**IT23C02**

**OPERATING SYSTEMS**

**(OS) laboratory**

A PRACTICAL RECORD

*Submitted by*

Sri Ganesh Thiyagarajan K

2023506010

B. Tech (4/8)

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**MADRAS INSTITUTE OF TECHNOLOGY**

**ANNA UNIVERSITY CHENNAI - 600 044.**

January / May 2025

**Bonafide Certificate**

Name : Sri Ganesh Thiyagarajan K

Reg. No : 2023506010

Subject : IT23C02 OPERATING SYSTEMS LABORATORY

Department : Information Technology

Certified to the bonafide record of practical work done by Mr.Sri Ganesh Thiyagarajan K in the

**IT23C02 OPERATING SYSTEMS LABORATORY** during the period January 2025 to May 2025

Date:

K.GOVINDASAMY

STAFF-IN-CHARGE

Submitted for the Practical Examination held on:

Internal Examiner External Examiner

|  |  |
| --- | --- |
| Exp.no | 1 |
| Date | 28.01.25 |

BASIC LINUX COMMANDS

**Aim**

To learn about the basic linux commands and their functionality in linux based environment.

**Basic Linux Commands:**

**1.ls command**

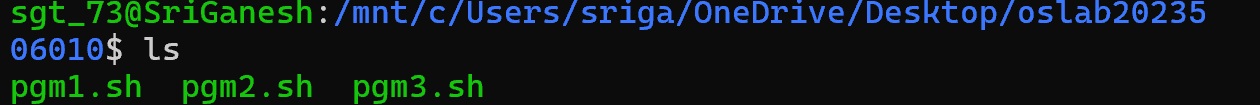
Function:

Displays the information about files in the current directory.

Syntax:

ls

output:



**i,ls-t**

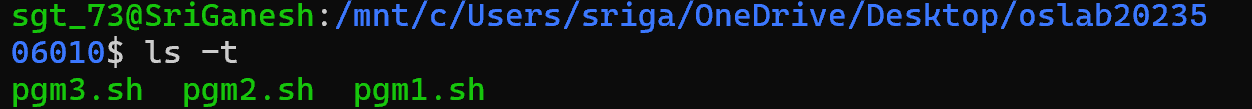
Function:

It sorts the file by modification time ,showing the last edited file first.

Syntax:

ls -t

output:



**ii,ls-1**

Function:

Displays one file per lline.

Syntax:

ls -1

output:

A black screen with blue text

AI-generated content may be incorrect.

**iii,ls-l**

Function:

Displays all information about fieles /directories.

Syntax:

ls -l

output:

A screenshot of a computer

AI-generated content may be incorrect.

**iv,ls-lh**

Function:

Displays the information about files in the current directory.

Syntax:

ls -lh

output:

A screenshot of a computer

AI-generated content may be incorrect.

**v,ls-a**

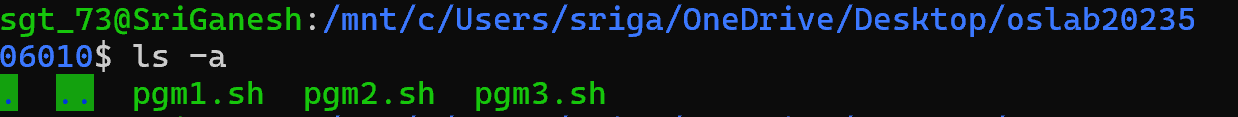
Function:

Displays Hidden files.

Syntax:

ls-a

output:



**vi,ls-i**

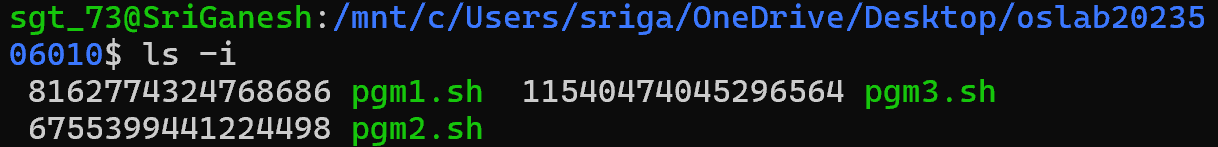
Function:

Displays file inode number using ls-i.

Syntax:

ls -i

output:



**2.pwd**

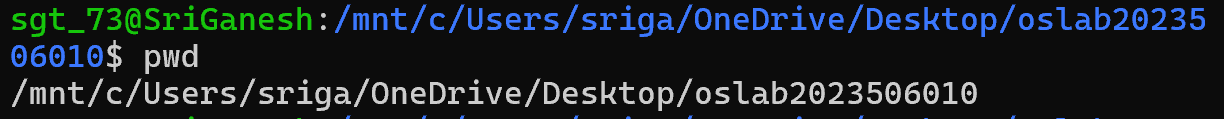
Function:

To print currently working directory.

Syntax:

pwd

output:



**3.mkdir**

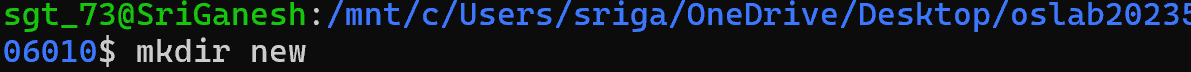
Function:

Create fresh directories in terminal.

Syntax:

Mkdir new

output:



**4.cd**

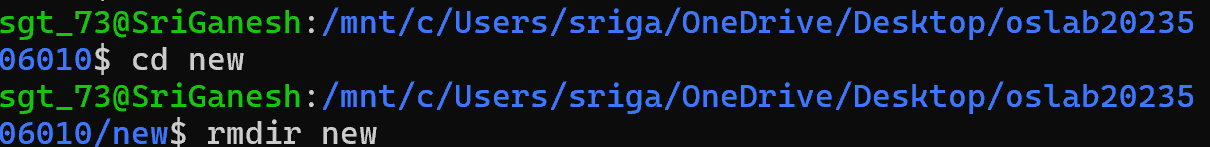
Function:

Navigate between directories.

Syntax:

Cd new

output:



**5.rmdir**

Function:

Delete permanently an empty directories

Syntax:

rmdir

output:

A screen shot of a computer program

AI-generated content may be incorrect.

**6.cp**

Function:

Equivalent to copy-paste and cut-paste in window.

Syntax:

cp a.txt b.txt

output:

A screen shot of a computer

AI-generated content may be incorrect.

**7.mv**

Function:

Renaming the files

Syntax:

mv b.txt second.txt

output:

A screen shot of a computer

AI-generated content may be incorrect.

**8.rm**

Function:

Delete the files created in directory.

Syntax:

rm second.txt

output:

A screen shot of a computer

AI-generated content may be incorrect.

**9.uname**

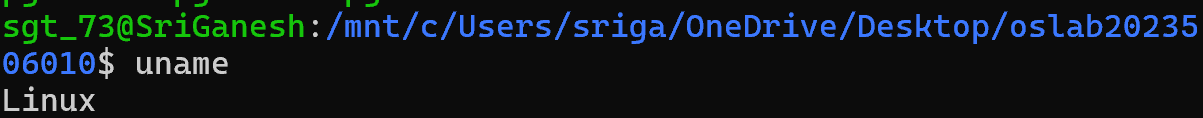
Function:

Check the complete OS information of system.

Syntax:

uname

output:



**10.locate**

Function:

Generally used to locate the files in database

Syntax:

locate “ ’ ”

output:



****

**11.Touch**

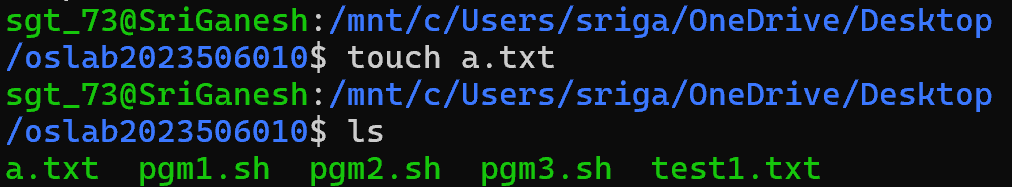
Function:

Touch command creates an empty files when put in the terminal in this format.

Syntax:

touch <filename>

output:



**12.ln**

Function:

Ln command is used to create a shortcut link to another file .

Syntax:

ln

output:

A screen shot of a computer screen

AI-generated content may be incorrect.

**13.cat**

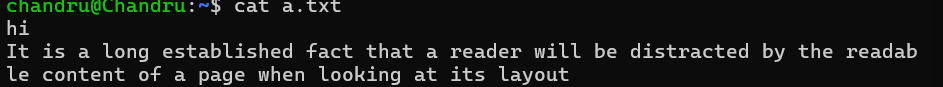
Function:

cat command is used to see the contents of a particular file.

Syntax:

cat a.txt

output:



**14.clear**

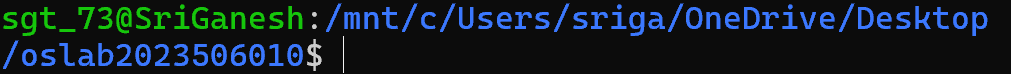
Function:

Clear command is standard command to clear the technical screen.

Syntax:

clear

output:



**15.ps**

Function:

ps command in linux is used to check the active process in terminal

Syntax:

ps

output:

A screen shot of a computer

AI-generated content may be incorrect.

**16.man**

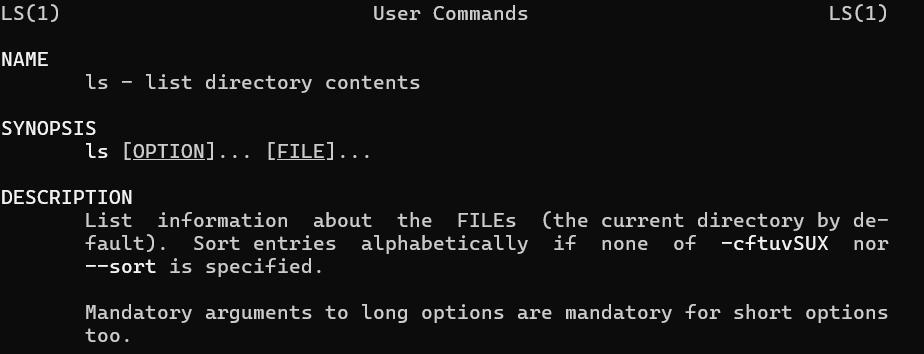
Function:

Man command displays a user manual for any commands or utilize available in terminal.

Syntax:

man -f ls

output:



**17.grep**

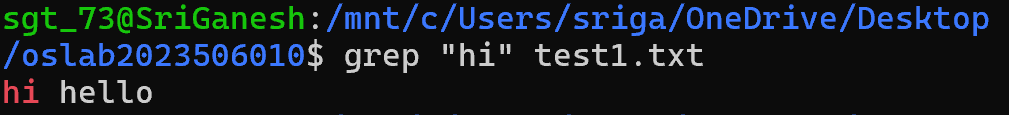
Function:

Grep command is used to find a specific string in a series of outputs.

Syntax:

Grep “<string to find”>

output:



**i,gre-i**

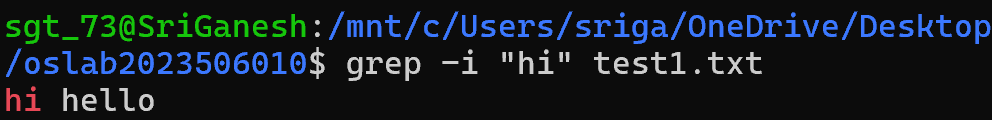
Function:

Grep-I option enables to search for a string case in given file.

Syntax:

grep -i ”hi” first.txt

output:



**ii,grep-c**

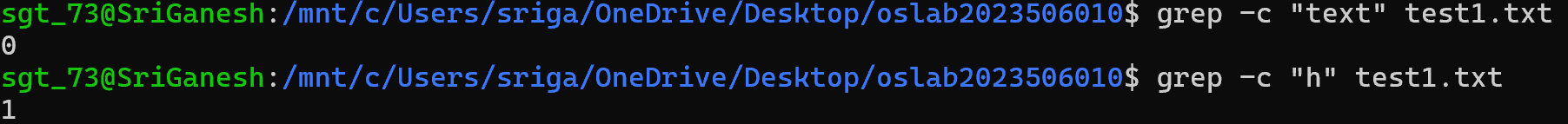
Function:

Find the no of lines that matches the given string /pattern.

Syntax:

Grep-c “text” first.txt

output:



**Iii,grep-l**

Function:

Display the files that contains the given string/pattern.

Syntax:

grep-l “hi” \*

output:

**A close-up of a computer screen

AI-generated content may be incorrect.**

**iv,grep-o**

Function:

grep displays the entire line which has the matched string.

Syntax:

grep-o

output:

A black screen with white text

AI-generated content may be incorrect.

**18.echo**

Function:

echo is specifically used to print something terminal

Syntax:

echo “Hello World”

output:

A black background with blue text

AI-generated content may be incorrect.

**19.wget**

Function:

Allows you to download files from the internet .

Syntax:

Wget www.google.com

output:

A computer screen with white text

AI-generated content may be incorrect.

**20.whoami**

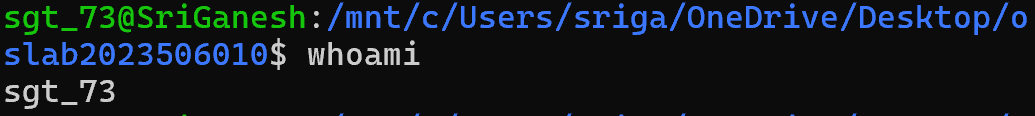
Function:

Provides basic information that is extremely useful when working on mobile systems.

Syntax:

whoami

output:



**21.sort**

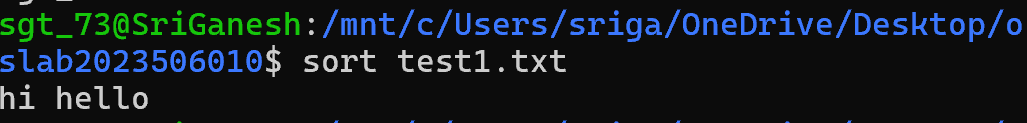
Function:

Sort command is used generally to sort the output of the file.

Syntax:

Sort a.txt

output:



22.fcal

Function:

To view the calendar for a particular month in terminal.

Syntax:

fcal 01 2025

output:

A screenshot of a computer

AI-generated content may be incorrect.

**23.whereis**

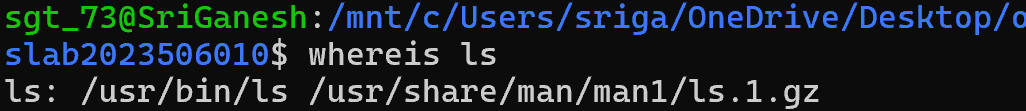
Function:

Used to see exact location of any command typed after this.

Syntax:

Whereis ls

output:



**24.df**

Function:

Get the details of the file systems.

Syntax:

df-h

output:

A screenshot of a computer

AI-generated content may be incorrect.

**25.wc**

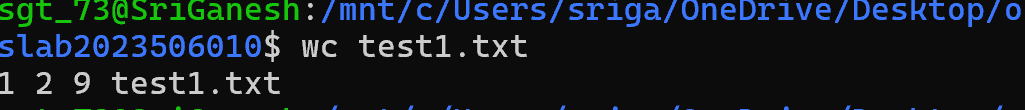
Function:

Indicates the no of words character lines etc.

Syntax:

wc a.txt

output:



**26.chmod**

Function:

To change the access mode of a file.

Syntax:

Chmod[options] [mode] [filename]

output:

A screen shot of a computer program

AI-generated content may be incorrect.

**Result:**

Thus the 25 Linux commands was executed successfully, and their outputs were observed. The commands helped in file handling, directory management, user information retrieval, and system monitoring.

|  |  |
| --- | --- |
| Exp.no | 2 |
| Date | 04.02.25 |

SHELL SCRIPTING

**Aim :**

To learn and run basic programs using shell script.

**1.Write a shell script program on arithmetic operations on two numbers.**

**Code:**

read -p "Enter the first number:" num1

read -p "Enter the second number:" num2

sum=$((num1 + num2))

diff=$((num1 - num2))

prod=$((num1 \* num2))

if [ $num3 -eq 0 ];

then

quot="undefined"

else

quot=$((num1 / num2))

fi

rem=$((num1 % num2))

echo "Sum:$sum"

echo "Difference:$diff"

echo "Product:$prod"

echo "Quotient:$quot"

echo "Remainder:$rem"

**output:**

**A computer screen with white text

AI-generated content may be incorrect.**

**2.Write shell script program to check whether to check whether a number is even or odd.**

**Code:**

read -p "Enter a number:" num

if [ $((num % 2))==0 ];

then

echo "$num is Even"

else

echo "$num is Odd"

fi

**Output:**

A black screen with white text

AI-generated content may be incorrect.

**3.Write shell script program to find sum of first n numbers.**

**Code:**

read -p "Enter value of n:" n

sum=0

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

do

sum=$(expr $sum + $i)

done

echo "The sum of the first $n numbers:$sum"

**Output:**

A black background with white text

AI-generated content may be incorrect.

**4.Write a shell script program to find the area of the triangle,rectangle,square.**

**Code:**

triangle\_area(){

echo "Triangle Area:"

read -p "Enter the base of the triangle:" base

read -p "Enter the height of the triangle:" height

area=$(echo "0.5 \* $base \* $height" | bc)

echo "Area of the triangle is : $area"

}

rectangle\_area(){

echo "Rectangle Area:"

read -p "Enter length:" length

read -p "Enter width:" width

area=$((length \* width))

echo "The area of the rectangle is :$area"

}

square\_area(){

echo "Square Area:"

read -p "Enter the side of the square:" side

area=$((side \* side))

echo "The area of the square is:$area"

}

triangle\_area

rectangle\_area

square\_area

**Output:**

A screenshot of a computer program

AI-generated content may be incorrect.

**5.Write a shell script program to find whether the number is positive, negative or zero.**

**Code:**

pos\_neg\_zero\_check(){

read -p "Enter a number:" num

if [ $num -gt 0 ];

then

echo "$num is positive"

elif [ $num -eq 0 ];

then

echo "$num is zero"

else

echo "$num is negative"

fi

}

pos\_neg\_zero\_check

**Output:**

A black background with white text

AI-generated content may be incorrect.

**6.Write a shell script program to find the Fibonacci number**

**Code:**

fibo(){

read -p "Enter the number of terms in fibonacci series:" n

a=0

b=1

echo "Fibonacci Series upto $n terms:"

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

do

echo -n "$a "

sum=$((a + b))

a=$b

b=$sum

done

echo

}

fibo

**output:**

A screen shot of a computer

AI-generated content may be incorrect.

**7.Write a shell script program to find factorial using function.**

**Code:**

factorial() {

num=$1

fact=1

for ((i=1; i<=num; i++)); do

fact=$((fact \* i))

done

echo "Factorial of $num is $fact"

}

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

8. **Write a shell script program to find the gcd of the given two number.**

**Code:**

gcd() {

a=$1

b=$2

while [ $b -ne 0 ]; do

temp=$b

b=$((a % b))

a=$temp

done

echo "GCD is $a"

}

read -p "Enter number1:" a

read -p "Enter number2:" b

gcd $a $b

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

**9. Write a shell script program to do arithmetic operations using function.**

**Code:**

arithmetic\_operations() {

a=$1

b=$2

echo "Sum: $((a + b))"

echo "Difference: $((a - b))"

echo "Product: $((a \* b))"

echo "Division: $((a / b))"

}

read -p "Enter num1:" num1

read -p "Enter num2:" num2

arithmetic\_operations $num1 $num2

Output:

A screen shot of a computer

AI-generated content may be incorrect.

**10.Write a shell script to display an array.**

**Code:**

display\_array(){

arr=("$@")

echo "Array Elements:"

for element in "${arr[@]}"

do

echo "$element"

done

}

array=()

read -p "How many elements do you want to enter?:" n

echo "Enter $n elements:"

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

do

read -p "Enter elements $((i + 1)):" value

array+=("$value")

done

display\_array "${array[@]}"

Output:

A screen shot of a computer

AI-generated content may be incorrect.

**11.Write a shell script to find minimum and maximum elements in array.**

**Code:**

display\_array\_min\_max() {

arr=("$@")

min=${arr[0]}

max=${arr[0]}

for num in "${arr[@]}"; do

if [ $num -lt $min ]; then

min=$num

fi

if [ $num -gt $max ]; then

max=$num

fi

done

echo "Array: ${arr[@]}"

echo "Minimum: $min"

echo "Maximum: $max"

}

array=()

read -p "Enter n:" n

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

do

read -p "Enter element $((i + 1)):" value

array+=("$value")

done

display\_array\_min\_max "${array[@]}"

**output:**

A screenshot of a computer

AI-generated content may be incorrect.

Result:

The shell scripts were successfully written, executed, and tested in a Linux environment. Each program demonstrates the use of basic shell scripting concepts such as variables, loops, conditionals, and functions.

|  |  |
| --- | --- |
| Exp.no | 3 |
| Date | 18.02.25 |

SYSTEM CALLS

**Aim :**

To use fork,exec,wait,exit system calls in c program.

**1.write a c program to execute Fork system call.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

int main() {

pid\_t pid = fork();

if (pid < 0) {

perror("Fork failed");

exit(1);

}

if (pid == 0) {

printf("Child process: PID = %d\n", getpid());

} else {

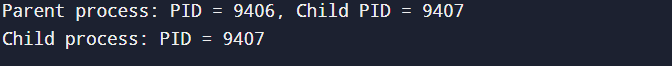
printf("Parent process: PID = %d, Child PID = %d\n", getpid(), pid);

}

return 0;

}

**Output:**



**2.Write a c program to execute execv system calls.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

int main() {

pid\_t pid = fork();

if (pid < 0) {

perror("Fork failed");

exit(1);

}

if (pid == 0) {

printf("Child process: Executing /bin/ls\n");

char \*argv[] = {"/bin/ls", "-l", NULL}; // List files with details

execv(argv[0], argv);

perror("Execv failed");

exit(1);

} else {

printf("Parent process: PID = %d, Child PID = %d\n", getpid(), pid);

}

return 0;

}

**Output:**

****

**3.Write a c program to execute wait system call.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main() {

pid\_t pid = fork();

if (pid < 0) {

// Fork failed

perror("Fork failed");

exit(1);

}

if (pid == 0) {

printf("Child process: PID = %d\n", getpid());

sleep(2);

printf("Child process: Exiting...\n");

exit(0); // Exit with status 0

} else {

int status;

wait(&status);

if (WIFEXITED(status)) {

printf("Parent process: Child terminated with exit status %d\n", WEXITSTATUS(status));

} else {

printf("Parent process: Child did not terminate normally\n");

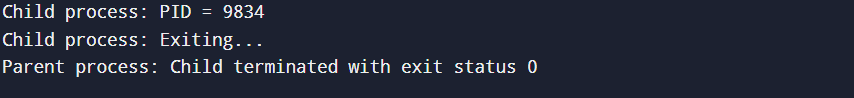
}

}

return 0;

}

**Output:**

****

**4.write a c program to execute exit system call.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main() {

pid\_t pid = fork();

if (pid < 0) {

perror("Fork failed");

exit(1);

}

if (pid == 0) {

printf("Child process (PID: %d) performing task...\n", getpid());

sleep(2);

printf("Child process (PID: %d) exiting...\n", getpid());

exit(0);

} else {

int status;

wait(&status);

if (WIFEXITED(status)) {

printf("Parent process (PID: %d) detected child exit with status %d\n", getpid(), WEXITSTATUS(status));

} else {

printf("Parent process (PID: %d) detected abnormal child termination\n", getpid());

}

printf("Parent process (PID: %d) exiting...\n", getpid());

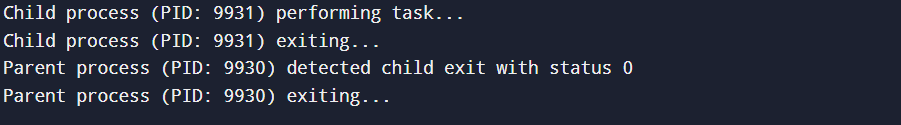
exit(0);

}

return 0;

}

**Output:**



**Result:**

The fork(), getpid() and the other system call programs were executed successfully. Each system call provides low-level access to hardware or the operating system’s kernel services. The behaviour of each system call observed and recorded.

|  |  |
| --- | --- |
| Exp.no | 4 |
| Date | 21.02.25 |

MULTI THREADING AND THREAD SYNCHRONIZATION

**Aim :**

To implement single thread and multiple thread using c program.

**1.Write a c program to demonstrate single threading.**

**Code:**

#include <stdio.h>

#include <unistd.h>

void task1() {

printf("Task 1 Started\n");

sleep(2);

printf("Task 1 Ended\n");

}

void task2() {

printf("Task 2 Started\n");

sleep(2);

printf("Task 2 Ended\n");

}

int main() {

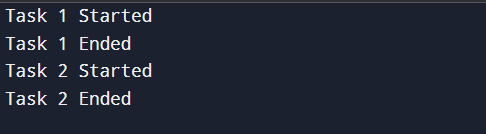
task1();

task2();

return 0;

}

**Output:**



**2.Write a c program to demonstrate multi threading.**

**Code:**

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

void\* task1(void\* arg) {

printf("Task 1 Started\n");

sleep(2);

printf("Task 1 Ended\n");

return NULL;

}

void\* task2(void\* arg) {

printf("Task 2 Started\n");

sleep(2);

printf("Task 2 Ended\n");

return NULL;

}

int main() {

pthread\_t t1, t2;

pthread\_create(&t1, NULL, task1, NULL);

pthread\_create(&t2, NULL, task2, NULL);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

return 0;

}

**Output:**



**Result:**

The Single and multithreaded programs were successfully executed. Thread creation, concurrent were execution, and synchronization mechanisms such as mutex locks and semaphores were implemented and tested.

|  |  |
| --- | --- |
| Exp.no | 5 |
| Date | 28.02.25 |

PREEMPTIVE AND NON PREEMPTIVE PROCESS

SCHEDULING ALGORITHMS

**Aim :**

To implement preemptive and on preemptive process scheduling algorithms.

1.First come first serve scheduling algorithm.

#include <stdio.h>

typedef struct {

int id;

int arrival\_time;

int burst\_time;

int completion\_time;

int turnaround\_time;

int waiting\_time;

} Process;

void print\_process\_table(Process processes[], int n) {

printf("P#\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time\n");

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",

processes[i].id,

processes[i].arrival\_time,

processes[i].burst\_time,

processes[i].completion\_time,

processes[i].turnaround\_time,

processes[i].waiting\_time);

}

}

void fcfs(Process processes[], int n) {

int current\_time = 0;

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

if (current\_time < processes[i].arrival\_time) {

current\_time = processes[i].arrival\_time;

}

processes[i].completion\_time = current\_time + processes[i].burst\_time;

processes[i].turnaround\_time = processes[i].completion\_time - processes[i].arrival\_time;

processes[i].waiting\_time = processes[i].turnaround\_time - processes[i].burst\_time;

current\_time = processes[i].completion\_time;

}

print\_process\_table(processes, n);

}

int main() {

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

Process processes[n];

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

processes[i].id = i + 1;

printf("Enter arrival time and burst time for process %d: ", i + 1);

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

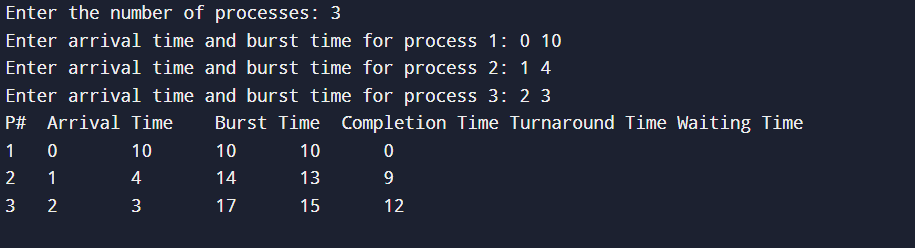
}

fcfs(processes, n);

return 0;

}

**Output:**



2.shortest job first algorithm.

#include<stdio.h>

typedef struct {

int id;

int arrival\_time;

int burst\_time;

int remaining\_time;

int completion\_time;

int turnaround\_time;

int waiting\_time;

} Process;

void print(Process processes[], int n) {

printf("P#\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time\n");

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",

processes[i].id,

processes[i].arrival\_time,

processes[i].burst\_time,

processes[i].completion\_time,

processes[i].turnaround\_time,

processes[i].waiting\_time);

}

}

void sjf(Process processes[], int n) {

int current\_time = 0, completed = 0;

while (completed < n) {

int min\_remaining\_time = 9999;

int short\_ind = -1;

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

if (processes[i].arrival\_time <= current\_time && processes[i].remaining\_time > 0 && processes[i].remaining\_time < min\_remaining\_time) {

min\_remaining\_time = processes[i].remaining\_time;

short\_ind = i;

}

}

if (short\_ind != -1) {

processes[short\_ind].remaining\_time--;

current\_time++;

if (processes[short\_ind].remaining\_time == 0) {

processes[short\_ind].completion\_time = current\_time;

processes[short\_ind].turnaround\_time = processes[short\_ind].completion\_time - processes[short\_ind].arrival\_time;

processes[short\_ind].waiting\_time = processes[short\_ind].turnaround\_time - processes[short\_ind].burst\_time;

completed++;

}

} else {

current\_time++;

}

}

print(processes, n);

}

int main() {

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

Process processes[n];

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

processes[i].id = i + 1;

printf("Enter arrival time and burst time for process %d: ", i + 1);

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

processes[i].remaining\_time = processes[i].burst\_time;

processes[i].completion\_time = -1;

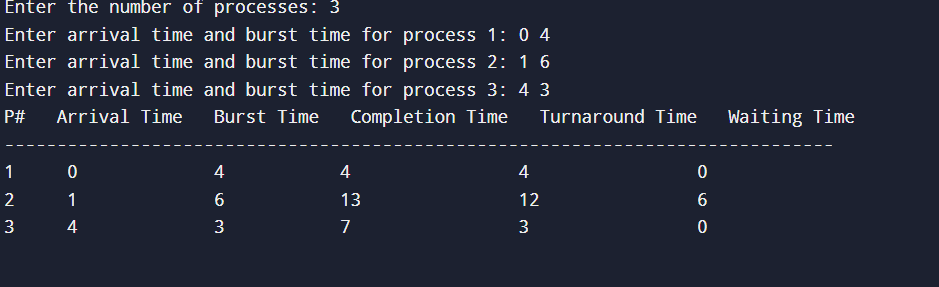
}

sjf(processes, n);

return 0;

}

**Output:**

****

3.Non preemptive shortest job first

#include<stdio.h>

typedef struct{

int id;

int arrival\_time;

int burst\_time;

int completion\_time;

int turnaround\_time;

int waiting\_time;

}Process;

void print(Process processes[],int n){

printf("P#\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time\n");

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",

processes[i].id,

processes[i].arrival\_time,

processes[i].burst\_time,

processes[i].completion\_time,

processes[i].turnaround\_time,

processes[i].waiting\_time);

}

}

void sjf(Process processes[],int n){

int current\_time=0;

int completed=0;

while(completed<n){

int min\_burst=9999;

int short\_ind=-1;

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

if(processes[i].arrival\_time<=current\_time && processes[i].completion\_time==-1 && processes[i].burst\_time<min\_burst){

min\_burst=processes[i].burst\_time;

short\_ind=i;

}

}

if(short\_ind!=-1){

Process \*p=&processes[short\_ind];

p->completion\_time=current\_time+p->burst\_time;

p->turnaround\_time=p->completion\_time-p->arrival\_time;

p->waiting\_time=p->turnaround\_time-p->burst\_time;

current\_time=p->completion\_time;

completed++;

}else{

current\_time++;

}

}

print(processes,n);

}

int main(){

int n;

printf("enter the number of processe:");

scanf("%d",&n);

Process processes[n];

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

processes[i].id=i+1;

printf("enter arrival time and burst time %d:",i+1);

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

processes[i].completion\_time=-1;

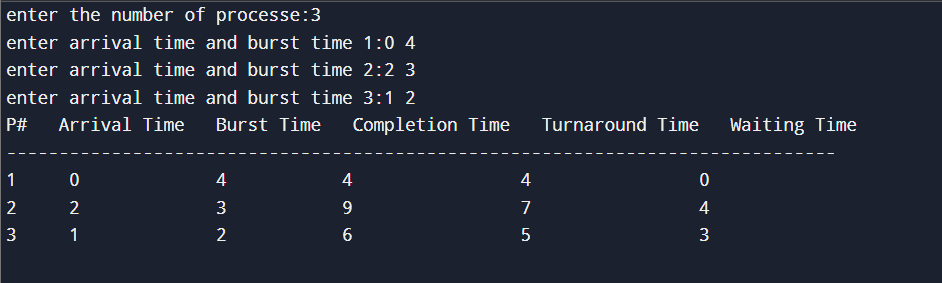
}

sjf(processes,n);

return 0;

}

**Output:**

****

4.Priority scheduling

#include <stdio.h>

typedef struct {

int process\_id;

int arrival\_time;

int burst\_time;

int completion\_time;

int waiting\_time;

int turnaround\_time;

int priority;

} Process;

void calculateWaitingTime(Process processes[], int n) {

int total\_time = 0;

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

processes[i].waiting\_time = total\_time - processes[i].arrival\_time;

if (processes[i].waiting\_time < 0) {

processes[i].waiting\_time = 0;

}

total\_time += processes[i].burst\_time;

}

}

void calculateTurnAroundTime(Process processes[], int n) {

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

processes[i].turnaround\_time = processes[i].burst\_time + processes[i].waiting\_time;

}

}

void sortProcessesByPriority(Process processes[], int n) {

Process temp;

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

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

if (processes[i].priority > processes[j].priority ||

(processes[i].priority == processes[j].priority && processes[i].arrival\_time > processes[j].arrival\_time)) {

// Swap the processes

temp = processes[i];

processes[i] = processes[j];

processes[j] = temp;

}

}

}

}

void displayResults(Process processes[], int n) {

printf("PN\tArrival Time\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",

processes[i].process\_id,

processes[i].arrival\_time,

processes[i].burst\_time,

processes[i].priority,

processes[i].waiting\_time,

processes[i].turnaround\_time);

}

}

void priorityScheduling(Process processes[], int n) {

sortProcessesByPriority(processes, n);

calculateWaitingTime(processes, n);

calculateTurnAroundTime(processes, n);

displayResults(processes, n);

}

int main() {

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

Process processes[n];

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

processes[i].process\_id = i + 1; // Process IDs are 1, 2, 3, ...

printf("Enter arrival time, burst time, and priority for process %d: ", i + 1);

scanf("%d %d %d", &processes[i].arrival\_time, &processes[i].burst\_time, &processes[i].priority);

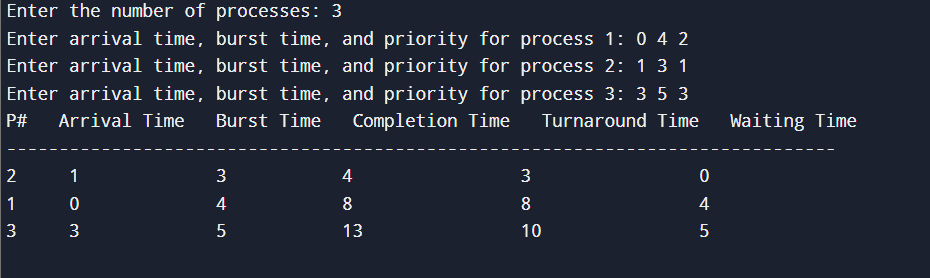
}

priorityScheduling(processes, n);

return 0;

}

**Outptut:**



5.Round Robin scheduling algorithm

#include<stdio.h>

#define max 20

int main() {

int i, burstTime[max], remainTime[max], remainProcess, arrivalTime[max], totalExecutionTime = 0, timeQuantum, flag = 0, n;

float totalWaitingTime = 0, totalTurnAroundTime = 0;

printf("Enter the Number of Process (max 20): ");

scanf("%d", &n); // n is the number of Process

remainProcess = n;

int completionTime[max], turnaroundTime[max], waitingTime[max];

printf("Enter Arrival Time\n");

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

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

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

}

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

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

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

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

remainTime[i] = burstTime[i]; // initially assume remaining time for any process is equal to its burst time!

}

printf("\nEnter Time Quantum: ");

scanf("%d", &timeQuantum);

for (i = 0; remainProcess != 0;) {

if (remainTime[i] <= timeQuantum && remainTime[i] > 0) {

totalExecutionTime += remainTime[i];

remainTime[i] = 0;

flag = 1;

}

else if (remainTime[i] > 0) {

remainTime[i] -= timeQuantum;

totalExecutionTime += timeQuantum;

}

if (flag == 1 && remainTime[i] == 0) {

completionTime[i] = totalExecutionTime;

turnaroundTime[i] = completionTime[i] - arrivalTime[i]; // Turnaround Time = Completion Time - Arrival Time

waitingTime[i] = turnaroundTime[i] - burstTime[i]; // Waiting Time = Turnaround Time - Burst Time

totalWaitingTime += waitingTime[i];

totalTurnAroundTime += turnaroundTime[i];

flag = 0;

remainProcess--;

}

if (i == n - 1)

i = 0;

else if (arrivalTime[i + 1] <= totalExecutionTime) {

i++;

}

else

i = 0;

}

totalWaitingTime = totalWaitingTime / n;

totalTurnAroundTime = totalTurnAroundTime / n;

printf("\nP#\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time\n");

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",

i + 1, arrivalTime[i], burstTime[i], completionTime[i], turnaroundTime[i], waitingTime[i]);

}

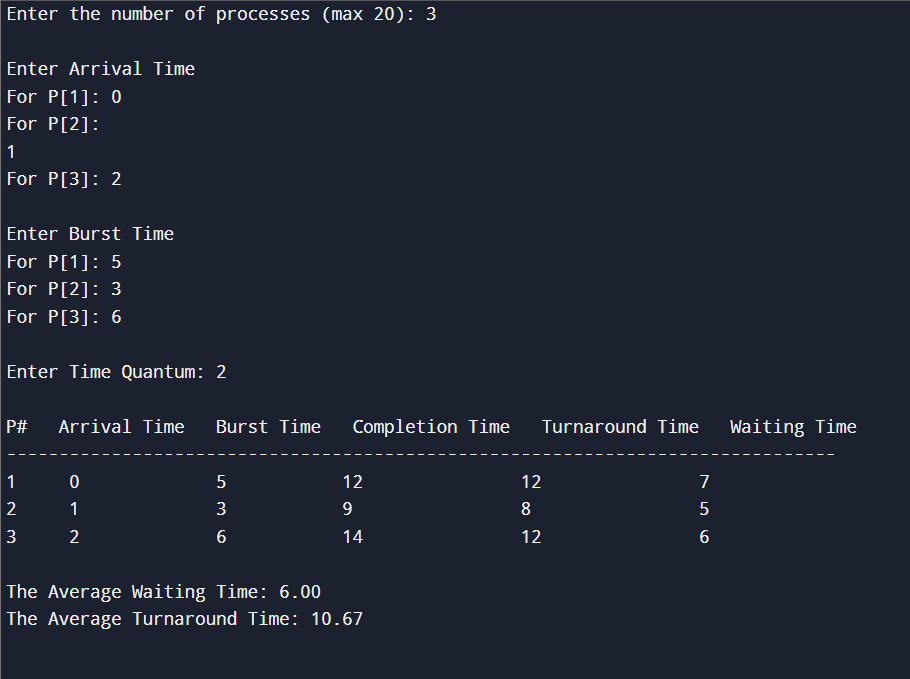
printf("\nThe Average Waiting Time: %.2f\n", totalWaitingTime);

printf("The Average Turnaround Time: %.2f\n", totalTurnAroundTime);

return 0;

}

**Output:**



Result:

The preemptive and non- preemptive scheduling algorithms were successfully implemented and their performance was observed by calculating average waiting time and wait around time. Gantt charts were used for visualizing the execution order.