  | CO2 |  |
| 8 | Implement the Round Robin Algorithm. | 66 | 13th October | 13th October |  | CO2 |  |
| 9 | Write a program to implement reader/writer problem using semaphore | 72 | 20th October | 27th October |  | CO3 |  |
| 10 | Write a program to implement Dining Philosopher problem using semaphore | 79 | 10th November | 10th November |  | CO3 |  |
| 11 | Write a program to implement Banker’s algorithm for deadlock avoidance. | 88 | 17th November |  |  | CO3 |  |
| 12 |  |  |  |  |  |  |  |

**Experiment No: 1**

|  |
| --- |
| **Student Name and Roll Number:** Namit Kumar 19CSU185 |
| **Semester /Section:** V/FS-A-1 |
| **Link to Code:** https://github.com/NamitKumar16/OS |
| **Date:** 6th August 2021 |
| **Faculty Signature:** |
| **Marks:** |



|  |
| --- |
| **Objective(s):**  To familiarize the students to Linux interface. |
| **Outcome:**   * The students will understand commands used in Linux. |
| **Problem Statement:**  Implement the following things:   * Cygwin Installation * Basic Linux commands |
| **Background Study:**  Cygwin is a open source tool which provides that functionality of the Linux in windows Operating System. Cygwin is a large collection of GNU and Open Source tools which provide functionality similar to a [Linux distribution](https://en.wikipedia.org/wiki/Linux_distribution) on Windows. It is a DLL (cygwin1.dll) which provides substantial POSIX API functionality. |
| **Question Bank:**   1. **What is Linux?** 2. How will you List files from a directory? 3. How files in a directory can be removed? 4. How to find out a word in a file? 5. What are wildcards? |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**Q1.** Linux is a Unix-like, open source and community-developed operating system for computers, servers, mainframes, mobile devices and embedded devices. It is supported on almost every major computer platform including x86, ARM and SPARC, making it one of the most widely supported operating systems.

**Q2.** The ls command is used to list files or directories in Linux and other Unix-based operating systems.

**Q3.** Files can be removed using the rm command.

**Q4.** Grep is a Linux / Unix command-line tool used to search for a string of characters in a specified file. The text search pattern is called a regular expression. When it finds a match, it prints the line with the result.

**Q5.** A wildcard in Linux is a symbol or a set of symbols that stands in for other characters. It can be used to substitute for any other character or characters in a string. For example, you can use a wildcard to get a list of all files in a directory that begin with the letter O

**Screenshots**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

A picture containing text

Description automatically generated

A picture containing text, screenshot

Description automatically generated

History Command to show all the commands used :-

A screenshot of a computer

Description automatically generated

**EXPERIMENT NO. 2**

|  |
| --- |
| **Student Name and Roll Number:** Namit Kumar 19CSU185 |
| **Semester /Section:** V/FSA1 |
| **Link to Code:** https://github.com/NamitKumar16/OS |
| **Date:** 10th August 2021 |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective:**  To write the shell programming code for the following. |
| **Outcome:**  Student is able to write code in shell programming |
| **Problem Statement:**  a) Write A Shell Program of Hello World  b) Write a shell program to find factorial of a number.  c) Write a shell program to find gross salary of an employee.  d) Write a shell program to display the menu and execute instructions accordingly  **(i)**List of files **(ii)**Process Status **(iii)** Date **(iv)** users in program **(v)** Quit |
| **Background Study:**  A shell script is a file with a set of commands in it. The shell reads this file and executes the instructions as if they were input directly on the command line.  A shell is a command-line interpreter and operations such as file manipulation, program execution and text printing is performed by shell script. So, we will use vi editor to edit our files. |
| **Question Bank:**   1. What is a shell? 2. What is the significance of $#? 3. What are the different types of commonly used shells on a typical Linux system? 4. How will you pass and access arguments to a script in Linux? 5. Use sed command to replace the content of the file (emulate tac command) |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**Algorithm/Flowchart/Code/Sample Outputs**

**Q1** Shell is a program that takes commands from the keyboard and gives them to the operating system to perform.

**Q2** $# shows the count of the arguments passed to the script.

**Q3** Bash, Zsh, Korn, Tcsh, Fish are commonly used shells

**Q4** Arguments can be passed to the script when it is executed, by writing them as a space-delimited list following the script file name.

Inside the script, the $1 variable references the first argument in the command line, $2 the second argument and so forth. The variable $0 references to the current script.

**Q5** sed ‘1! G; h;$!d’ file1

Here G command appends to the pattern space,

h command copies pattern buffer to hold buffer

and d command deletes the current pattern space.

**Screenshots**

1. Write A Shell Program of Hello World

A screenshot of a computer

Description automatically generated

Graphical user interface, text, application

Description automatically generated

1. Write a shell program to find factorial of a number.

A screenshot of a computer

Description automatically generated

Graphical user interface, text, application

Description automatically generated

1. Write a shell program to find gross salary of an employee.

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated

1. Write a shell program to display the menu and execute instructions accordingly

**(i)**List of files **(ii)**Process Status **(iii)** Date **(iv)** users in program **(v)** Quit

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

**EXPERIMENT NO. 3**

|  |
| --- |
| **Student Name and Roll Number:** Namit Kumar |
| **Semester /Section:** V/FS-A-1 |
| **Link to Code:** https://github.com/NamitKumar16/OS |
| **Date:** 17th August 2021 |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective:**  To write the shell programming code for the following. |
| **Outcome:**  Student is able to write code in shell programming |
| **Problem Statement:**  a) Write a shell program to find Fibonacci series.  b) Write a shell program to find largest of three numbers.  c) Write a shell program to find average of N numbers |
| **Background Study:**  A shell script is a file with a set of commands in it. The shell reads this file and executes the instructions as if they were input directly on the command line.  A shell is a command-line interpreter and operations such as file manipulation, program execution and text printing is performed by shell script. So, we will use vi editor to edit our files. |
| **Question Bank:**   1. How to use multi line comments in shell script? 2. What is the difference between soft and hard links? 3. Explain about loops and what are the loops available in LINUX? 4. What are absolute and relative paths. 5. How to debug a shell script. |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**Q1** Ans - In shell or bash shell, we can comment on multiple lines using << and name of comment.

**Q2** Ans - Hard Links cannot be used across file systems whereas soft links can be used across file systems.

**Q3** Ans - The while loop

The for loop

The until loop

The select loop

**Q4** Ans - An absolute path always contains the root element and the complete directory list required to

locate the file.

A relative path needs to be combined with another path in order to access a file.

1. Write a shell program to find largest of three numbers

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

1. Write a shell program to find average of N numbers

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

1. Find average of 3 numbers

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

**EXPERIMENT NO. 4**

|  |
| --- |
| **Student Name and Roll Number:** Namit Kumar |
| **Semester /Section:** V/FS-A-1 |
| **Link to Code:** https://github.com/NamitKumar16/OS |
| **Date:** 8th September |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective:**  To write the shell programming code for the following. |
| **Outcome:**  Student is able to write code in shell programming |
| **Problem Statement:**  a) Write a shell program to check whether a number is even or odd  b) Write a shell program to find whether a number is prime or not.  c) Write a shell program to find whether a number is palindrome or not.  d) Write a shell program to type number 1 to 7 and then print its corresponding day of week |
| **Background Study:**  A shell script is a file with a set of commands in it. The shell reads this file and executes the instructions as if they were input directly on the command line.  A shell is a command-line interpreter and operations such as file manipulation, program execution and text printing is performed by shell script. So, we will use vi editor to edit our files. |
| **Question Bank:**   1. What are Zoombie Process? 2. What are different types of variables used in shell script? 3. What are the different types of modes available in Vi editor? 4. What are the different types of permission at file level in shell? 5. How to use comments in shell script. |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample output**

**Q1** Ans - A zombie process is a process whose execution is completed but it still has an entry in the

process table.

**Q2** Ans - System-Defined Variables.

User-Defined Variables.

**Q3** Ans - Command Mode , Entry Mode And Last Line Mode

**Q4** Ans - Execute Permission

Write Permission

Write and execute permission.

Read Permission

**Q5** Ans - By Using #

**Screenshots**

1. Write a shell program to check whether a number is even or odd

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Write a shell program to find whether a number is prime or not.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Write a shell program to find whether a number is palindrome or not.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Write a shell program to type number 1 to 7 and then print its corresponding day of week

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

**Experiment No: 5**

|  |
| --- |
| **Student Name and Roll Number:** Namit Kumar |
| **Semester /Section:** V/FS-A1 |
| **Link to Code:** https://github.com/NamitKumar16/OS |
| **Date:** 8th September |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective:**  Write a program to implement CPU scheduling for first come first serve approach. |
| **Outcome:**  The students will understand the First-cum-first-serve algorithm |
| **Problem Statement:**  Implement the following CPU scheduling Algorithms.   1. FCFS with Arrival time 2. FCFS without Arrival time |
| **Background Study:**  **FCFS**   * The simplest CPU-scheduling algorithm is the first-come, first-served (FCFS) scheduling algorithm. With this algorithm, processes are assigned the CPU in the order they request it. * There is a single queue of ready processes. * The implementation of the FCFS policy is easily managed with a FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. * The average waiting time under the FCFS policy, however, is often quite long. |
| **Question Bank:**   1. Which module gives control of the CPU to the process selected by the short-term scheduler?    1. **dispatche**r    2. interrupt    3. scheduler    4. none of the mentioned 2. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called    1. job queue    2. **ready queue**    3. execution queue    4. process queue 3. The interval from the time of submission of a process to the time of completion is termed as    1. waiting time    2. **turnaround time**    3. response time    4. throughput 4. Which scheduling algorithm allocates the CPU first to the process that requests the CPU first?    1. **first-come, first-served scheduling**    2. shortest job scheduling    3. priority scheduling    4. none of the mentioned 5. In priority scheduling algorithm    1. **CPU is allocated to the process with highest priority**    2. CPU is allocated to the process with lowest priority    3. equal priority processes cannot be scheduled    4. none of the mentioned |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

#include<stdio.h>

int main()

{

int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;

printf("Enter total number of processes(maximum 20):");

scanf("%d",&n);

printf("\nEnter Process Burst Time\n");

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

{

printf("P[%d]:",i+1);

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

}

wt[0]=0;

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

{

wt[i]=0;

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

wt[i]+=bt[j];

}

printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

//calculating turnaround time

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

{

tat[i]=bt[i]+wt[i];

avwt+=wt[i];

avtat+=tat[i];

printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);

}

avwt/=i;

avtat/=i;

printf("\n\nAverage Waiting Time:%d",avwt);

printf("\nAverage Turnaround Time:%d",avtat);

return 0;

}

A screenshot of a computer

Description automatically generated with medium confidence

**Experiment No: 6**

|  |
| --- |
| **Student Name and Roll Number:** Namit Kumar 19CSU185 |
| **Semester /Section:** V/FSA1 |
| **Link to Code:** https://github.com/NamitKumar16/OS |
| **Date:** 8th September 2021 |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective:**  Write a program to implement CPU scheduling for shortest job first (Preemptive and Non-Preemptive) |
| **Outcome:**  The students will understand the Shortest Job First scheduling mechanism |
| **Problem Statement:**  Implement the following CPU scheduling Algorithms.   * SJF (Non-Preemptive) * SJTF (shortest remaining time first -Preemptive SJF) |
| **Background Study:**   * Shortest Job first is having the advantage of a minimum average waiting time . * This algorithm associates with each process the length of the process next burst time.When CPU is available it assigned to the process that has the smallest next CPU burst time.if CPU burst time of two process is same then it follows FCFS. * It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing. * It is practically infeasible as Operating System may not know burst time and therefore may not sort them. |
| **Question Bank:**   1. [scheduling algorithm](https://t4tutorials.com/round-robin-process-scheduling-algorithm-in-operating-systems/) In multilevel feedback A. processes are not classified into groups B. a process can move to a different classified ready queue… C. classification of the ready queue is permanent D. none of the mentioned 2. Select one which algorithms tend to minimize the process flow time? A. First come First served B. Earliest Deadline First C. Shortest Job First D. Longest Job First 3. The process can be classified into many groups in A. shortest job scheduling algorithm  B. multilevel queue scheduling algorithm C. round-robin scheduling algorithm D. priority scheduling algorithm 4. The turnaround time for short jobs during multiprogramming is usually Shortened and that for long jobs is slightly \_\_\_\_\_\_\_\_\_\_\_ A. Shortened B. Unchanged C. Lengthened D. Shortened 5. Time quantum can be said A. multilevel queue scheduling algorithm B. round-robin scheduling algorithm C. shortest job scheduling algorithm D. priority scheduling algorithm |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**SJF Non - Preemptive**

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter number of process:");

scanf("%d",&n);

printf("\nEnter Burst Time:\n");

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

{

printf("p%d:",i+1);

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

p[i]=i+1;

}

//sorting of burst times

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

{

pos=i;

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

{

if(bt[j]<bt[pos])

pos=j;

}

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

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

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n;

total=0;

printf("\nProcesst Burst Time \tWaiting Time\tTurnaround Time");

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

{

tat[i]=bt[i]+wt[i];

total+=tat[i];

printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=(float)total/n;

printf("\n\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}

A screenshot of a computer

Description automatically generated with medium confidence

**SJF Preemptive**

#include <stdio.h>

int main()

{

int arrival\_time[10], burst\_time[10], temp[10];

int i, smallest, count = 0, time, limit;

double wait\_time = 0, turnaround\_time = 0, end;

float average\_waiting\_time, average\_turnaround\_time;

printf("\nEnter the Total Number of Processes:\t");

scanf("%d", &limit);

printf("\nEnter Details of %d Processesn", limit);

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

{

printf("\nEnter Arrival Time:\t");

scanf("%d", &arrival\_time[i]);

printf("Enter Burst Time:\t");

scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

burst\_time[9] = 9999;

for(time = 0; count != limit; time++)

{

smallest = 9;

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

{

if(arrival\_time[i] <= time && burst\_time[i] < burst\_time[smallest] && burst\_time[i] > 0)

{

smallest = i;

}

}

burst\_time[smallest]--;

if(burst\_time[smallest] == 0)

{

count++;

end = time + 1;

wait\_time = wait\_time + end - arrival\_time[smallest] - temp[smallest];

turnaround\_time = turnaround\_time + end - arrival\_time[smallest];

}

}

average\_waiting\_time = wait\_time / limit;

average\_turnaround\_time = turnaround\_time / limit;

printf("\n\nAverage Waiting Time:\t%lf\n", average\_waiting\_time);

printf("Average Turnaround Time:\t%lf\n", average\_turnaround\_time);

return 0;

}

Text

Description automatically generated

**Experiment No: 7**

|  |
| --- |
| **Student Name and Roll Number:** Namit Kumar |
| **Semester /Section:** V/FSA-1 |
| **Link to Code:** https://github.com/NamitKumar16/OS |
| **Date:** 6th October 2021 |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective:**  Write a program to perform priority scheduling among a set of processes. |
| **Outcome:**  Student will understand the working of priority scheduling among a set of processes. |
| **Problem Statement:**  Implement the priority scheduling. |
| **Background Study:**  Priority scheduling is a non-preemptive algorithm and used in batch systems. Each process is assigned a priority. Process with highest priority is to be executed first and so on. |
| **Question Bank:**   1. What are advantages of Priority scheduling? 2. What are disadvantages of priority scheduling? 3. At the ready queue when a process arrives In [priority scheduling](https://t4tutorials.com/priority-based-process-scheduling-in-operating-systems/) algorithm, the priority of this process is compared with the priority of? A. currently running process B. parent process C. all process D. init process 4. Differentiate between pre-emptive and non pre-emptive scheduling? 5. What is total no. of queue required to perform Priority Scheduling ? |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**Q1**

* Easy to use scheduling method
* Processes are executed on the basis of priority so high priority does not need to wait for long which saves time
* This method provides a good mechanism where the relative important of each process may be precisely defined.
* Suitable for applications with fluctuating time and resource requirements.

**Q2**

* If the system eventually crashes, all low priority processes get lost.
* If high priority processes take lots of CPU time, then the lower priority processes may starve and will be postponed for an indefinite time.
* This scheduling algorithm may leave some low priority processes waiting indefinitely.
* A process will be blocked when it is ready to run but has to wait for the CPU because some other process is running currently.
* If a new higher priority process keeps on coming in the ready queue, then the process which is in the waiting state may need to wait for a long duration of time.

**Q3** A – Currently Running Process

**Q4** – The basic difference between preemptive and non-preemptive scheduling is that in preemptive scheduling the CPU is allocated to the processes for the limited time. While in Non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to waiting state.

**Algorithm/Flowchart/Code/Sample Outputs**

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg\_wt,avg\_tat;

printf("Enter Total Number of Process:");

scanf("%d",&n);

printf("\nEnter Burst Time and Priority\n");

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

{

printf("\nP[%d]\n",i+1);

printf("Burst Time:");

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

printf("Priority:");

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

p[i]=i+1;

}

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

{

pos=i;

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

{

if(pr[j]<pr[pos] || (pr[j]==pr[pos] && bt[j]>bt[pos]))

pos=j;

}

temp=pr[i];

pr[i]=pr[pos];

pr[pos]=temp;

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

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

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=total/n;

total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

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

{

tat[i]=bt[i]+wt[i];

total+=tat[i];

printf("\nP[%d]\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=total/n;

printf("\n\nAverage Waiting Time=%d",avg\_wt);

printf("\nAverage Turnaround Time=%d\n",avg\_tat);

return 0;

}

Text

Description automatically generated

**Experiment No: 8**

|  |
| --- |
| **Student Name and Roll Number:** Namit Kumar 19CSU185 |
| **Semester /Section:** V/FS-A-1 |
| **Link to Code:** https://github.com/NamitKumar16/OS |
| **Date:** 13th October 2021 |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective: Objective**  To familiarize the students about CPU scheduling Algorithms |
| **Program Outcome**  The students will understand the Round Robin Algorithm. |
| **Problem Statement:**  Implement the Round Robin Algorithm. |
| **Background Study:**   * In Round Robin each process is assigned a fixed time slot in a cyclic way and this is preemptive. It has a disadvantage of context switch and have quantum time |
| **Question Bank:**   1. What is Preemptive and Non- Preemptive CPU scheduling? Explain with examples. 2. Explain the difference between short term, long term and medium term scheduling. 3. Explain the function of Dispatcher and Context Switch mechanism. 4. What are the advantages and disadvantages of Round robin? 5. Give the application are of Robin Robin. |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**Q1** – Preemptive scheduling is used when a process switches from running state to ready state or from the waiting state to ready state. The resources (mainly CPU cycles) are allocated to the process for a limited amount of time and then taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in the ready queue till it gets its next chance to execute. Examples of preemptive scheduling are Round Robin and Shortest Remaining Time First.

**Q2** –

|  |  |  |
| --- | --- | --- |
| **Long Term** | **Short Term** | **Medium Term** |
| It is a job scheduler. | It is a CPU scheduler. | It is swapping. |
| Speed is less than short term scheduler. | Speed is very fast. | Speed is in between both |
| It controls the degree of multiprogramming | Less control over the degree of multiprogramming. | Reduce the degree of multiprogramming. |
| It selects processes from the pool and load them into memory for execution. | It selects from among the processes that are ready to execute. | Process can be reintroduced into the meat and its execution can be continued. |

**Q3** – Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run.

A dispatcher is a special program which comes into play after the scheduler. When the scheduler completes its job of selecting a process, it is the dispatcher which takes that process to the desired state/queue.

**Q4** – Advantages -

* Each process is served by the CPU for a fixed time quantum, so all processes are given the same priority.
* Starvation doesn't occur because, for each round robin cycle, every process is given a fixed time to execute. No process is left behind.

Disadvantages -

* The throughput in RR largely depends on the choice of the length of the time quantum. If time quantum is too large it behaves as FCFS. If time quantum is too short much of the time is spent in process switching and hence low throughput.
* Also here one cannot assign priority to any process which can be a drawback.

**Q5** - Round-robin scheduling can be applied to other scheduling problems, such as data packet scheduling in computer networks.

#include<stdio.h>

#include<conio.h>

void main()

{

int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];

float avg\_wt, avg\_tat;

printf(" Total number of process in the system: ");

scanf("%d", &NOP);

y = NOP;

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

{

printf("\n Enter the Arrival and Burst time of the Process%d\n", i+1);

printf(" Arrival time is: \t");

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

printf(" \nBurst time is: \t");

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

temp[i] = bt[i];

}

printf("Enter the Time Quantum for the process: \t");

scanf("%d", &quant);

printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");

for(sum=0, i = 0; y!=0; )

{

if(temp[i] <= quant && temp[i] > 0)

{

sum = sum + temp[i];

temp[i] = 0;

count=1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - quant;

sum = sum + quant;

}

if(temp[i]==0 && count==1)

{

y--;

printf("\n Process %d \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);

wt = wt+sum-at[i]-bt[i];

tat = tat+sum-at[i];

count =0;

}

if(i==NOP-1)

{

i=0;

}

else if(at[i+1]<=sum)

{

i++;

}

else

{

i=0;

}

}

avg\_wt = wt \* 1.0/NOP;

avg\_tat = tat \* 1.0/NOP;

printf("\n Average Turn Around Time: \t%f", avg\_wt);

printf("\n Average Waiting Time: \t%f", avg\_tat);

getch();

}

Text

Description automatically generated

**Experiment No: 9**

|  |
| --- |
| **Student Name and Roll Number:** Namit Kumar |
| **Semester /Section:** V/FS-A-1 |
| **Link to Code:** https://github.com/NamitKumar16/OS |
| **Date:** 20th October 2021 |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective:** Write a program to implement reader/writer problem using semaphore |
| **Program Outcome**  The students will understand the reader/writer problem using semaphore |
| **Problem Statement:**  Write a program to implement reader/writer problem using semaphore |
| **Background Study:** There is a shared resource which should be accessed by multiple processes. There are two types of processes in this context. They are reader and writer. Any number of readers can read from the shared resource simultaneously, but only one writer can write to the shared resource. When a writer is writing data to the resource, no other process can access the resource. A writer cannot write to the resource if there are non-zero number of readers accessing the resource at that time. |
| **Question Bank:**   1. An un-interruptible unit is known as \_\_\_\_\_\_\_\_\_\_\_\_ a) single b) atomic c) static d) none of the mentioned 2. TestAndSet instruction is executed \_\_\_\_\_\_\_\_\_\_\_\_ a) after a particular process b) periodically c) atomically d) none of the mentioned 3. Semaphore is a/an \_\_\_\_\_\_\_ to solve the critical section problem. a) hardware for a system b) special program for a system c) integer variable d) none of the mentioned 4. What are the two atomic operations permissible on semaphores? a) wait b) stop c) hold d) none of the mentioned 5. When several processes access the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place is called \_\_\_\_\_\_\_\_ a) dynamic condition b) race condition c) essential condition d) critical condition |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

#include<stdio.h>

#include<pthread.h>

#include<unistd.h>

#include<stdlib.h>

pthread\_mutex\_t wr,mutex;

int a = 10,readcount=0;

void \* reader(void \*arg)

{

long int num;

num=(long int) arg;

pthread\_mutex\_lock(&mutex);

readcount++;

pthread\_mutex\_unlock(&mutex);

if(readcount==1)

{

pthread\_mutex\_lock(&wr);

}

printf("\nReader %ld is in critical section",num);

printf("\nReader %ld is reading data %d",num,a);

sleep(1);

pthread\_mutex\_lock(&mutex);

readcount--;

pthread\_mutex\_unlock(&mutex);

if(readcount==0)

{

pthread\_mutex\_unlock(&wr);

}

printf("\nReader %ld left criticial section",num);

}

void \* writer(void \*arg)

{

long int num;

num=(long int) arg;

pthread\_mutex\_lock(&wr);

printf("\nWriter %ld is in critical section",num);

printf("\nWriter %ld have written data as %d",num,++a);

sleep(1);

pthread\_mutex\_unlock(&wr);

printf("\nWriter %ld left critical section",num);

}

int main()

{

pthread\_t r[10],w[10];

long int i,j;

int nor,now;

pthread\_mutex\_init(&wr,NULL);

pthread\_mutex\_init(&mutex,NULL);

printf("Enter number of readers and writers\t");

scanf("%d %d",&nor,&now);

for(i=0;i<nor;i++)

{

pthread\_create(&r[i],NULL,reader,(void \*)i);

}

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

{

pthread\_create(&w[j],NULL,writer,(void \*)j);

}

for(i=0;i<nor;i++)

{

pthread\_join(r[i],NULL);

}

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

{

pthread\_join(w[j],NULL);

}

return 0;

}

Text

Description automatically generated

**Experiment No: 10**

|  |
| --- |
| **Student Name and Roll Number:** Namit Kumar |
| **Semester /Section:** V/FSA1 |
| **Link to Code:** https://github.com/NamitKumar16/OS |
| **Date:** 10th November 2021 |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective:**  Write a program to implement Dining Philosopher problem using semaphore |
| **Outcome:**  The students will understand the problem of synchronization among processes and its solution through Dining Philosopher problem using semaphore |
| **Problem Statement:**  Write a program to implement Dining Philosopher problem using semaphore |
| **Background Study:**  Five philosophers, spend their time thinking and eating spaghetti. They eat at a round table with five individual seats. For eating each philosopher needs two forks (the resources). There are five forks on the table, one left and one right of each seat. When a philosopher cannot grab both forks it sits and waits. Eating takes random time, then the philosopher puts the forks down and leaves the dining room. After spending some random time thinking he again becomes hungry, and the circle repeats itself. |
| **Question Bank:**   1. Which one of the following is a synchronization tool? a) thread b) pipe c) semaphore d) socket 2. A semaphore is a shared integer variable \_\_\_\_\_\_\_\_\_\_ a) that can not drop below zero b) that can not be more than zero c) that can not drop below one d) that can not be more than one 3. The bounded buffer problem is also known as \_\_\_\_\_\_\_\_\_\_\_\_ a) Readers – Writers problem b) Dining – Philosophers problem c) Producer – Consumer problem d) None of the mentioned 4. In the bounded buffer problem \_\_\_\_\_\_\_\_\_\_\_\_ a) there is only one buffer b) there are n buffers ( n being greater than one but finite) c) there are infinite buffers d) the buffer size is bounded 5. To ensure difficulties do not arise in the readers – writers problem \_\_\_\_\_\_\_ are given exclusive access to the shared object. a) readers b) writers c) readers and writers d) none of the mentioned |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

sem\_t mutex;

sem\_t S[N];

void test(int phnum)

{

if (state[phnum] == HUNGRY

&& state[LEFT] != EATING

&& state[RIGHT] != EATING) {

// state that eating

state[phnum] = EATING;

sleep(2);

printf("Philosopher %d takes fork %d and %d\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is Eating\n", phnum + 1);

// sem\_post(&S[phnum]) has no effect

// during takefork

// used to wake up hungry philosophers

// during putfork

sem\_post(&S[phnum]);

}

}

// take up chopsticks

void take\_fork(int phnum)

{

sem\_wait(&mutex);

// state that hungry

state[phnum] = HUNGRY;

printf("Philosopher %d is Hungry\n", phnum + 1);

// eat if neighbours are not eating

test(phnum);

sem\_post(&mutex);

// if unable to eat wait to be signalled

sem\_wait(&S[phnum]);

sleep(1);

}

// put down chopsticks

void put\_fork(int phnum)

{

sem\_wait(&mutex);

// state that thinking

state[phnum] = THINKING;

printf("Philosopher %d putting fork %d and %d down\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is thinking\n", phnum + 1);

test(LEFT);

test(RIGHT);

sem\_post(&mutex);

}

void\* philospher(void\* num)

{

while (1) {

int\* i = num;

sleep(1);

take\_fork(\*i);

sleep(0);

put\_fork(\*i);

}

}

int main()

{

int i;

pthread\_t thread\_id[N];

// initialize the semaphores

sem\_init(&mutex, 0, 1);

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

sem\_init(&S[i], 0, 0);

for (i = 0; i < N; i++) {

// create philosopher processes

pthread\_create(&thread\_id[i], NULL,

philospher, &phil[i]);

printf("Philosopher %d is thinking\n", i + 1);

}

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

pthread\_join(thread\_id[i], NULL);

}

A screenshot of a computer

Description automatically generated with medium confidence

**Experiment No:11**

|  |
| --- |
| **Student Name and Roll Number:** Namit Kumar 19CSU185 |
| **Semester /Section:** V/FSA1 |
| **Link to Code:** https://github.com/NamitKumar16/OS |
| **Date:** 17th November, 2021 |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective:**  Write a program to implement Banker’s algorithm for deadlock avoidance. |
| **Outcome:**  The students will understand how system handles deadlock using Banker’s algorithm for deadlock avoidance |
| **Problem Statement:**  Write a program to implement Banker’s algorithm for deadlock avoidance. |
| **Background Study:**  The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue. Banker’s algorithm is named so because it is used in banking system to check whether loan can be sanctioned to a person or not. Suppose there are n number of account holders in a bank and the total sum of their money is S. If a person applies for a loan then the bank first subtracts the loan amount from the total money that bank has and if the remaining amount is greater than S then only the loan is sanctioned. It is done because if all the account holders comes to withdraw their money then the bank can easily do it. |
| **Question Bank:**   1. Each request requires that the system consider the \_\_\_\_\_\_\_\_\_\_\_\_\_ to decide whether the current request can be satisfied or must wait to avoid a future possible deadlock. a) resources currently available b) processes that have previously been in the system c) resources currently allocated to each process d) future requests and releases of each process 2. Given a priori information about the \_\_\_\_\_\_\_\_ number of resources of each type that maybe requested for each process, it is possible to construct an algorithm that ensures that the system will never enter a deadlock state. a) minimum b) average c) maximum d) approximate 3. A deadlock avoidance algorithm dynamically examines the \_\_\_\_\_\_\_\_\_\_ to ensure that a circular wait condition can never exist. a) resource allocation state b) system storage state c) operating system d) resources 4. A state is safe, if \_\_\_\_\_\_\_\_\_\_\_\_ a) the system does not crash due to deadlock occurrence b) the system can allocate resources to each process in some order and still avoid a deadlock c) the state keeps the system protected and safe d) all of the mentioned 5. The two ways of aborting processes and eliminating deadlocks are \_\_\_\_\_\_\_\_\_\_\_\_ a) Abort all deadlocked processes b) Abort all processes c) Abort one process at a time until the deadlock cycle is eliminated d) All of the mentioned |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

// Banker's Algorithm

#include <stdio.h>

int main()

{

int n, m, i, j, k;

n = 5;

m = 3;

int alloc[5][3] = { { 0, 1, 0 },

{ 2, 0, 0 },

{ 3, 0, 2 },

{ 2, 1, 1 },

{ 0, 0, 2 } };

int max[5][3] = { { 7, 5, 3 },

{ 3, 2, 2 },

{ 9, 0, 2 },

{ 2, 2, 2 },

{ 4, 3, 3 } };

int avail[3] = { 3, 3, 2 };

int f[n], ans[n], ind = 0;

for (k = 0; k < n; k++) {

f[k] = 0;

}

int need[n][m];

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

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

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

}

int y = 0;

for (k = 0; k < 5; k++) {

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

if (f[i] == 0) {

int flag = 0;

for (j = 0; j < m; j++) {

if (need[i][j] > avail[j]){

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

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

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

printf("SAFE SEQUENCE\n");

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

printf(" P%d ->", ans[i]);

printf(" P%d", ans[n - 1]);

return (0);

}

Text

Description automatically generated

**Experiment No: 12**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective:**  Write a program for page replacement policy using a) LRU b) FIFO c) Optimal. |
| **Outcome:**  The students will understand concept of memory management through various page replacement policy using a) LRU b) FIFO c) Optimal. |
| **Problem Statement:**  Write a program to implement page replacement policy using a) LRU b) FIFO c) Optimal. |
| **Background Study:**  In operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults. |
| **Question Bank:**  1. The main memory accommodates \_\_\_\_\_\_\_\_\_\_\_\_ a) operating system b) cpu c) user processes d) all of the mentioned  2. What is the operating system? a) in the low memory b) in the high memory c) either low or high memory (depending on the location of interrupt vector) d) none of the mentioned  3. In contiguous memory allocation \_\_\_\_\_\_\_\_\_\_\_\_ a) each process is contained in a single contiguous section of memory b) all processes are contained in a single contiguous section of memory c) the memory space is contiguous d) none of the mentioned  4. The relocation register helps in \_\_\_\_\_\_\_\_\_\_\_\_ a) providing more address space to processes b) a different address space to processes c) to protect the address spaces of processes d) none of the mentioned  5. With relocation and limit registers, each logical address must be \_\_\_\_\_\_\_ the limit register. a) less than b) equal to c) greater than d) none of the mentioned |