

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

**Operating Systems Lab Manual**

**CSL 303**

**Dr. Priyanka Vasisth**

**Dr. Divya Sharma**

**Ms. Poonam Chaudhary**



Department of Computer Science and Engineering

NorthCap University, Gurugram- 122001, India

Session 2019-20

*Published by:*

**School of Engineering and Technology**

**Department of Computer Science & Engineering**

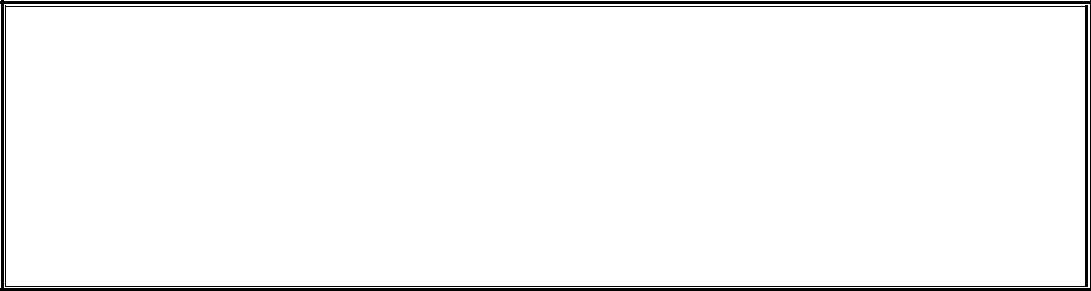
**The NorthCap University Gurugram**

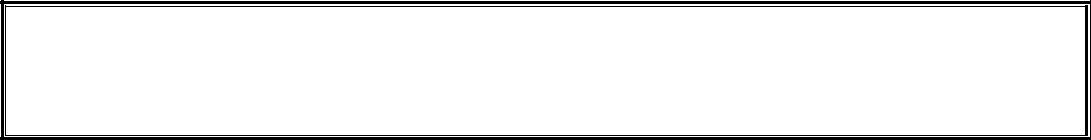
**• Laboratory Manual is for Internal Circulation only**

**© 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 |
| 1 | Basics of Linux | Explain the structure of Linux Operating System |
| 2 | Installation of Ubuntu Operating system |
| 1 | Shell Programs | Write a shell program to find factorial of a number. |
| 2 | Write a shell program to find gross salary of an employee. |
| 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 |
| 4 | Write a shell program to find Fibonacci series. |
| 5 | Write a shell program to find largest of three numbers. |
| 6 | Write a shell program to find average of N numbers |
| 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 |
| 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. |
| Implement the following CPU scheduling Algorithms.  i) Round Robin  ii) Priority Based |
| 9 | Deadlock Management  Technique | Write a program to simulate Bankers algorithm for the purpose of deadlock avoidance |
| 10 | Page Replacement  Algorithms | Write a C program to simulate page replacement algorithms  a) FIFO b) LRU c) LFU |
| 11 | Write a C program to simulate page replacement algorithms  a) Optimal |

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 Systems**

**(CSL 303)**

Lab Practical Report



Faculty name Student name-Hitesh Sharma

Roll No.: 18CSU086

Semester: 5

Group: 2

Department of Computer Science and Engineering

NorthCap University, Gurugram- 122001, India

Session 2019-20

**INDEX**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **S.No** | **Experiment** | **Page No.** | **Date of Experiment** | **Date of Submission** | **Marks** | **CO Covered** | **Signature** |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**Experiment No: 1**

Student Name and Roll Number: Hitesh 18csu086

Semester /Section: 5/A

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

Write a shell program to find factorial of a number.

**Program Outcome**

* The students will understand the shell program to find factorial of a number..

**Problem Statement**

Write a shell program to find factorial of a number.

**Background Study:**

A factorial is a function that multiplies a number by every number below it. For example 5!= 5\*4\*3\*2\*1=120. The function is used, among other things, to find the number of way “n” objects can be arranged.

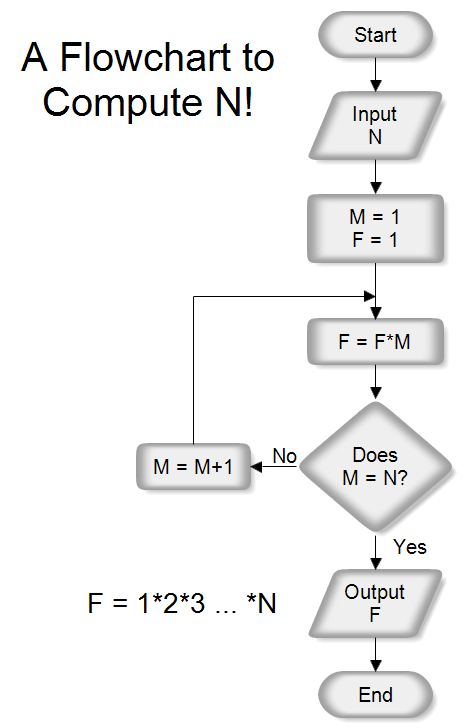
**Algorithm/ flowchart**

1. Get a number

2. Use for loop or while loop to compute the factorial by using the below formula

3. fact(n) = n \* n-1 \* n-2 \* .. 1

4. Display the result.



**Code**

echo "Enter a number"

read num

fact=1

while [ $num -gt 1 ]

do

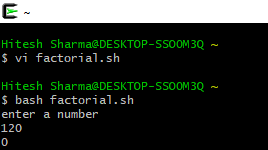
fact=$((fact \* num)) #fact = fact \* num

num=$((num - 1)) #num = num - 1

done

echo $fact

**Output: Screenshots**



**Experiment No: 2**

Student Name and Roll Number: Hitesh 18csu086

Semester /Section: 5/A

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

Write a shell program to find Fibonacci series.

**Program Outcome**

* The students will understand a shell program to find Fibonacci series..

**Problem Statement**

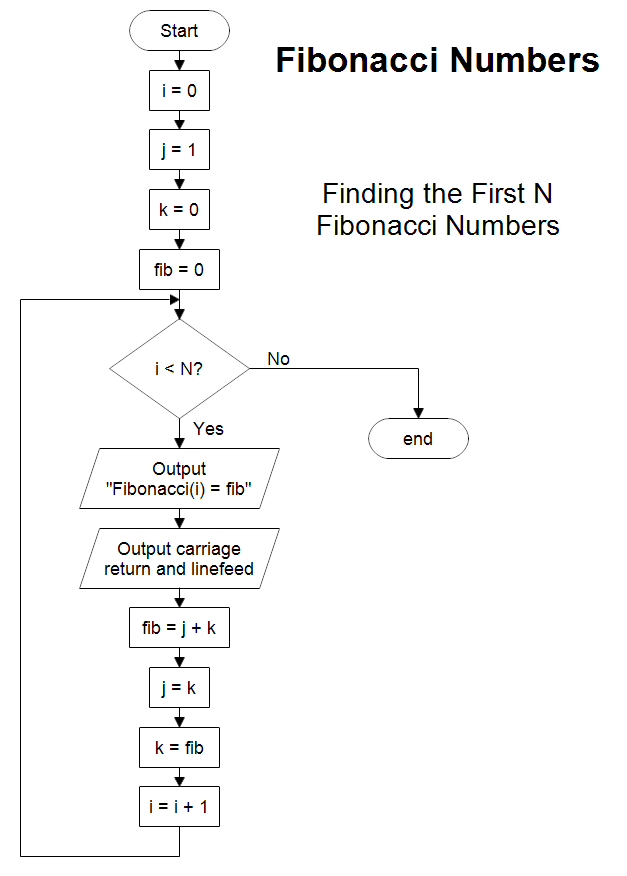
Write a shell program to find Fibonacci series.

**Background Study:**

The Fibonacci sequence is one of the most famous formulas in mathematics.

Each number in the sequence is the sum of the two numbers that precede it. So, the sequence goes: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, and so on. The mathematical equation describing it is *Xn+2= Xn+1 + Xn*

**Algorithm/ flowchart**



**Code**

echo "Enter a number"

read num

fact=1

while [ $num -gt 1 ]

do

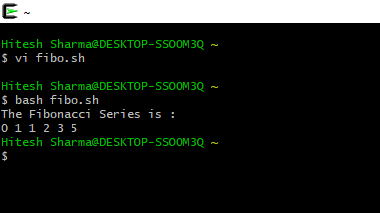
fact=$((fact \* num)) #fact = fact \* num

num=$((num - 1)) #num = num - 1

done

echo $fact

**Output: Screenshots**



**Experiment No: 3**

Student Name and Roll Number: Hitesh 18csu086

Semester /Section: 5/A

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

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

**Program Outcome**

The students will understand Write a shell program to find gross salary of an employee.

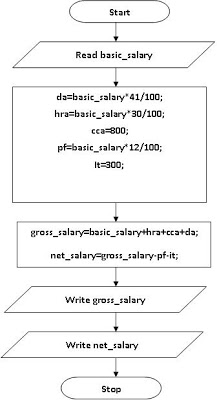
**Problem Statement**

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

**Background Study:**

To find the gross salary we have to take all the value of allowances and other things that we have to reduce or to increment the things accordingly.

**Algorithm/ flowchart**



**Code**

echo "Enter name of the employee file:"

read filename

j=1

for i in `cat $filename`

do

echo "EMPLOYEE $j"

echo "==========="

name=$(echo $i | cut -d "|" -f2 )

basic=$(echo $i | cut -d "|" -f7 )

echo "Name :"$name

echo "Basic salary :"$basic

hra=$(expr $basic \\* 10)

hra=$(expr $hra / 100)

da=$(expr $basic \\* 15)

da=$(expr $da / 100)

echo "HRA :"$hra

echo "DA :"$da

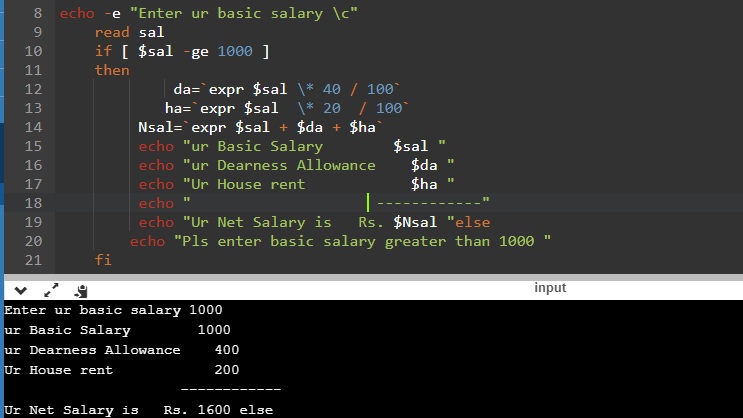
gross=$(expr $basic + $hra + $da)

echo "GROSS :"$gross

j=$(expr $j + 1)

 done

**Output: Screenshots**



**Experiment No: 4**

Student Name and Roll Number: Hitesh 18csu086

Semester /Section: 5/A

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

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

**Program Outcome**

The students will understand 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

**Problem Statement**

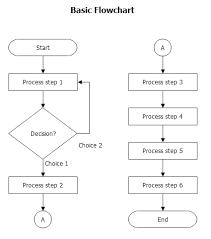
Write a shell program to display the menu and execute instructions accordingly

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

**Background Study:**

By this program we can execute the file list that we have and to see the process status , date for that and user in program

**Algorithm/ flowchart**

****

**Code**

Echo “ select option”

Echo “list of files”

Echo “ process status”

Echo “ date”

Echo “ users in program”

Echo “quit”

Read n

If [$n -eq 1]

Then

Ls

Elif [$n -eq 2]

Then

Ps -aux

Elif [ $n -eq 3]

Then

Date + “%m-%d-%y”

Elif [$n -eq 4]

Then

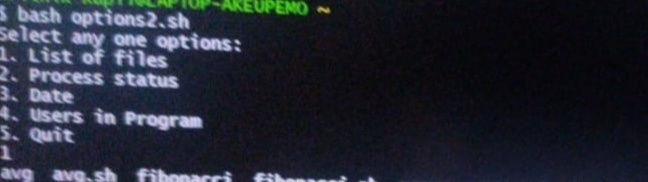
Users

Else

Echo “exiting..”

fi

**Output: Screenshots**



**Experiment No: 5**

Student Name and Roll Number: Hitesh 18csu086

Semester /Section: 5/A

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

Write a shell program to find largest of three numbers.

**Program Outcome**

Write a shell program to find largest of three numbers.

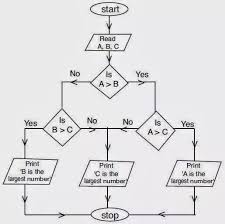
**Problem Statement**

Write a shell program to find largest of three numbers.

**Background Study**:

To find the largest of three number we can check if a particular number is greater then other else check for other numbers also if found the greatest then report for the answer .

**Algorithm/ flowchart**

****

1. Get three numbers. Say num1, num2, num2

2. If (num1 > num2) and (num1 > num3)

     echo value of num1

3. elif(num2 > num1) and (num2 > num3)

     echo value of num2

4. Otherwise,

     echo value of num3

**Code**

#!bin/bash

read a b c

if [ $a -eq $b -a $a -eq $c ]; then

echo "All the three numbers are equal"

elif [[ $a -eq $b || $b -eq $c || $c -eq $a ]]; then

echo "I cannot figure out which number is largest"

else

if [ $a -gt $b -a $a -gt $c ]; then

echo "$a is biggest number"

elif [ $b -gt $a -a $b -gt $c ]; then

echo "$b is biggest number"

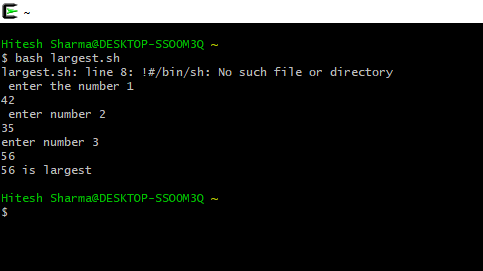
elif [ $c -gt $a -a $c -gt $b ]; then

echo "$c is biggest number"

fi

fi

**Output: Screenshots**



**Experiment No: 6**

Student Name and Roll Number: Hitesh 18csu086

Semester /Section: 5/A

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

Write a shell program to find average of N numbers

**Program Outcome**

Student will be able to understand Write a shell program to find average of N numbers

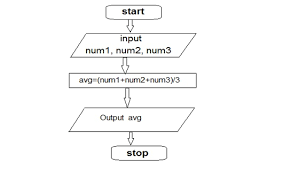
**Problem Statement**

Write a shell program to find average of N numbers

**Background Study**:

To find the average of N number we can check if a enter values that we have in our data and then check for the total value entered in the data then we can do the sigma of value divide by the total values .

**Algorithm/ flowchart**

****

**Code**

# Total numbers

n=5

# copying the value of n

m=$n

# initialized sum by 0

sum=0

# array initialized with

# some numbers

array=(1 2 3 4 5)

# loop until n is greater

# than 0

while [ $n -gt 0 ]

do

# copy element in a

# temp variable

num=${array[`expr $n - 1`]}

# add them to sum

sum=`expr $sum + $num`

# decrement count of n

n=`expr $n - 1`

done

# displaying the average

# by piping with bc command

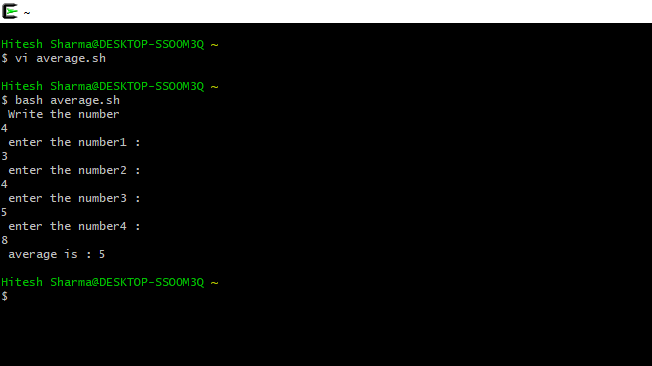
# bc is bash calculator

# command

avg=`echo "$sum / $m" | bc -l`

printf '%0.3f' "$avg"

**Output: Screenshots**



**Experiment No: 7**

Student Name and Roll Number: Hitesh 18csu086

Semester /Section: 5/A

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

To familiarize the students about CPU scheduling Algorithms.

**Program Outcome**

* The students will understand the First-cum-first-serve

**Problem Statement**

Implement the following CPU scheduling Algorithms.

1. FCFS

**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.

**Algorithm/ flowchart**

Input the processes along with their burst time

Find waiting time for all processes

As first process that comes need not to wait

Wt[i] = bt[i-1] + wt[i-1]

Turn around time = waiting time +burst time

Average waiting time = total waiting time / no. of processes

Average turn around time = total turn around time / no. of processes

**Code without Arrival Time**

# Python3 program for implementation

# of FCFS scheduling

# Function to find the waiting

# time for all processes

def findWaitingTime(processes, n,

bt, wt):

# waiting time for

# first process is 0

wt[0] = 0

# calculating waiting time

for i in range(1, n ):

wt[i] = bt[i - 1] + wt[i - 1]

# Function to calculate turn

# around time

def findTurnAroundTime(processes, n,

bt, wt, tat):

# calculating turnaround

# time by adding bt[i] + wt[i]

for i in range(n):

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

# Function to calculate

# average time

def findavgTime( processes, n, bt):

wt = [0] \* n

tat = [0] \* n

total\_wt = 0

total\_tat = 0

# Function to find waiting

# time of all processes

findWaitingTime(processes, n, bt, wt)

# Function to find turn around

# time for all processes

findTurnAroundTime(processes, n,

bt, wt, tat)

# Display processes along

# with all details

print( "Processes Burst time " +

" Waiting time " +

" Turn around time")

# Calculate total waiting time

# and total turn around time

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" " + str(i + 1) + "\t\t" +

str(bt[i]) + "\t " +

str(wt[i]) + "\t\t " +

str(tat[i]))

print( "Average waiting time = "+

str(total\_wt / n))

print("Average turn around time = "+

str(total\_tat / n))

# Driver code

if \_\_name\_\_ =="\_\_main\_\_":

# process id's

processes = [ 1, 2, 3]

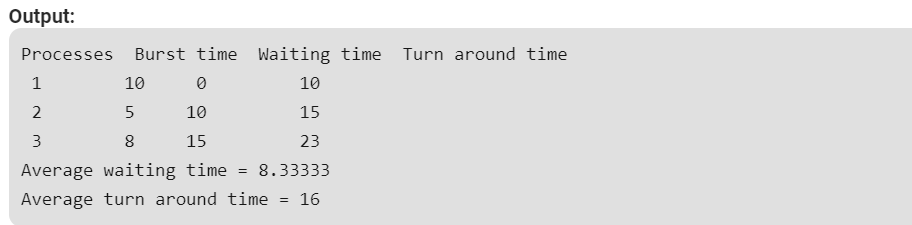
n = len(processes)

# Burst time of all processes

burst\_time = [10, 5, 8]

findavgTime(processes, n, burst\_time)

**Output: Screenshots**



**Code with Arrival Time**

# Python3 program for implementation of FCFS

# scheduling with different arrival time

# Function to find the waiting time

# for all processes

def findWaitingTime(processes, n, bt, wt, at):

service\_time = [0] \* n

service\_time[0] = 0

wt[0] = 0

# calculating waiting time

for i in range(1, n):

# Add burst time of previous processes

service\_time[i] = (service\_time[i - 1] +

bt[i - 1])

# Find waiting time for current

# process = sum - at[i]

wt[i] = service\_time[i] - at[i]

# If waiting time for a process is in

# negative that means it is already

# in the ready queue before CPU becomes

# idle so its waiting time is 0

if (wt[i] < 0):

wt[i] = 0

# Function to calculate turn around time

def findTurnAroundTime(processes, n, bt, wt, tat):

# Calculating turnaround time by

# adding bt[i] + wt[i]

for i in range(n):

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

# Function to calculate average waiting

# and turn-around times.

def findavgTime(processes, n, bt, at):

wt = [0] \* n

tat = [0] \* n

# Function to find waiting time

# of all processes

findWaitingTime(processes, n, bt, wt, at)

# Function to find turn around time for

# all processes

findTurnAroundTime(processes, n, bt, wt, tat)

# Display processes along with all details

print("Processes Burst Time Arrival Time Waiting",

"Time Turn-Around Time Completion Time \n")

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

compl\_time = tat[i] + at[i]

print(" ", i + 1, "\t\t", bt[i], "\t\t", at[i],

"\t\t", wt[i], "\t\t ", tat[i], "\t\t ", compl\_time)

print("Average waiting time = %.5f "%(total\_wt /n))

print("\nAverage turn around time = ", total\_tat / n)

# Driver code

if \_\_name\_\_ =="\_\_main\_\_":

# Process id's

processes = [1, 2, 3]

n = 3

# Burst time of all processes

burst\_time = [5, 9, 6]

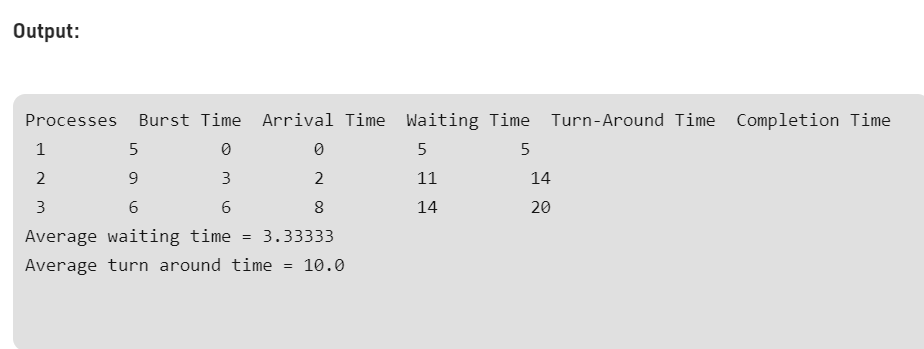
# Arrival time of all processes

arrival\_time = [0, 3, 6]

findavgTime(processes, n, burst\_time,

arrival\_time)

**Output: Screenshots**



**Experiment No: 8**

Student Name and Roll Number: Hitesh 18csu086

Semester /Section: 5/A

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

To familiarize the students about CPU scheduling Algorithms.

**Program Outcome**

* The students will understand the Shortest Job First Algorithm , Shortest Remaining Job First

**Problem Statement**

Implement the following CPU scheduling Algorithms.

1. SJF
2. SRJF

**Background Study:**

**SJF**

* This is the best approach to minimize waiting time.
* This is used in [Batch Systems](https://www.studytonight.com/operating-system/types-of-os).
* It is of two types:
  1. Non Pre-emptive
  2. Pre-emptive
* To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.
* This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all)

**Algorithm/ flowchart**

Sort all the process according to the arrival time.

Then select that process which has minimum arrival time and minimum Burst time.

After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

**Code for SJF**

// Java program to implement Shortest Job first with Arrival Time

import java.util.\*;

class GFG {

static int[][] mat = new int[10][6];

static void arrangeArrival(int num, int[][] mat) {

for (int i = 0; i < num; i++) {

for (int j = 0; j < num - i - 1; j++) {

if (mat[j][1] > mat[j + 1][1]) {

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

int temp = mat[j][k];

mat[j][k] = mat[j + 1][k];

mat[j + 1][k] = temp;

}

}

}

}

}

static void completionTime(int num, int[][] mat) {

int temp, val = -1;

mat[0][3] = mat[0][1] + mat[0][2];

mat[0][5] = mat[0][3] - mat[0][1];

mat[0][4] = mat[0][5] - mat[0][2];

for (int i = 1; i < num; i++) {

temp = mat[i - 1][3];

int low = mat[i][2];

for (int j = i; j < num; j++) {

if (temp >= mat[j][1] && low >= mat[j][2]) {

low = mat[j][2];

val = j;

}

}

mat[val][3] = temp + mat[val][2];

mat[val][5] = mat[val][3] - mat[val][1];

mat[val][4] = mat[val][5] - mat[val][2];

for (int k = 0; k < 6; k++) {

int tem = mat[val][k];

mat[val][k] = mat[i][k];

mat[i][k] = tem;

}

}

}

// Driver Code

public static void main(String[] args) {

int num;

Scanner sc = new Scanner(System.in);

System.out.println("Enter number of Process: ");

num = sc.nextInt();

System.out.println("...Enter the process ID...");

for (int i = 0; i < num; i++) {

System.out.println("...Process " + (i + 1) + "...");

System.out.println("Enter Process Id: ");

mat[i][0] = sc.nextInt();

System.out.println("Enter Arrival Time: ");

mat[i][1] = sc.nextInt();

System.out.println("Enter Burst Time: ");

mat[i][2] = sc.nextInt();

}

System.out.println("Before Arrange...");

System.out.println("Process ID\tArrival Time\tBurst Time");

for (int i = 0; i < num; i++) {

System.out.printf("%d\t\t%d\t\t%d\n",

mat[i][0], mat[i][1], mat[i][2]);

}

arrangeArrival(num, mat);

completionTime(num, mat);

System.out.println("Final Result...");

System.out.println("Process ID\tArrival Time\tBurst" +

" Time\tWaiting Time\tTurnaround Time");

for (int i = 0; i < num; i++) {

System.out.printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",

mat[i][0], mat[i][1], mat[i][2], mat[i][4], mat[i][5]);

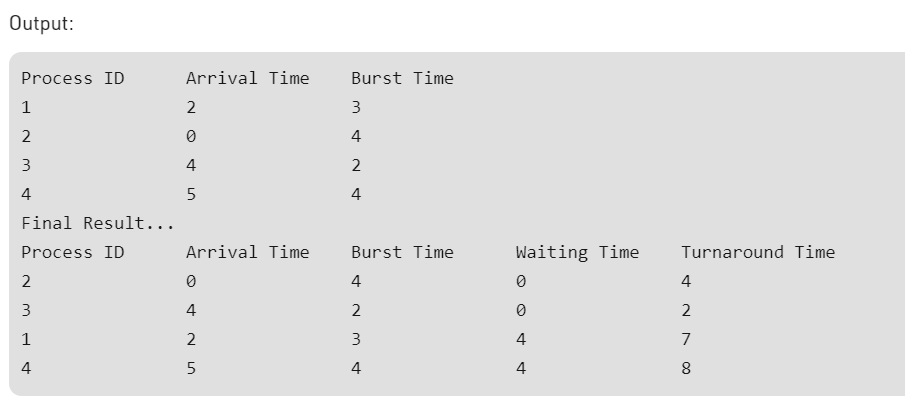
}

sc.close();

}

}

**Output: Screenshots**



**Code for SRJF**

# Python3 program to implement Shortest Remaining Time First

# Shortest Remaining Time First (SRTF)

# Function to find the waiting time

# for all processes

def findWaitingTime(processes, n, wt):

rt = [0] \* n

# Copy the burst time into rt[]

for i in range(n):

rt[i] = processes[i][1]

complete = 0

t = 0

minm = 999999999

short = 0

check = False

# Process until all processes gets

# completed

while (complete != n):

# Find process with minimum remaining

# time among the processes that

# arrives till the current time`

for j in range(n):

if ((processes[j][2] <= t) and

(rt[j] < minm) and rt[j] > 0):

minm = rt[j]

short = j

check = True

if (check == False):

t += 1

continue

# Reduce remaining time by one

rt[short] -= 1

# Update minimum

minm = rt[short]

if (minm == 0):

minm = 999999999

# If a process gets completely

# executed

if (rt[short] == 0):

# Increment complete

complete += 1

check = False

# Find finish time of current

# process

fint = t + 1

# Calculate waiting time

wt[short] = (fint - proc[short][1] -

proc[short][2])

if (wt[short] < 0):

wt[short] = 0

# Increment time

t += 1

# Function to calculate turn around time

def findTurnAroundTime(processes, n, wt, tat):

# Calculating turnaround time

for i in range(n):

tat[i] = processes[i][1] + wt[i]

# Function to calculate average waiting

# and turn-around times.

def findavgTime(processes, n):

wt = [0] \* n

tat = [0] \* n

# Function to find waiting time

# of all processes

findWaitingTime(processes, n, wt)

# Function to find turn around time

# for all processes

findTurnAroundTime(processes, n, wt, tat)

# Display processes along with all details

print("Processes Burst Time Waiting",

"Time Turn-Around Time")

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" ", processes[i][0], "\t\t",

processes[i][1], "\t\t",

wt[i], "\t\t", tat[i])

print("\nAverage waiting time = %.5f "%(total\_wt /n) )

print("Average turn around time = ", total\_tat / n)

# Driver code

if \_\_name\_\_ =="\_\_main\_\_":

# Process id's

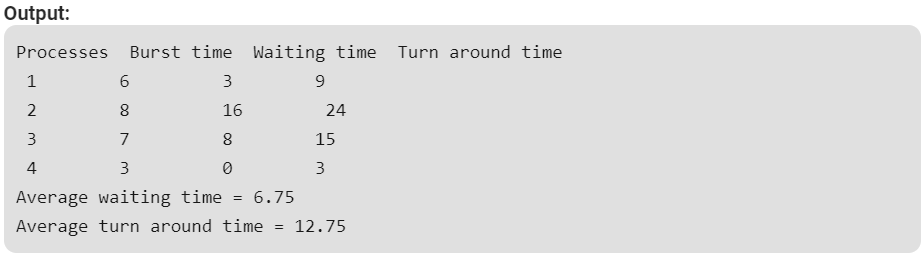
proc = [[1, 6, 1], [2, 8, 1],

[3, 7, 2], [4, 3, 3]]

n = 4

findavgTime(proc, n)

**Output: Screenshots**



**Experiment No: 9**

Student Name and Roll Number: Hitesh 18csu086

Semester /Section: 5/A

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

To familiarize the students about CPU scheduling Algorithms.

**Program Outcome**

* The students will understand the Priority Scheduling with Arrival and without Arrival Time

**Problem Statement**

Implement the following CPU scheduling Algorithms.

Priority Scheduling

**Background Study:**

**Priority**

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

**Algorithm/ flowchart**

First input the processes with their burst time

And priority.

Sort the processes, burst time and priority according to the priority.

Now simply apply FCFS algorithm

**Code for Priority Without Arrival Time**

# Python3 program for implementation of

# Priority Scheduling

# Function to find the waiting time

# for all processes

def findWaitingTime(processes, n, wt):

wt[0] = 0

# calculating waiting time

for i in range(1, n):

wt[i] = processes[i - 1][1] + wt[i - 1]

# Function to calculate turn around time

def findTurnAroundTime(processes, n, wt, tat):

# Calculating turnaround time by

# adding bt[i] + wt[i]

for i in range(n):

tat[i] = processes[i][1] + wt[i]

# Function to calculate average waiting

# and turn-around times.

def findavgTime(processes, n):

wt = [0] \* n

tat = [0] \* n

# Function to find waiting time

# of all processes

findWaitingTime(processes, n, wt)

# Function to find turn around time

# for all processes

findTurnAroundTime(processes, n, wt, tat)

# Display processes along with all details

print("\nProcesses Burst Time Waiting",

"Time Turn-Around Time")

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" ", processes[i][0], "\t\t",

processes[i][1], "\t\t",

wt[i], "\t\t", tat[i])

print("\nAverage waiting time = %.5f "%(total\_wt /n))

print("Average turn around time = ", total\_tat / n)

def priorityScheduling(proc, n):

# Sort processes by priority

proc = sorted(proc, key = lambda proc:proc[2],

reverse = True);

print("Order in which processes gets executed")

for i in proc:

print(i[0], end = " ")

findavgTime(proc, n)

# Driver code

if \_\_name\_\_ =="\_\_main\_\_":

# Process id's

proc = [[1, 10, 1],

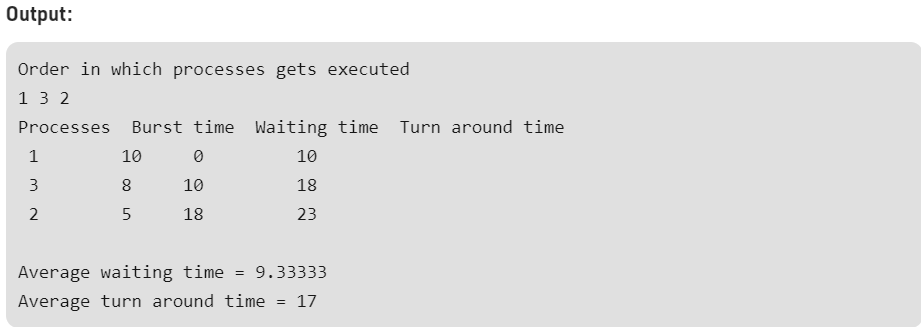
[2, 5, 0],

[3, 8, 1]]

n = 3

priorityScheduling(proc, n)

**Output: Screenshots**



**Code for Priority Without Arrival Time**

# Python3 implementation for Priority Scheduling with

# Different Arrival Time priority scheduling

"""1. sort the processes according to arrival time

2. if arrival time is same the acc to priority

3. apply fcfs """

totalprocess = 5

proc = []

for i in range(5):

l = []

for j in range(4):

l.append(0)

proc.append(l)

# Using FCFS Algorithm to find Waiting time

def get\_wt\_time( wt):

# declaring service array that stores

# cumulative burst time

service = [0] \* 5

# Initilising initial elements

# of the arrays

service[0] = 0

wt[0] = 0

for i in range(1, totalprocess):

service[i] = proc[i - 1][1] + service[i - 1]

wt[i] = service[i] - proc[i][0] + 1

# If waiting time is negative,

# change it o zero

if(wt[i] < 0) :

wt[i] = 0

def get\_tat\_time(tat, wt):

# Filling turnaroundtime array

for i in range(totalprocess):

tat[i] = proc[i][1] + wt[i]

def findgc():

# Declare waiting time and

# turnaround time array

wt = [0] \* 5

tat = [0] \* 5

wavg = 0

tavg = 0

# Function call to find waiting time array

get\_wt\_time(wt)

# Function call to find turnaround time

get\_tat\_time(tat, wt)

stime = [0] \* 5

ctime = [0] \* 5

stime[0] = 1

ctime[0] = stime[0] + tat[0]

# calculating starting and ending time

for i in range(1, totalprocess):

stime[i] = ctime[i - 1]

ctime[i] = stime[i] + tat[i] - wt[i]

print("Process\_no\tStart\_time\tComplete\_time",

"\tTurn\_Around\_Time\tWaiting\_Time")

# display the process details

for i in range(totalprocess):

wavg += wt[i]

tavg += tat[i]

print(proc[i][3], "\t\t", stime[i],

"\t\t", end = " ")

print(ctime[i], "\t\t", tat[i], "\t\t\t", wt[i])

# display the average waiting time

# and average turn around time

print("Average waiting time is : ", end = " ")

print(wavg / totalprocess)

print("average turnaround time : " , end = " ")

print(tavg / totalprocess)

# Driver code

if \_\_name\_\_ =="\_\_main\_\_":

arrivaltime = [1, 2, 3, 4, 5]

bursttime = [3, 5, 1, 7, 4]

priority = [3, 4, 1, 7, 8]

for i in range(totalprocess):

proc[i][0] = arrivaltime[i]

proc[i][1] = bursttime[i]

proc[i][2] = priority[i]

proc[i][3] = i + 1

# Using inbuilt sort function

proc = sorted (proc, key = lambda x:x[2])

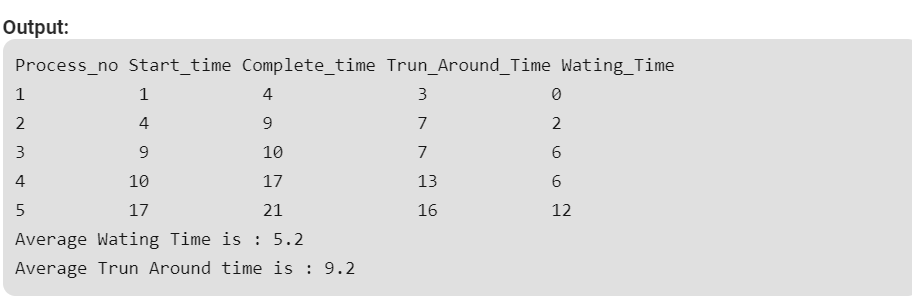
proc = sorted (proc)

# Calling function findgc for

# finding Gantt Chart

findgc()

**Output: Screenshots**



**Experiment No: 10**

Student Name and Roll Number: Hitesh 18csu086

Semester /Section: 5/A

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

To familiarize the students about CPU scheduling Algorithms.

**Program Outcome**

* The students will understand the Round robin Scheduling with Arrival and without Arrival Time

**Problem Statement**

Implement the following CPU scheduling Algorithms.

**Round robin Scheduling**

**Background Study:**

**Round robin**

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

**Algorithm/ flowchart**

Create an array rem bt[] to keep track of remaining

Burst time of processes this array is initially a

Copy of bt[]

Create another array wt[] to store waiting times

Of processes . Initialise this array to 0.

Initialise time : t =0

Keep traversing the all processes while all processes

Are not done . Do following for i’th process if it is

Not done yet .

If rem bt[i] > quantum

T = t +quantum

Bt rem[i] = quantum

Else // Last cycle for this process

T = t + bt rem[i]

Wt[i] = t- bt[i]

Bt rem[i] = 0;

**Code for Round robin Without Arrival Time**

# Python3 program for implementation of

# RR scheduling

# Function to find the waiting time

# for all processes

def findWaitingTime(processes, n, bt,

wt, quantum):

rem\_bt = [0] \* n

# Copy the burst time into rt[]

for i in range(n):

rem\_bt[i] = bt[i]

t = 0 # Current time

# Keep traversing processes in round

# robin manner until all of them are

# not done.

while(1):

done = True

# Traverse all processes one by

# one repeatedly

for i in range(n):

# If burst time of a process is greater

# than 0 then only need to process further

if (rem\_bt[i] > 0) :

done = False # There is a pending process

if (rem\_bt[i] > quantum) :

# Increase the value of t i.e. shows

# how much time a process has been processed

t += quantum

# Decrease the burst\_time of current

# process by quantum

rem\_bt[i] -= quantum

# If burst time is smaller than or equal

# to quantum. Last cycle for this process

else:

# Increase the value of t i.e. shows

# how much time a process has been processed

t = t + rem\_bt[i]

# Waiting time is current time minus

# time used by this process

wt[i] = t - bt[i]

# As the process gets fully executed

# make its remaining burst time = 0

rem\_bt[i] = 0

# If all processes are done

if (done == True):

break

# Function to calculate turn around time

def findTurnAroundTime(processes, n, bt, wt, tat):

# Calculating turnaround time

for i in range(n):

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

# Function to calculate average waiting

# and turn-around times.

def findavgTime(processes, n, bt, quantum):

wt = [0] \* n

tat = [0] \* n

# Function to find waiting time

# of all processes

findWaitingTime(processes, n, bt,

wt, quantum)

# Function to find turn around time

# for all processes

findTurnAroundTime(processes, n, bt,

wt, tat)

# Display processes along with all details

print("Processes Burst Time Waiting",

"Time Turn-Around Time")

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" ", i + 1, "\t\t", bt[i],

"\t\t", wt[i], "\t\t", tat[i])

print("\nAverage waiting time = %.5f "%(total\_wt /n) )

print("Average turn around time = %.5f "% (total\_tat / n))

# Driver code

if \_\_name\_\_ =="\_\_main\_\_":

# Process id's

proc = [1, 2, 3]

n = 3

# Burst time of all processes

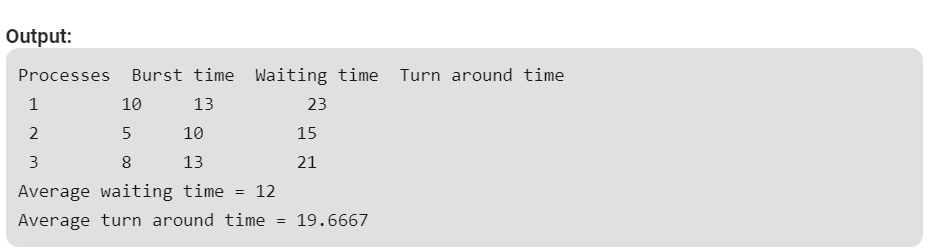
burst\_time = [10, 5, 8]

# Time quantum

quantum = 2;

findavgTime(proc, n, burst\_time, quantum)

**Output: Screenshots**



**Code for Round robin With Arrival Time**

// Scheduling with different arrival time

import java.util.\*;

//import java.util.concurrent.CompletableFuture;

public class RoundRobin {

public static void findWaitingTime(int process[],int wt\_time[],int n ,int brusttime[],int quantum,int completion\_time[],int arrival\_time[]){

// copy the value of brusttime array into wt\_time array.

int rem\_time[] = new int[n];

for(int i=0;i<wt\_time.length;i++){

rem\_time[i]= brusttime[i];

}

int t=0;

int arrival=0;

// processing until the value of element of rem\_time array is 0

while(true){

// ;

boolean done = true;

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

if(rem\_time[i]>0){

done =false;

if(rem\_time[i]>quantum && arrival\_time[i]<=arrival){

t +=quantum;

rem\_time[i]-=quantum;

arrival++;

}

else{

if(arrival\_time[i]<=arrival){

arrival++;

t+=rem\_time[i];

rem\_time[i]=0;

completion\_time[i]=t; }

}

}

}

if(done==true)

{

break;

}

}

}

public static void findTurnAroundTime(int process[] ,int wt\_time[],int n,int brusttime[],int tat\_time[],int completion\_time[],int arrival\_time[]){

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

tat\_time[i]= completion\_time[i]-arrival\_time[i];

wt\_time[i] = tat\_time[i]-brusttime[i];

}

}

public static void findAvgTime(int process[],int n,int brusttime[],int quantum,int arrival\_time[]){

int wt\_time[] = new int[n];

int tat\_time[] = new int[n];

int completion\_time[] = new int[n];

findWaitingTime(process,wt\_time,n,brusttime,quantum,completion\_time,arrival\_time);

findTurnAroundTime(process,wt\_time,n,brusttime,tat\_time,completion\_time,arrival\_time);

int total\_wt = 0, total\_tat = 0;

System.out.println("Processes " +" Arrival Time\t"+ " Burst time " +" completion time"+

" Turn Around Time " + " Waiting time");

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

{

total\_wt = total\_wt + wt\_time[i];

total\_tat = total\_tat + tat\_time[i];

System.out.println(" " + (i+1) + "\t\t"+ arrival\_time[i]+"\t\t"+ + brusttime[i] +"\t " +completion\_time[i]+"\t\t"

+tat\_time[i] +"\t\t " + wt\_time[i]);

}

System.out.println("Average waiting time = " +

(float)total\_wt / (float)n);

System.out.println("Average turn around time = " +

(float)total\_tat / (float)n);

}

public static void main(String []agrs){

RoundRobin object = new RoundRobin();

Scanner scan = new Scanner(System.in);

int quantum = 2;

int arrival\_time[] = new int[]{0,1,2,3};

int process[] = new int[]{1,2,3,4};

int brusttime[] = new int[]{5,4,2,1};

int n = process.length;

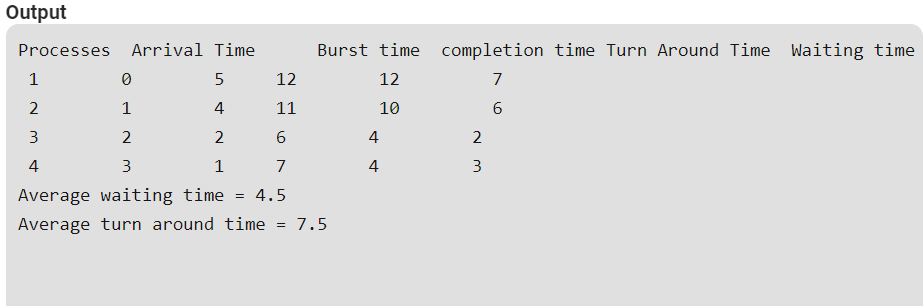
findAvgTime(process,n,brusttime,quantum,arrival\_time);

scan.close();

}

}

**Output: Screenshots**



**Suggested Question Bank**

**Preparatory Questions**

1. Which module gives control of the CPU to the process selected by the short-term scheduler?
   1. **dispatcher**
   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 can not be scheduled
   4. none of the mentioned
6. Process are classified into different groups in
   1. shortest job scheduling algorithm
   2. round robin scheduling algorithm
   3. priority scheduling algorithm
   4. **multilevel queue scheduling algorithm**
7. Which one of the following can not be scheduled by the kernel?
   1. kernel level thread
   2. **user level thread**
   3. process
   4. none of the mentioned
8. CPU scheduling is the basis of \_\_\_\_\_\_\_\_\_\_\_\_.
   1. multiprocessor systems
   2. **multiprogramming operating systems**
   3. larger memory sized systems
   4. None of these
9. With multiprogramming, \_\_\_\_\_\_ is used productively.

**a) time** b) space c) money d) All of these

1. The two steps of a process execution are : (choose two)

**a) I/O Burst b) CPU Burst** c) Memory Burst d) OS Burst

1. An I/O bound program will typically have :
   1. a few very short CPU bursts
   2. many very short I/O bursts
   3. **many very short CPU bursts**
   4. a few very short I/O bursts
2. A process is selected from the \_\_\_\_\_\_ queue by the \_\_\_\_\_\_\_\_ scheduler, to be executed.

a) blocked, short term b) wait, long term c) **ready, short term** d) ready, long term

1. In the following cases non – preemptive scheduling occurs : (Choose two)
   1. When a process switches from the running state to the ready state
   2. **When a process goes from the running state to the waiting state**
   3. When a process switches from the waiting state to the ready state
   4. When a process terminates
2. Dispatch latency is :
   1. the speed of dispatching a process from running to the ready state
   2. the time of dispatching a process from running to ready state and keeping the CPU idle
   3. **the time to stop one process and start running another one**
   4. None of these
3. Scheduling is done so as to :
   1. **increase the throughput**
   2. decrease the throughput
   3. increase the duration of a specific amount of work
   4. None of these
4. Turnaround time is :
   1. the total waiting time for a process to finish execution
   2. the total time spent in the ready queue
   3. the total time spent in the running queue
   4. **the total time from the completion till the submission of a process**
5. Scheduling is done so as to :
   1. increase the turnaround time
   2. **decrease the turnaround time**
   3. keep the turnaround time same
   4. there is no relation between scheduling and turnaround time

**Experiment No: (Mini Project)**

Student Name and Roll Number:

Semester /Section:

Link to Code:

Date:

Faculty Signature:

Remarks:

**Project Title:**

**Description of Project:**

**Problem Statement:**

**Problem Analysis:**

**Program Design:**

**Programming Requirements:**

**Data/Input Output Description:**

**Algorithmic Approach/Algorithm/DFD/ER diagram/Program Steps**

**Implementation and Testing (stage/module wise)**

**Output (Screenshots)**