**EXERCISE – 1**

**Name – Bhavy Kharbanda**

**SAP ID – 500082531**

**Roll Number – R214220342**

**Batch – B11**

**Date of Performance – 01/02/2021**

**Date of Submission – 27/03/2021**

OPERATING SYSTEM LAB

EXERCISE – 1.1

Shell Programming

Objective:**TO STUDY AND EXECUTE THE COMMANDS IN UNIX.**

* SIMPLE COMMANDS :
  + **Date Command.**
  + **Calendar Command.**
  + **Echo Command.**
  + **Banner Command.**
  + **“Who” Command**
  + **“Who am I” Command.**
  + **“tty” Command.**
  + **“Binary” Calculator Command.**
  + **“CLEAR” Command.**
  + **MAN Command.**
  + **Cd command**
  + **ls command**

1. **Date Command**

**This command can be used to display the date and time of the system. This can also be used to change the display pattern of the date and can show the date from past or future. Syntax:-**

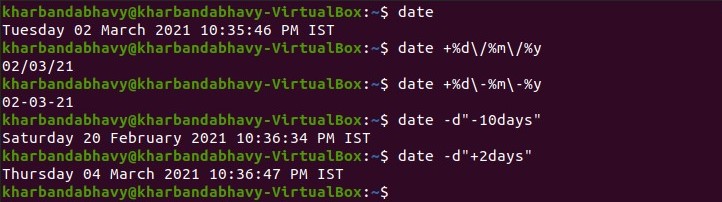
**~$ date**

**~$ date +%d\/%m\/%y**

**~$ date +%d\-%m\-%y**

**~$ date –d”(+/-)(number of days)days”**

**For example:- date –d”+7days” so it will show the date in future for the 7th day.**

****

1. **Calendar Command**

**This command can be used to display the calendar of the present month or the calendar of any month. Syntax:-**

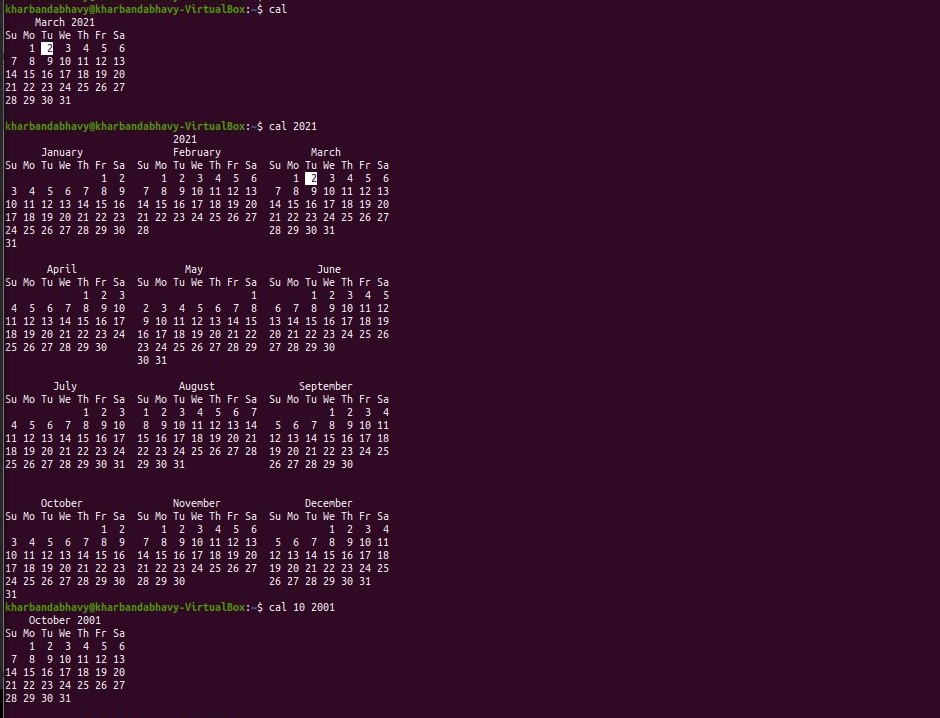
**~$ cal**

**~$ cal (year)**

**For example- cal 2021**

**~$ cal (month) (year)**

**For example- cal 10 2001**

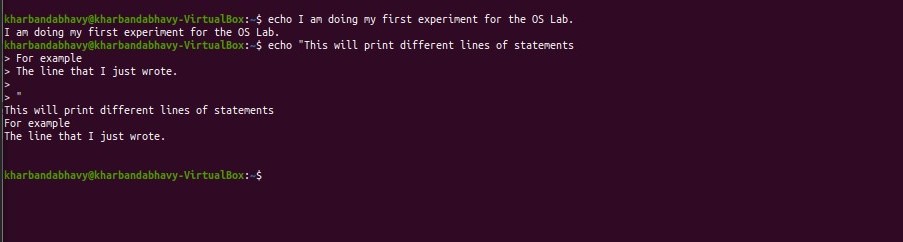
****

1. **Echo Command**

**This command is used to display sentence or the string passed as an argument. Syntax:-**

**~$ echo (string)**

**For Example:-echo I am doing my first experiment for the OS Lab.**

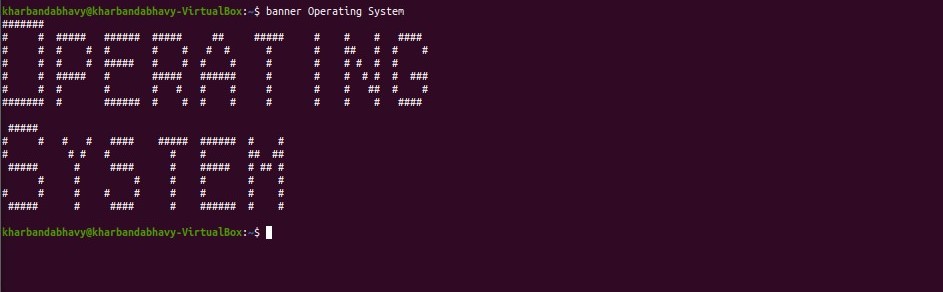
****

1. **Banner Command**

**This command can print the string in large letters like a banner. Syntax:-**

**~$ banner (string)**

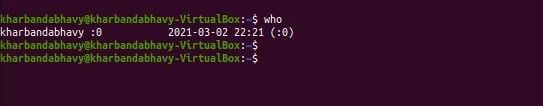
**For Example: - banner Operating System**

****

1. **WHO command**

**This command displays the information of currently logged in user. Syntax:-**

**~$ who**

****

1. **Clear Command**

**This command is used to clear the terminal.**

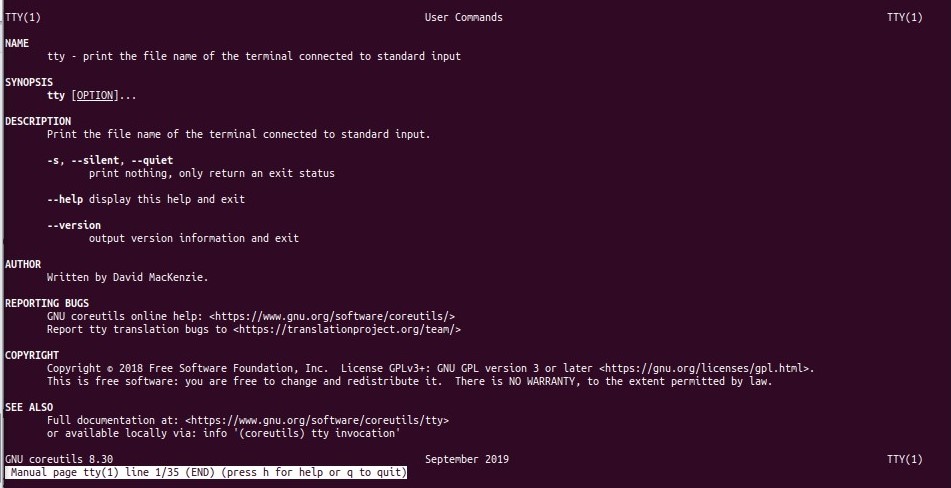
1. **MAN Command**

**This command helps us to display the user manual for any command that we can run on the terminal. Syntax:-**

**~$ man (command name)**

**For Example:-**

**~$ man tty**

****

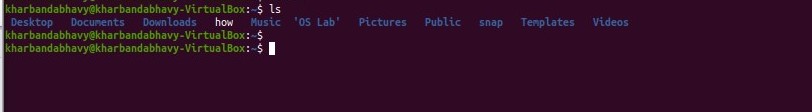
1. **Binary Calculator Command**

**This command is used to perform calculations on the terminal itself. For example: - All kinds of Arithmetic operations(+ - / \*), Conversion of base of numbers.**

****

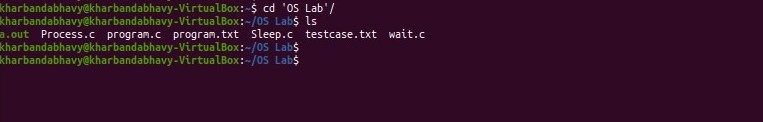
1. **Ls Command**

**This command displays the list of the files of the current directory.**

****

1. **Cd Command**

**This command is used to change the current working directory to other directory.**

****

**\*\*\*\*\*\*\*\*\*\*\* END OF EXERCISE – 1.1 \*\*\*\*\*\*\*\*\*\*\*\***

OPERATING SYSTEM LAB

EXERCISE – 1.2

System calls

Objective:**To write programs to perform following operations in UNIX:**

**a) Process Creation**

**b) Executing a command**

**c) Sleep command**

**d) Wait command**

1. **Process Creation Command**

**Process can be created using the fork() function, it creates a process at a separate address location in the Linux system.**

**Algorithm:-**

**Step1:- Start the program.**

**Step2:- Declare pid as Integer.**

**Step3:- Create the process using the fork() command.**

**Step4:- Check if pid is less than 0 then print error, else if pid is equal to 0 then print child process, else print parent process.**

**Step5:- Stop the program.**

**Code:-**

**#include<stdio.h>**

**#include <sys/types.h>**

**#include <unistd.h>**

**int main() {**

**intpid=getpid();**

**pid=fork();**

**if(pid<0)**

**printf("\n Error");**

**else if(pid==0)**

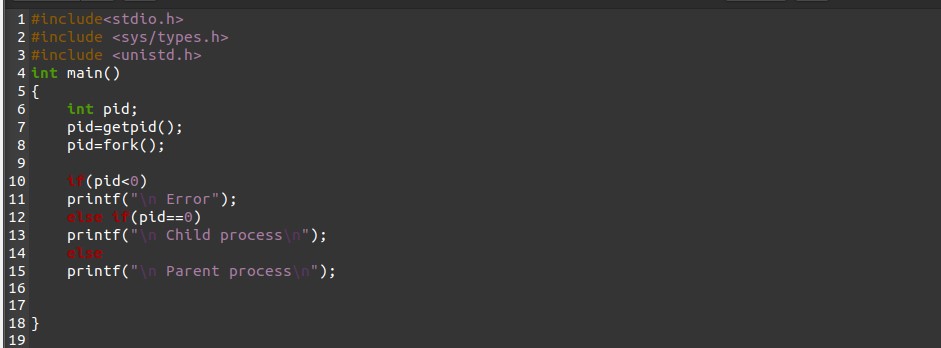
**printf("\n Child process\n");**

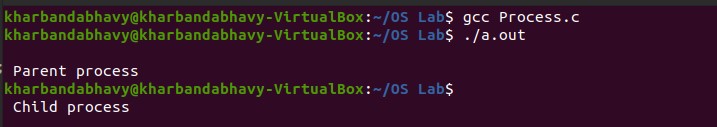
**else**

**printf("\n Parent process\n");**

**}**

**Screenshot/Output:-**

****

****

1. **Sleep Command**

**Sleep command helps the user to delay the execution. It takes the time in seconds by default. It basically pauses the execution for few seconds.**

**Algorithm:-**

**Step1:- Start the program.**

**Step2:- Declare pid as Integer.**

**Step3:- Create the process using the fork() command.**

**Step4:- Check if pid is less than 0 then print “Can’t Create, error”, else if pid is equal to 0 then print “Child Process Created”, else print “Parent Process Created”.**

**Step5:- Create the child process with sleep of 2.**

**Step6:- Stop the program.**

**Code:-**

**#include <stdio.h>**

**#include <unistd.h>**

**#include<sys/types.h>**

**int main() {**

**intpid;**

**pid = getpid();**

**pid = fork();**

**if (pid < 0)**

**printf("Can't Create, ERROR!\n");**

**else if (pid == 0) {**

**sleep(2);**

**printf("\n Child process! Created\n");**

**}**

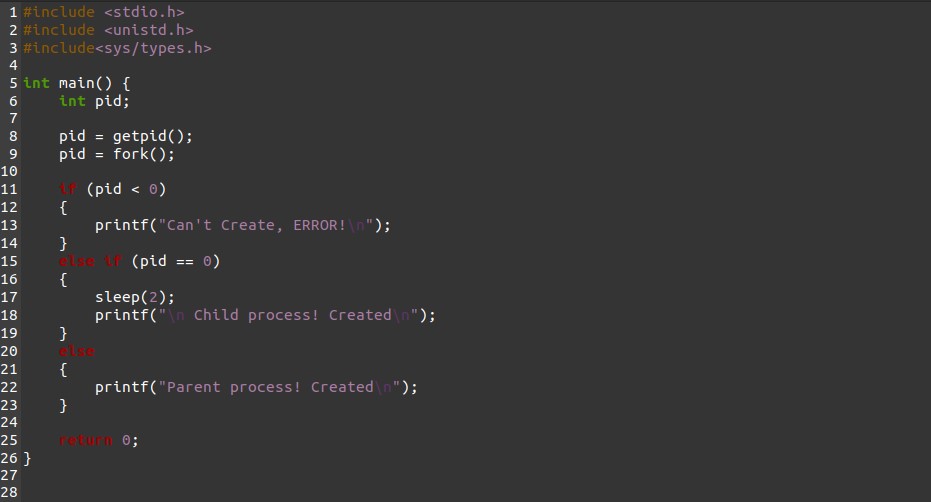
**else**

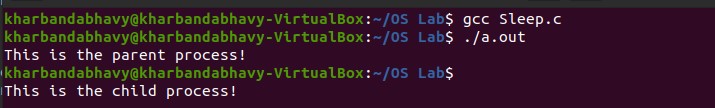
**printf("Parent process! Created\n");**

**return 0;**

**}**

**Screenshot/Output:-**

****

****

1. **Wait Command**

**Wait command waits for the completion of the previous process and then executes the upcoming process. It is helpful when multiple processes are running at the same time.**

**Algorithm:-**

**Step1:- Start the program.**

**Step2:- Create the process using the fork() command and assign it to a variable(pid).**

**Step3:- take another variable as I and let its value is 10.**

**Step4:- Check if pid is less than 0 print “error” else if pid is equal to 0, print the value of I and print “Terminate the child process. Else print “Parent process wait for child process to terminate”.**

**Step5:- Stop the program.**

**Code:-**

**#include<stdio.h>**

**#include<sys/types.h>**

**#include<unistd.h>**

**#include<sys/wait.h>**

**#include<stdlib.h>**

**int main()**

**{**

**int i=10;**

**intpid=getpid();**

**pid=fork();**

**if(pid<0)**

**printf("\n Error");**

**else if(pid==0)**

**{**

**printf("\n The initial value of i is %d", i);**

**i=i+10;**

**printf("\n The value of i is: %d", i);**

**printf("\n Child process started");**

**printf("\n Child process terminating....");**

**exit(0); }**

**else {**

**wait(0);**

**printf("\n Parent process wait for child process to terminate");**

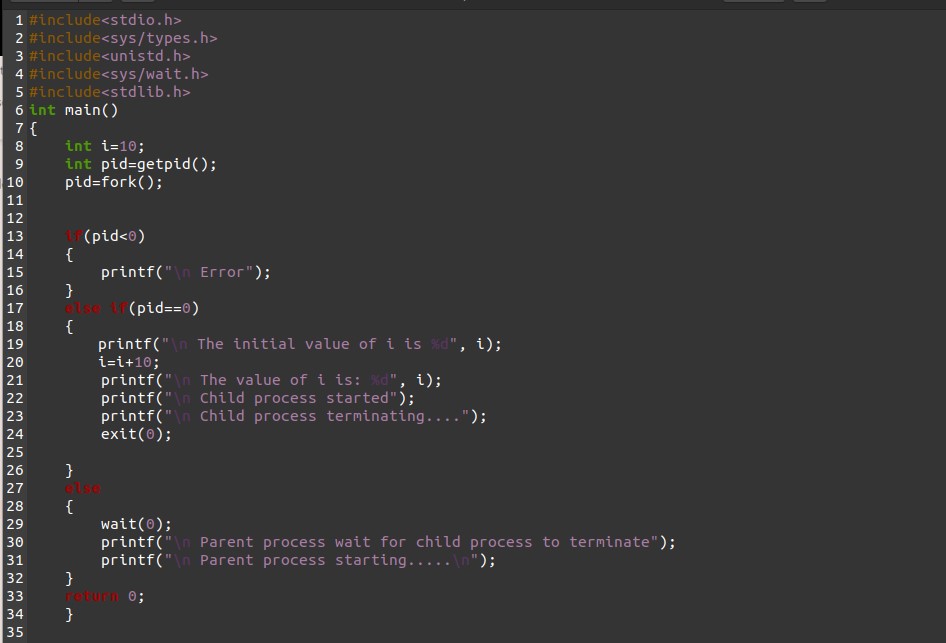
**printf("\n Parent process starting.....\n");**

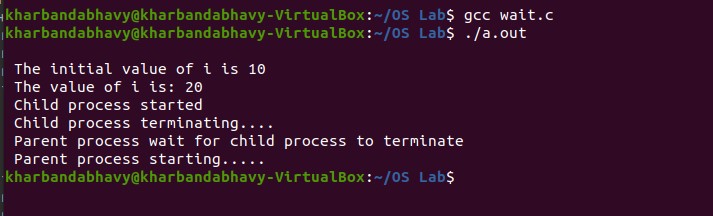
**}**

**return 0;**

**}**

**Screenshot/Output:-**

****

****

**\*\*\*\*\*\*\*\*\*\*\* END OF EXERCISE – 1.2 \*\*\*\*\*\*\*\*\*\*\*\***

OPERATING SYSTEM LAB

EXERCISE – 1.3

I/O System calls

Objective: **To write programs to perform following operations in UNIX:**

**a) Reading from a file**

**b) Writing into a file**

**c) File Creation**

1. **Reading from a file**

AIM: **To create the file, read data from the file, update the file.** ALGORITHM:

**Step 1. Get the data from the user.**

**Step 2. Open a file.**

**Step 3. Read from the file.**

**Step 4. Close the file.**

**Code:-**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main() {**

**char c[1000];**

**// file pointer is declared**

**FILE \*fptr;**

**if ((fptr = fopen("file.txt", "r")) == NULL)**

**{**

**printf("Error! opening file");**

**// Program exits if file pointer returns NULL.**

**exit(1);**

**}**

**// reads text until newline is encountered**

**fscanf(fptr, "%[^\n]", c);**

**printf("Data from the file:\n%s", c);**

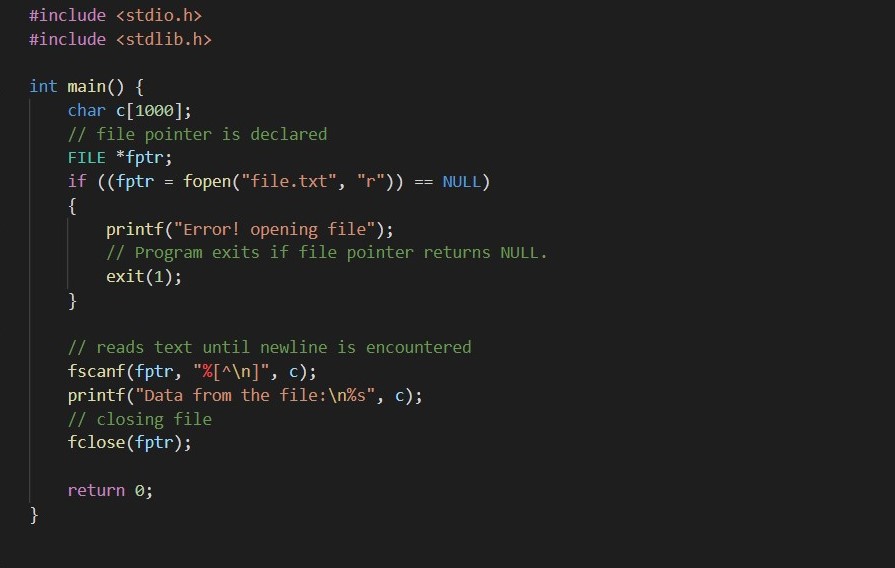
**// closing file**

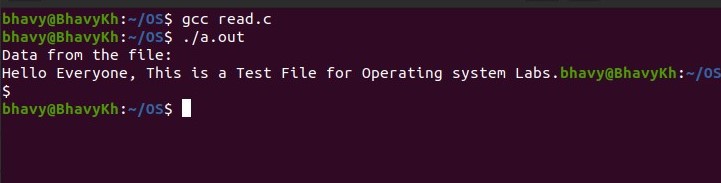
**fclose(fptr);**

**return 0;**

**}**

**Screenshot/Output:-**

****

****

1. **Writing into a file**

AIM: **To Write a C program to write the data in the file.**ALGORITHM:

**Step1. Get the data from the user.**

**Step2. Open a file.**

**Step3. Write the data from the file.**

**Step4. Get the data and update the file.**

**Code:-**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main() {**

**FILE \*fout;**

**char data[100];**

**fout = fopen("file.txt", "w");**

**if (fout == NULL) {**

**printf("File not created!");**

**}**

**else {**

**printf("Enter the data to write to the file:\n");**

**fgets(data, 100, stdin);**

**fputs(data, fout);**

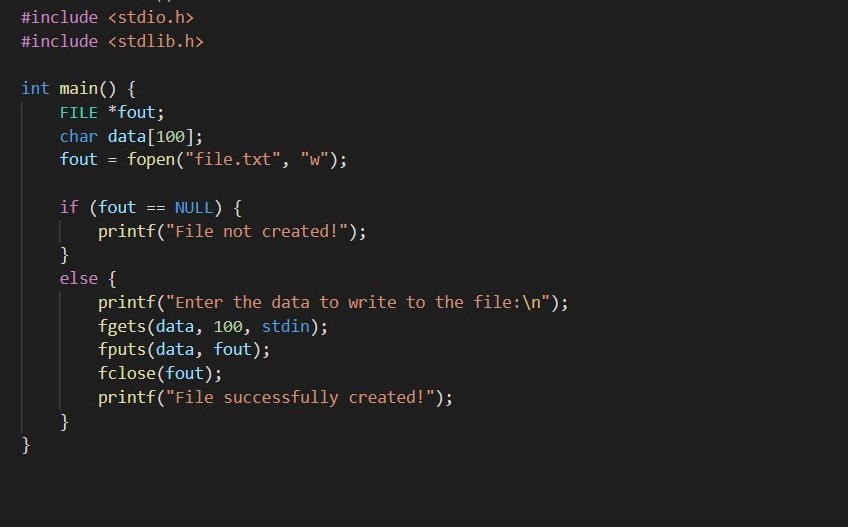
**fclose(fout);**

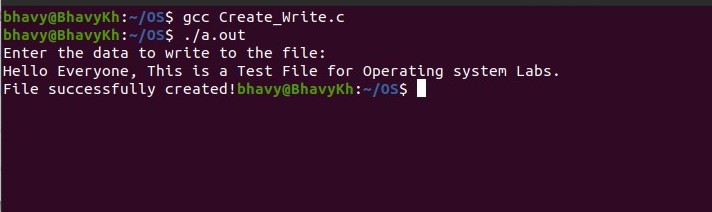
**printf("File successfully created!");**

**}**

**}**

**Screenshot/Output:-**

****

****

1. **File Creation**

AIM: **To write a C program to create a file.**

ALGORITHM:

**Step1: Start the program.**

**Step2: Create the file using create function and assign a variable to it.**

**Step3: If the value of the variable is less then print file cannot be created, otherwise print file is created.**

**Step4: Stop the program**

**Code:-**

**#include <stdio.h>**

**#include <stdlib.h> // for exit()**

**void main() {**

**FILE \*fout;**

**if (fout == NULL) {**

**printf("File not created!");**

**exit(0);**

**}**

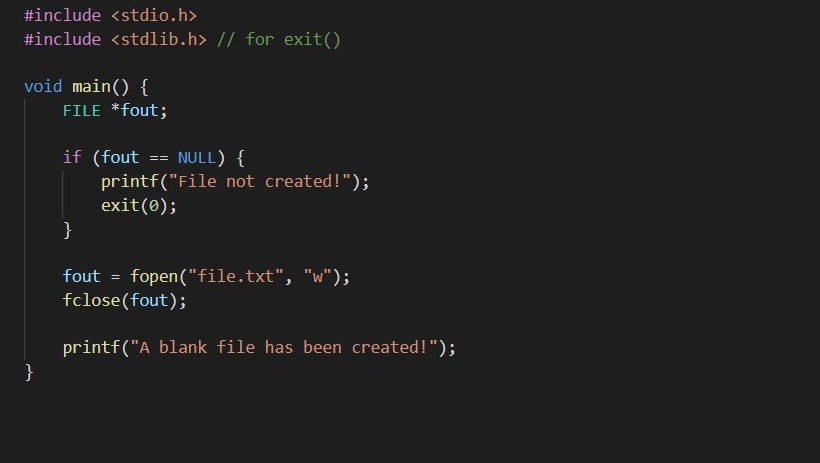
**fout = fopen("file.txt", "w");**

**fclose(fout);**

**printf("A blank file has been created!");**

**}**

**Screenshot/Output:-**

****

**\*\*\*\*\*\*\*\*\*\*\* END OF EXERCISE – 1.3 \*\*\*\*\*\*\*\*\*\*\*\***

**THANK YOU**