**SURYADATTA COLLEGE OF MANAGEMENT**

**INFORMATION RESEARCH & TECHNOLOGY**

**BAVDHAN ,PUNE-411021**

**LAB COURSE ON CS-501-MJ**

**( ADVANCE OPERATING SYSTEM)**

**Submitted by**

• **NAME OF STUDENT**

1. **Shriyash Raut**

**Under the Guidance of**

**Mr. DEEPAK SINGH**

**SUBMITTED IN PARTIAL**

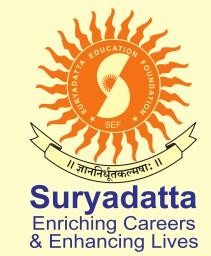
**FULLFILLMENT OF MASTER OF**

**SCIENCE(COMPUTER SCIENCE)**

**SEM-I**

**SAVITRIBAI PHULE PUNE UNIVERSITY**

**For Academic Year 2023-2024**



# CERTIFICATE

**This is to certify that Mr./Ms.**

**student of MSC(CS) Semester having Seat No. at**

**Suryadatta College of Management Information Research & Technology**

**(SCMIRT), Pune, has successfully completed the assigned practical in**

**prescribed by the Savitribai Phule Pune**

|  |  |
| --- | --- |
| **University during the academic year** | **.** |
| **Internal Examiner** | **External Examiner** |

**Principal Place: Pune Date:**

**1) Create a File with Hole in it**

**Ans:-**

#include <stdio.h>

#include <stdlib.h>

int main() {

// Define a file pointer FILE \*file;

// Open a file for writing (creating if it doesn't exist)

file = fopen("example.txt", "w"); if (file == NULL) {

perror("Error opening file for writing"); return 1;

}

// Write data to the file fprintf(file, "Hello, World!\n"); fprintf(file, "This is a test file.\n");

// Close the file fclose(file);

// Open the file for reading file = fopen("example.txt", "r"); if (file == NULL) {

perror("Error opening file for reading"); return 1;

}

// Read and print the contents of the file char buffer[256]; while (fgets(buffer, sizeof(buffer), file) != NULL) {

printf("%s", buffer);

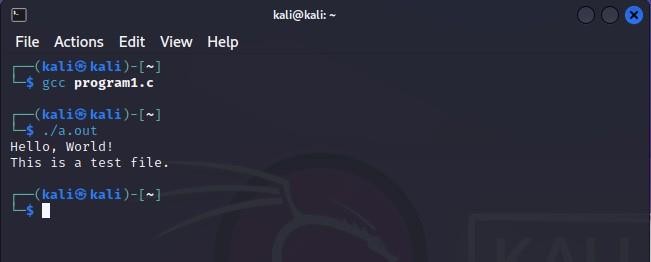
}

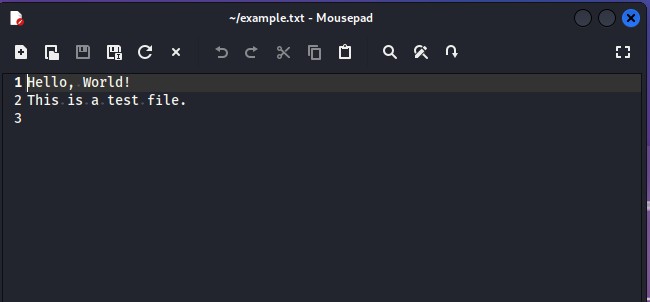
// Close the file fclose(file);

return 0;

}

**Output:-**





2) **Takes multiple files as Command Line Arguments And print their inode number.**

#include<stdio.h>

#include<sys/stat.h> #include<unistd.h> int main(int argc, char\*argv[]){ struct stat buf; int I;

// loop through coom for(i=0;i<argc;i++){

// get file

if(stat(argv[i],&buf)<0){ perror("stat

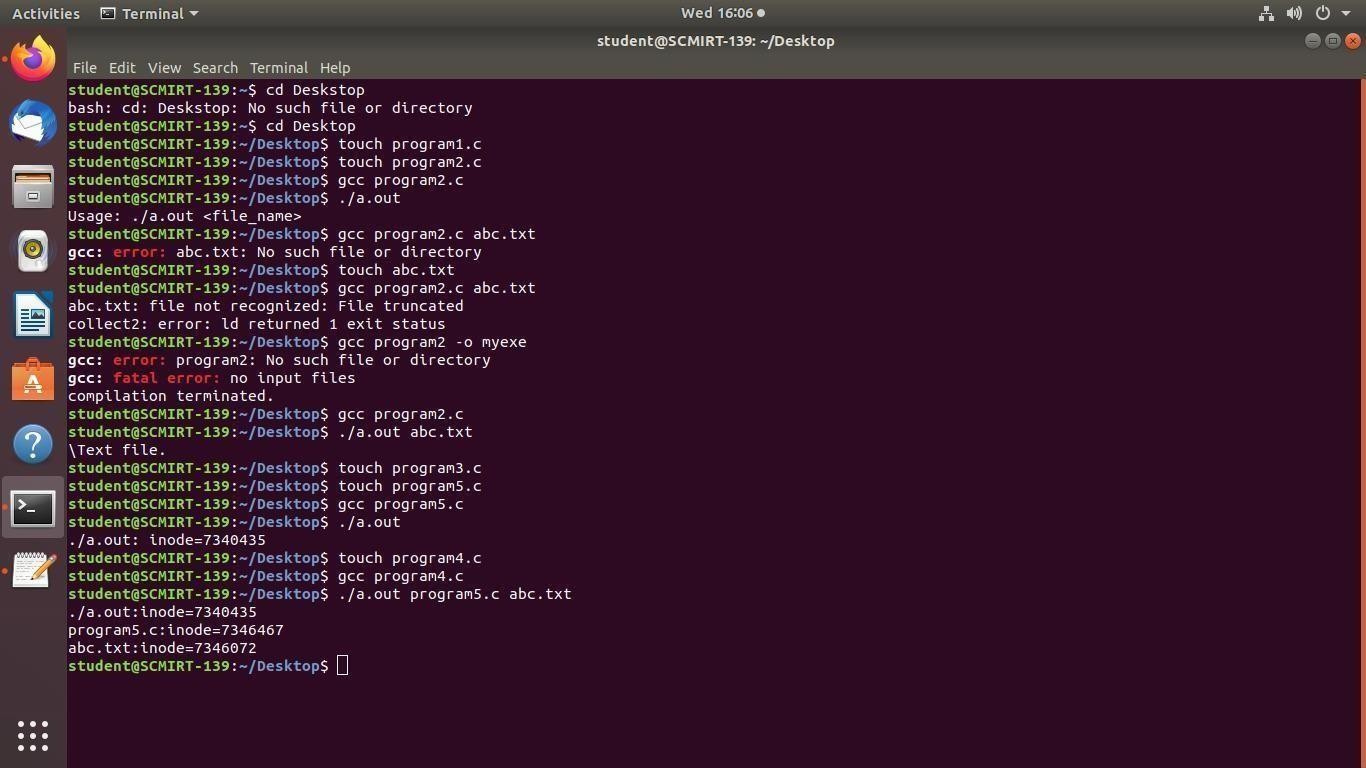
error"); continue;

}

//print node printf("%s:inode=%ld\n",argv[i],buf.st\_ino);} return 0;

}

**Output:**



3) **Print the type of file where file name accepted Through Command line.**

#include <stdio.h>

#include <stdlib.h> #include <string.h> int main(int argc, char \*argv[]) { if (argc !=

2)

{ printf("Usage: %s <file\_name>\n", argv[0]); return 1;

} char \*file\_name = argv[1];

char \*extension = strrchr(file\_name, '.'); if (extension

== NULL) { printf("File type cannot be determined.\n"); return 1;

}

if (strcmp(extension, ".txt") == 0) { printf("Text file.\n");

} else if (strcmp(extension, ".doc") == 0 || strcmp(extension,".docx")

== 0) { printf("Microsoft Word document.\n"); } else

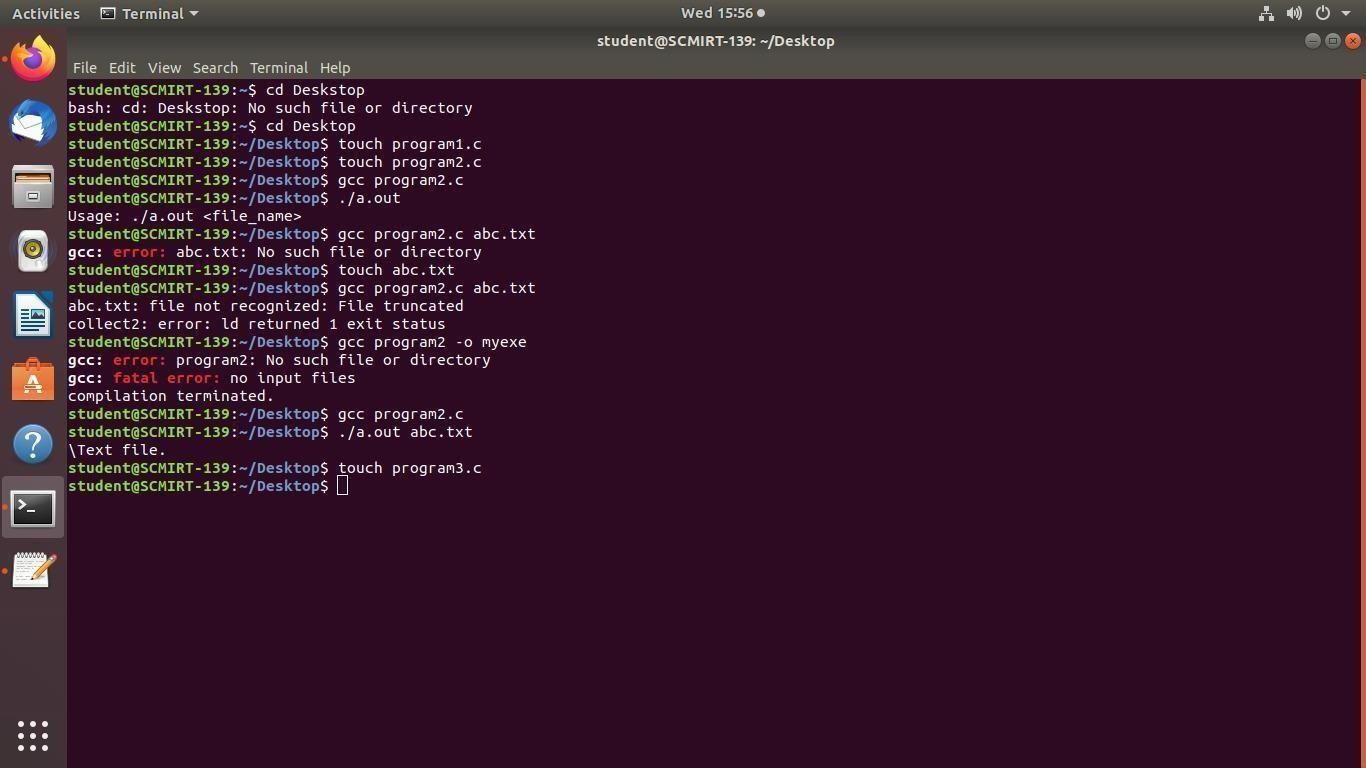
if(strcmp(extension, ".pdf") == 0) { printf("PDF document.\n");

} else { printf("File type not recognized.\n");

} return 0;

}

**Output:**



4) **Read the current directory and display the name of the files, no of files in current directory.**

#include <dirent.h> #include <stdio.h> int main() { DIR \*d;

struct dirent \*dir; int count = 0; d = opendir("."); if (d) { while ((dir = readdir(d)) !=

NULL) {printf("%s\n", dir-

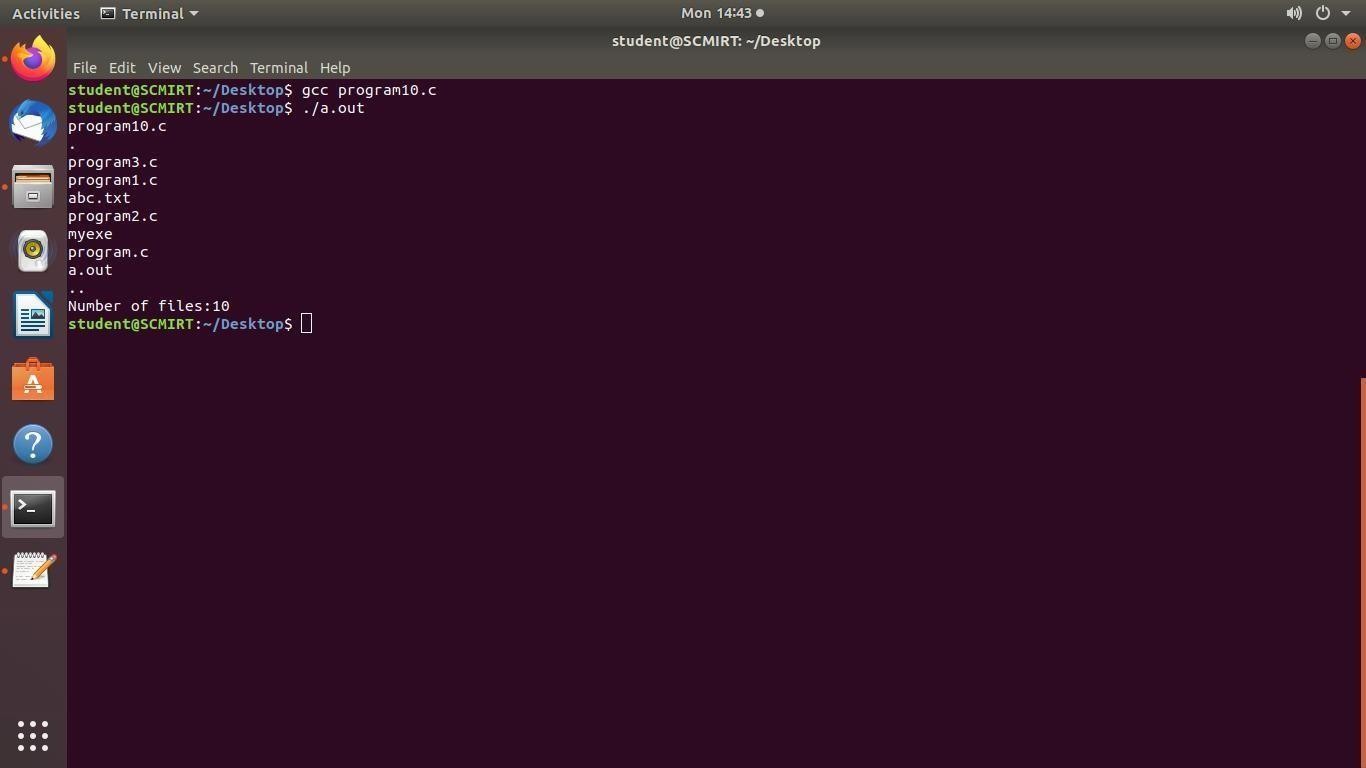
>d\_name); count++;

} closedir(d);

} printf("Number of files: %d\n", count);return 0;

}

Output:



5 **Write a C program which receives file names as command line arguments and display those filenames in ascending order according to their sizes**

Program:

#include <stdio.h>

#include <stdlib.h> #include <sys/stat.h> int compare(const void \*a, const void \*b) { struct stat \*stat\_a = (struct stat \*) a; struct stat \*stat\_b = (struct stat \*) b; return stat\_a->st\_size - stat\_b->st\_size;

}

int main(int argc, char \*argv[]) { struct stat \*stats = malloc(sizeof(struct stat) \* argc);if

(stats == NULL) { fprintf(stderr, "Failed to allocate memory.\n"); return EXIT\_FAILURE;

}

for (int i = 1; i < argc; i++) { if (stat(argv[i], &stats[i]) != 0) { fprintf(stderr, "Failed to get file size for %s.\n", argv[i]); return EXIT\_FAILURE;

} }

qsort(&stats[1], argc - 1, sizeof(struct stat), compare);for

(int i = 1; i < argc; i++) { printf("%s - %ld bytes\n", argv[i], stats[i].st\_size);

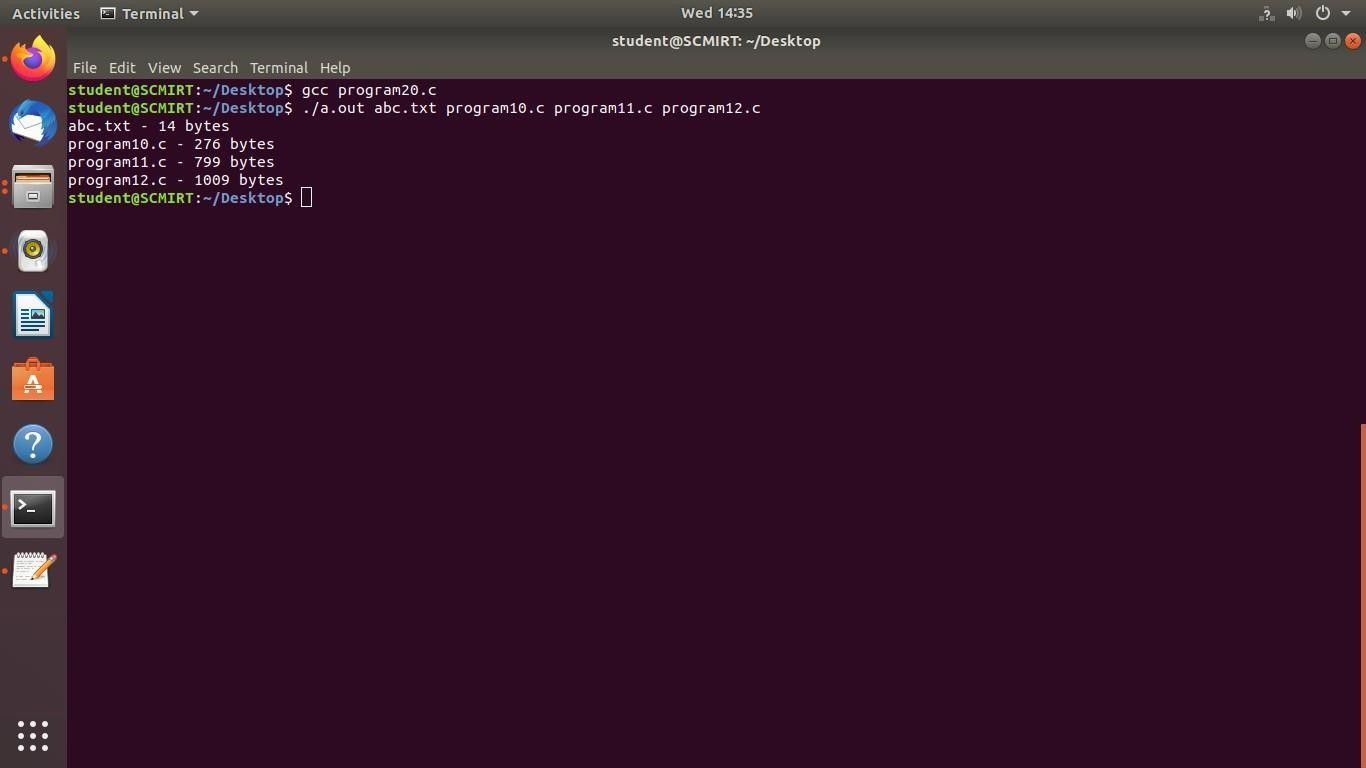
}

free(stats);

return EXIT\_SUCCESS;

}

**Output:**



6) **Write program to display all the files from current directory which arecreated in particular month programing code in c**

#include <dirent.h>

#include <stdio.h>

#include <sys/stat.h>

#include <time.h>

int main()

{DIR

\*d; struct dirent \*dir; struct stat st; int month = 4; // change to desired month (1-12)

char month\_str[4][4] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",

"Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};

d = opendir("."); if (d) { while ((dir = readdir(d)) != NULL) { if (stat(dir->d\_name, &st) == -1) { perror("stat"); continue;

}

int file\_month = localtime(&st.st\_ctime)->tm\_mon; if (file\_month == month-1) {

printf("%s\n", dir->d\_name);

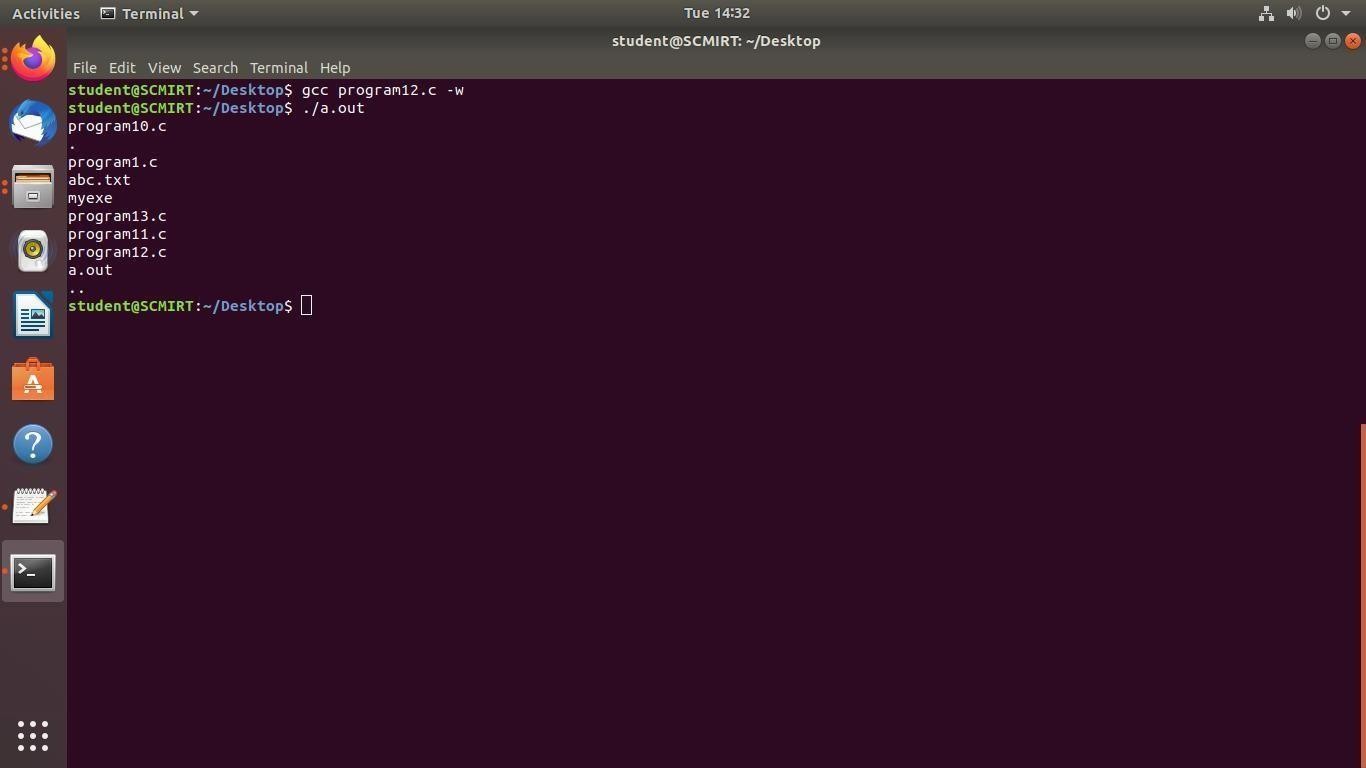
}

} closedir(d);

} else { perror("opendir"); } return 0;

}

**Output:**



7) **Write program to display all the files from current directory whose sizeis greater that n Bytes , Where n is accept from user.**

#include <dirent.h>

#include <stdio.h>

#include <sys/stat.h>

int main()

{DIR

\*d; struct dirent \*dir; struct stat st; long min\_size;

printf("Enter minimum file size in bytes: "); scanf("%ld", &min\_size);

d = opendir("."); if (d) { while ((dir = readdir(d)) != NULL) { if (stat(dir->d\_name, &st) == -1) { perror("stat"); continue;

}

if (st.st\_size > min\_size) {

printf("%s\n", dir->d\_name);

}

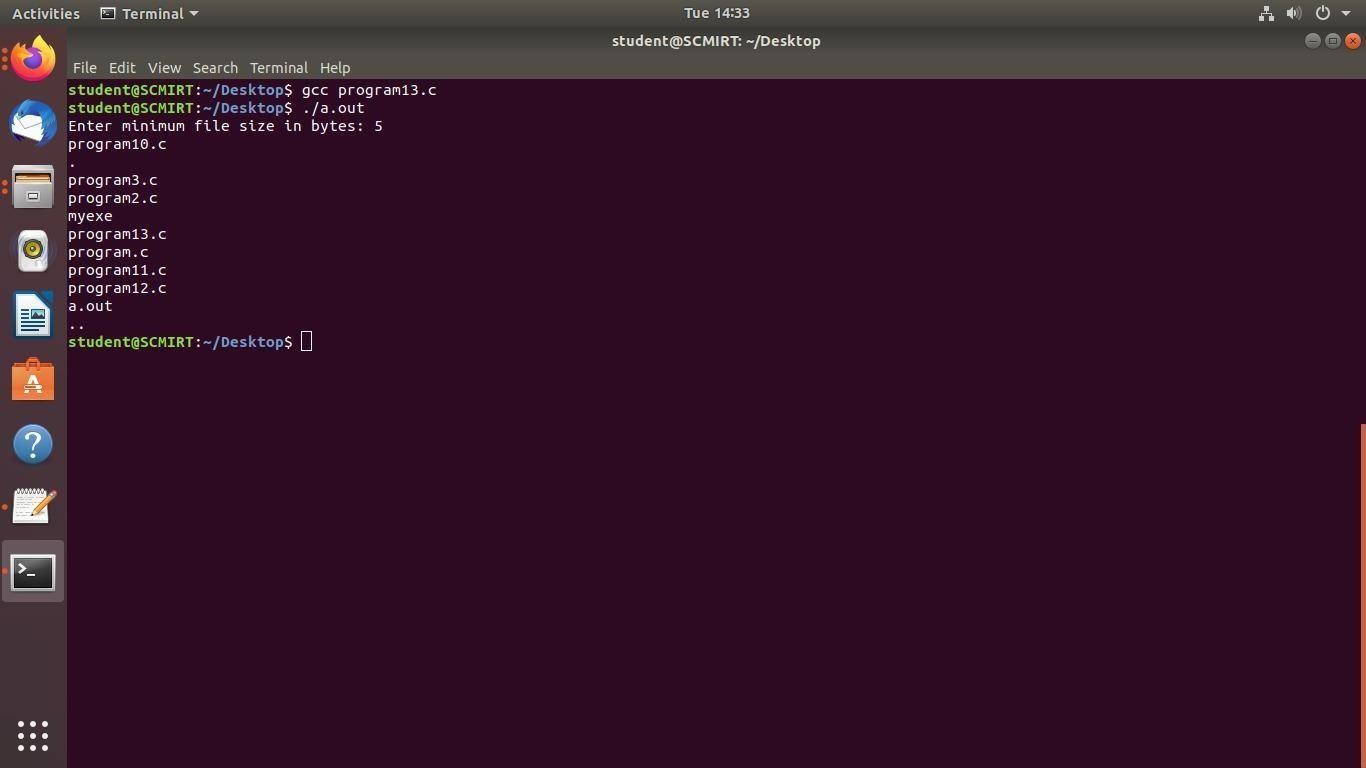
}

closedir(d);

} else { perror("opendir"); } return 0;

}

Output:



8).**Write Program that demonstrates redirection of standard output to a file.**

Program:

#include <stdio.h> int main() {

FILE \*fp;

// Open a file for writing

fp = freopen("output.txt", "w", stdout);

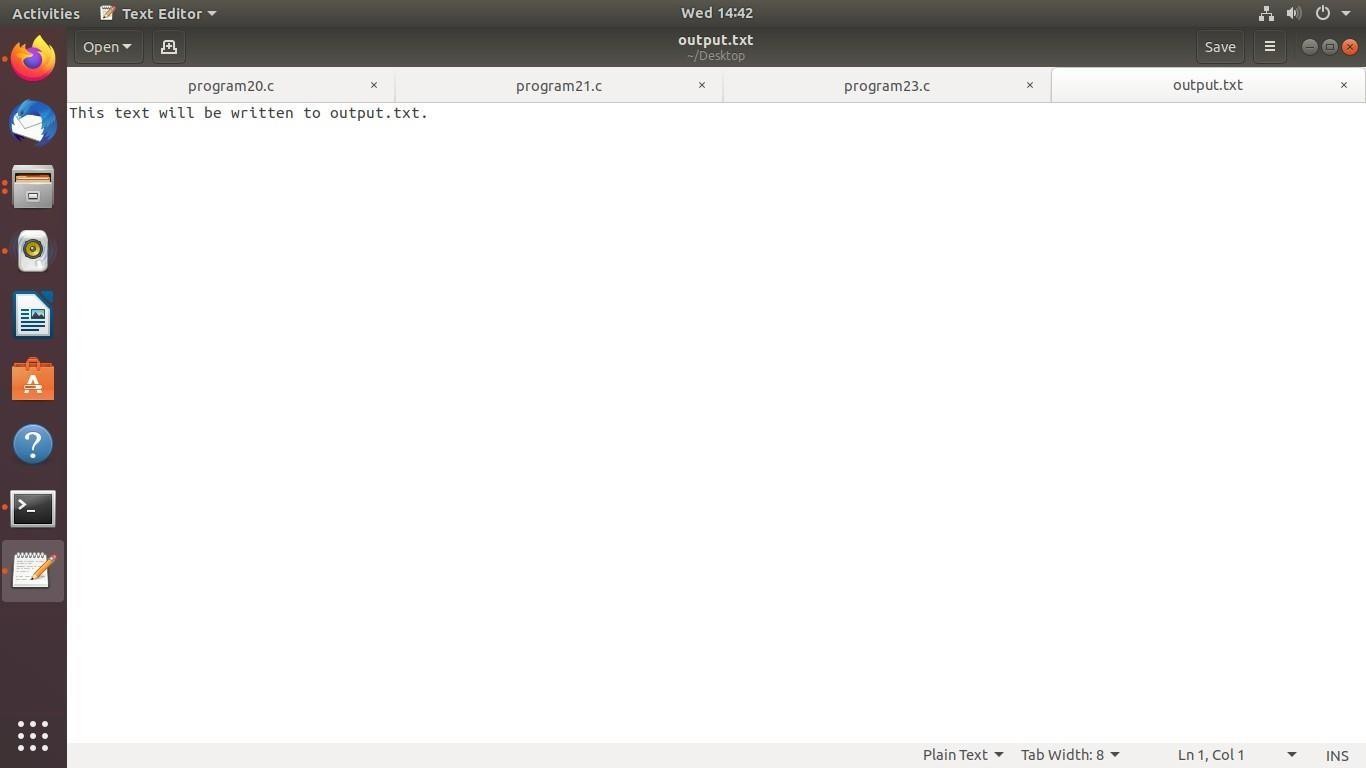
// Print some text printf("Hello, world!\n");

// Close the file fclose(fp); return

0;

}

Output:



9) **To generate parent process to write unnamed Pipe and will read from it.**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<string.h> #define BUFFER\_SIZE 25 int main() {int fd[2]; pid\_t pid;

char write\_msg[BUFFER\_SIZE]="Hello, child!";

char read\_msg[BUFFER\_SIZE];

if(pipe(fd)<0) {perror("pipeerror"); exit(1);

}

pid=fork(); if(pid<0) {perror("fork error"); exit(1);

}else if (pid==0) { close(fd[1]); if(read(fd[0],

read\_msg,BUFFER\_SIZE)<0){perror("read

error"); exit(1);

}

printf("child read from pipe:%s\n", read\_msg); close(fd[0]);

exit(0); }else { close(fd[0]);

if(write(fd[1],write\_msg, strlen(write\_msg)+1)<0){

perror("write error"); exit(1);

}

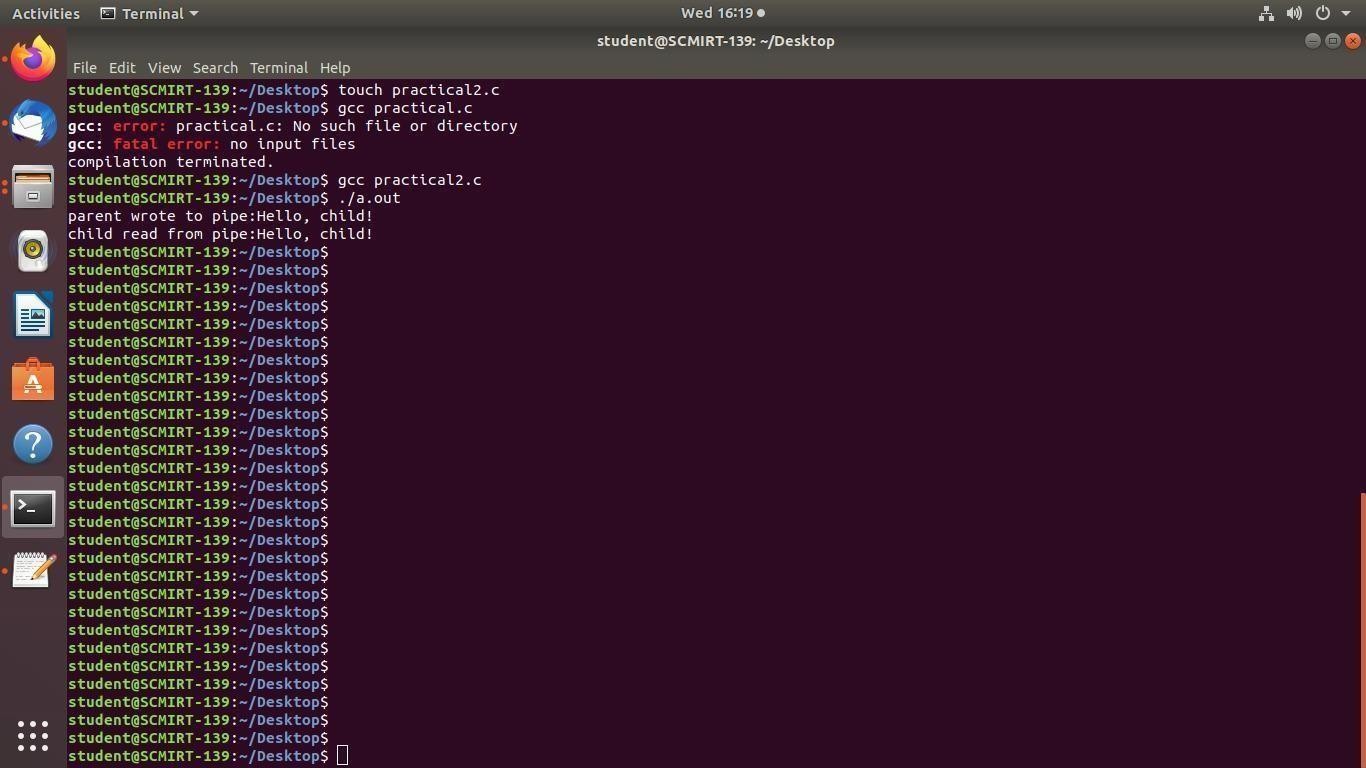
printf("parent wrote to pipe:%s\n",write\_msg); close(fd[1]); exit(0);

}

return 0;

}

Output:



10) **To handle the two way communication between Parent and child using pipe.**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<string.h>

#define BUFFER\_SIZE 1024 //buffer size in bytes int main()

{

int fd[2]; //file descriptors for the pipe pid\_t pid; //processID char parent\_message[]="hello from parent!"; char child\_message[]="hello from child!"; char buffer[434]; //

CREATE PIPE if(pipe(fd)>0)

{

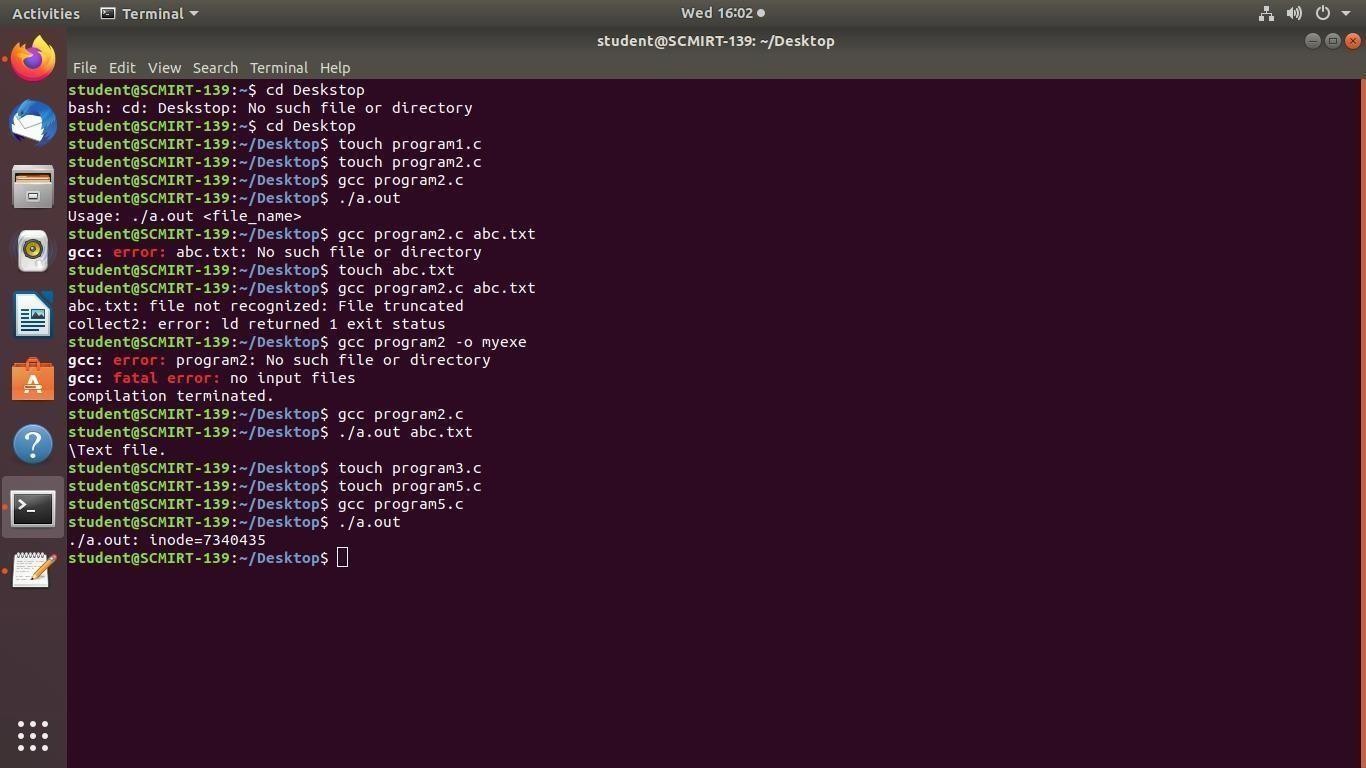
perror("pipe error");

printf("helloo"); exit(1);

} return 0;

}

**Output:**



11) **To demonstrate the use of atexit() function.**

#include <stdio.h> #include <stdlib.h> void cleanup1() { printf("Cleanup 1\n");

} void cleanup2() { printf("Cleanu

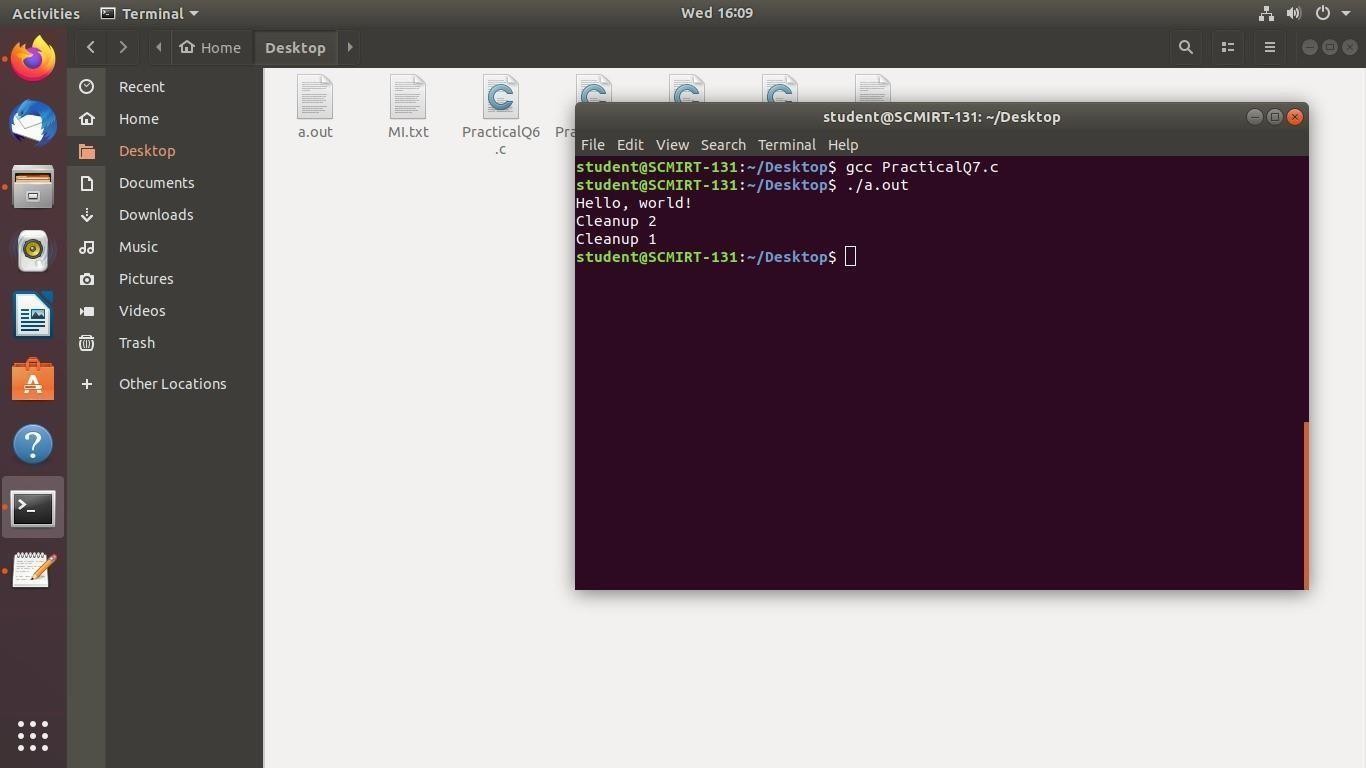
p2\n"); }

int main() { atexit(cleanup1); atexit(cleanup2);

printf("Hello, world!\n"); return 0;

}

Output:



12) **Write a C program to implement the following unix/linux command (use fork, pipe and exec system call) ls –l | wc –l**

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h> #include <sys/wait.h> int main() { int fd[2]; pid\_t pid1, pid2; // create pipe if (pipe(fd) == -1) { perror("pipe");

exit(EXIT\_FAILURE);

}

// fork first child (ls - l)pid1 = fork(); if (pid1 == -1) { perror("fork");

exit(EXIT\_FAILURE);

} else if (pid1 == 0) {

// redirect stdout to write end of pipe dup2(fd[1], STDOUT\_FILENO);

// close unused read end of pipe close(fd[0]); // execute ls -l execlp("ls", "ls", "-l",

NULL);perror("execlp");

exit(EXIT\_FAILURE);

}

// fork second child (wc -l) pid2 = fork(); if (pid2 == -1) {

perror("fork");

exit(EXIT\_FAILURE);

} else if (pid2 == 0) { // redirect stdin to read end of pipe dup2(fd[0], STDIN\_FILENO); // close unused write end of pipe close(fd[1]); // execute wc -l execlp("wc", "wc", "-l", NULL);perror("execlp"); exit(EXIT\_FAILURE);

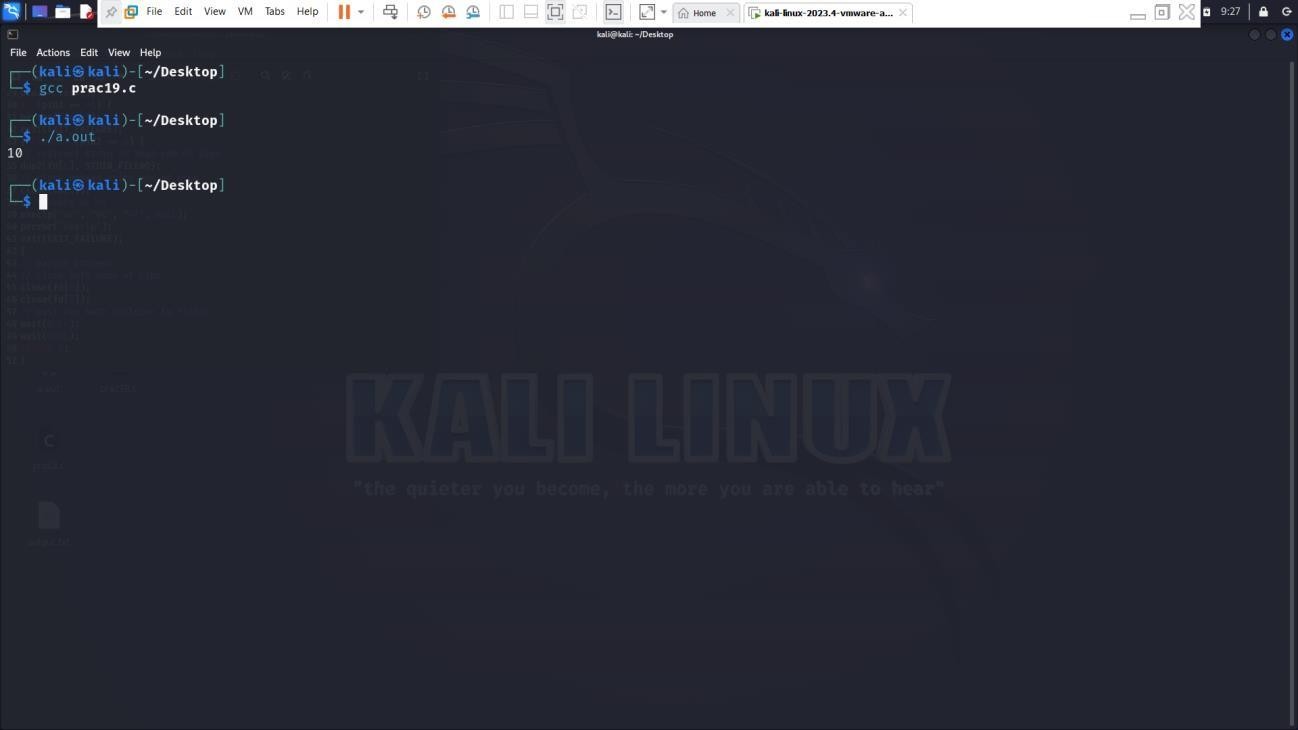
}

// parent process // close both ends of pipe close(fd[0]); close(fd[1]); // wait for both children to finish wait(NULL); wait(NULL

);return 0;

}

Output



13) **To create ‘n’ children . When the children will terminate,display totalcumulative time children Spent in user and kernel mode.**

#include<stdio.h>

#include<stdlib.h>

#include<sys/time.h>

#include<sys/resource.h>

#include<sys/wait.h>

#include<unistd.h> int main(int argc , char \*\*argv){ int n = atoi(argv[1]); int i,status; pid\_t pid; struct rusage r\_usage;

struct timeval user\_time,kernel\_time; long total\_user\_usec=0, total\_kernal\_usec=0;

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

= fork(); if(pid < 0){ perror("fork error");

exit(1); }

else if(pid==0){

printf("child %d started \n",i+1); sleep(5); printf("child %d finished \n",i+1); exit(0);

}

}

while((pid = wait(&status))>0){ if(getrusage(RUSAGE\_CHILDREN,& r\_usage)

< 0){perror("getrusage error");

exit(1); }

user\_time = r\_usage.ru\_utime; kernel\_time = r\_usage.ru\_stime;

printf("child %d: user time =%ld microseconds,kernel time = %ld

microseconds.\n",pid,user\_time.tv\_usec,kernel\_time.tv\_usec); total\_user\_usec

+= user\_time.tv\_usec; total\_kernal\_usec += kernel\_time.tv\_usec;

} printf("Total time spend :%ld \n",total\_user\_usec); printf("Total time spend :%ld \n",total\_kernal\_usec);

return 0;

}

**Output**



14) **Write a program that illustrates how to execute two commands concurrently with a pipe.**

Program:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h> #include <sys/wait.h> int main() {

int pipefd[2]; pid\_t pid1, pid2; // Create a pipe if (pipe(pipefd) == -1)

{perror("pipe"); exit(EXIT\_FAILURE);

}

// Fork first child process pid1 = fork(); if (pid1 == -1) {

perror("fork");

exit(EXIT\_FAILURE);

} else if (pid1 == 0)

{

// Child process 1

// Close the read end of the pipe

close(pipefd[0]);

// Redirect stdout to the write end of the pipe dup2(pipefd[1], STDOUT\_FILENO);

// Execute the first command execlp("ls", "ls", NULL); // Exit the child process if execlp fails perror("execlp"); exit(EXIT\_FAILURE);

}

// Fork second child process pid2 = fork(); if (pid2 == -1) {

perror("fork");

exit(EXIT\_FAILURE);

} else if (pid2 == 0)

{

// Child process 2 // Close the write end of the pipe close(pipefd[1]);

// Redirect stdin to the read end of the pipe dup2(pipefd[0], STDIN\_FILENO); // Execute the second command execlp("wc", "wc", "-l", NULL); // Exit the child process if execlp fails perror("execlp"); exit(EXIT\_FAILURE);

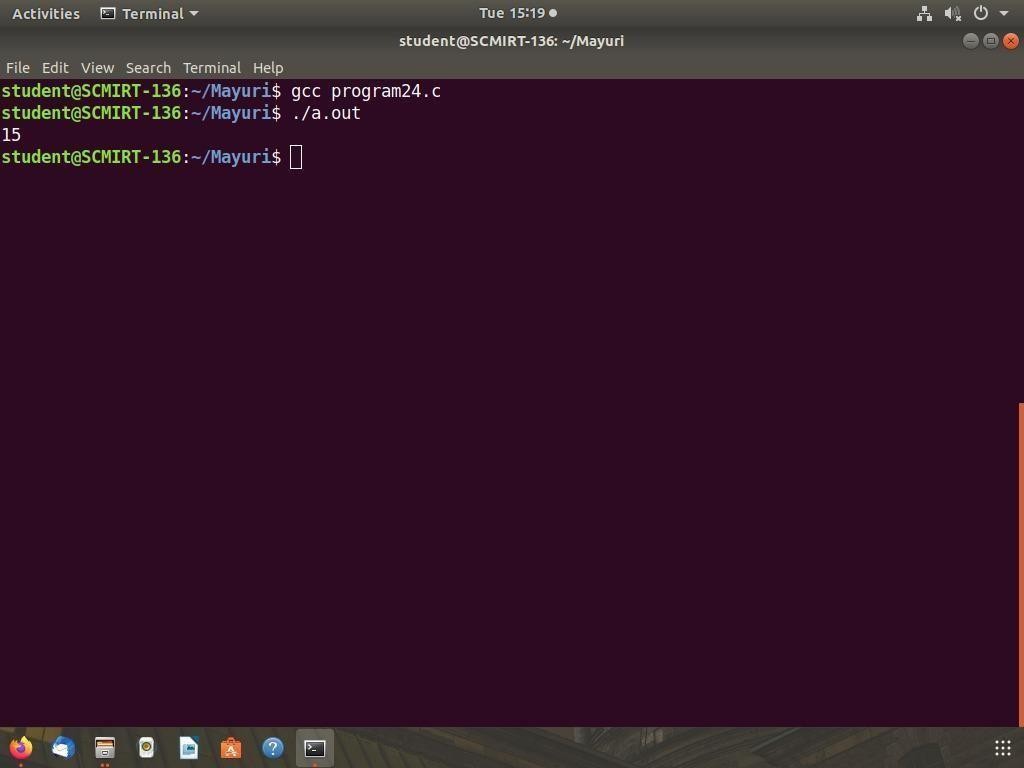
}

// Parent process // Close both ends of the pipe close(pipefd[0]); close(pipefd[1]);

// Wait for both child processes to exit waitpid(pid1, NULL, 0); waitpid(pid2, NULL, 0); return 0;

}

Output:



15) **Write a C program which create a child process which catch a signal sighup, sigint and sigquit. The Parent process send a sighup or sigint signal afterevery 3 seconds, at the end of 15 second parent send sigquit signal to child andchild terminates my displaying message "My Papa has Killed me!!!”.**

Program:

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#include <unistd.h> #include <sys/wait.h> volatile sig\_atomic\_t flag = 0; void signal\_handler(int signal) { flag = 1;

} int main() { pid\_t pid = fork();if (pid == -

1) { fprintf(stderr, "Failed to fork.\n"); return EXIT\_FAILURE;

}

if (pid == 0) { // child process

signal(SIGHUP, signal\_handler); signal(SIGINT, signal\_handler); signal(SIGQUIT, signal\_handler);while (1) { if (flag) { printf("Signal received.\n"); flag = 0;

} sleep(1);

}

} else { // parent process for (int i = 1; i <= 10;

i++) {sleep(3); if (i % 2 == 0) {

kill(pid, SIGHUP);

} else { kill(pid, SIGINT);

} } kill(pid,

SIGQUIT);int

status; wait(&status); if (WIFEXITED(status)) {

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

} else if (WIFSIGNALED(status)) { printf("Child process terminated by signal %d.\n", WTERMSIG(status));

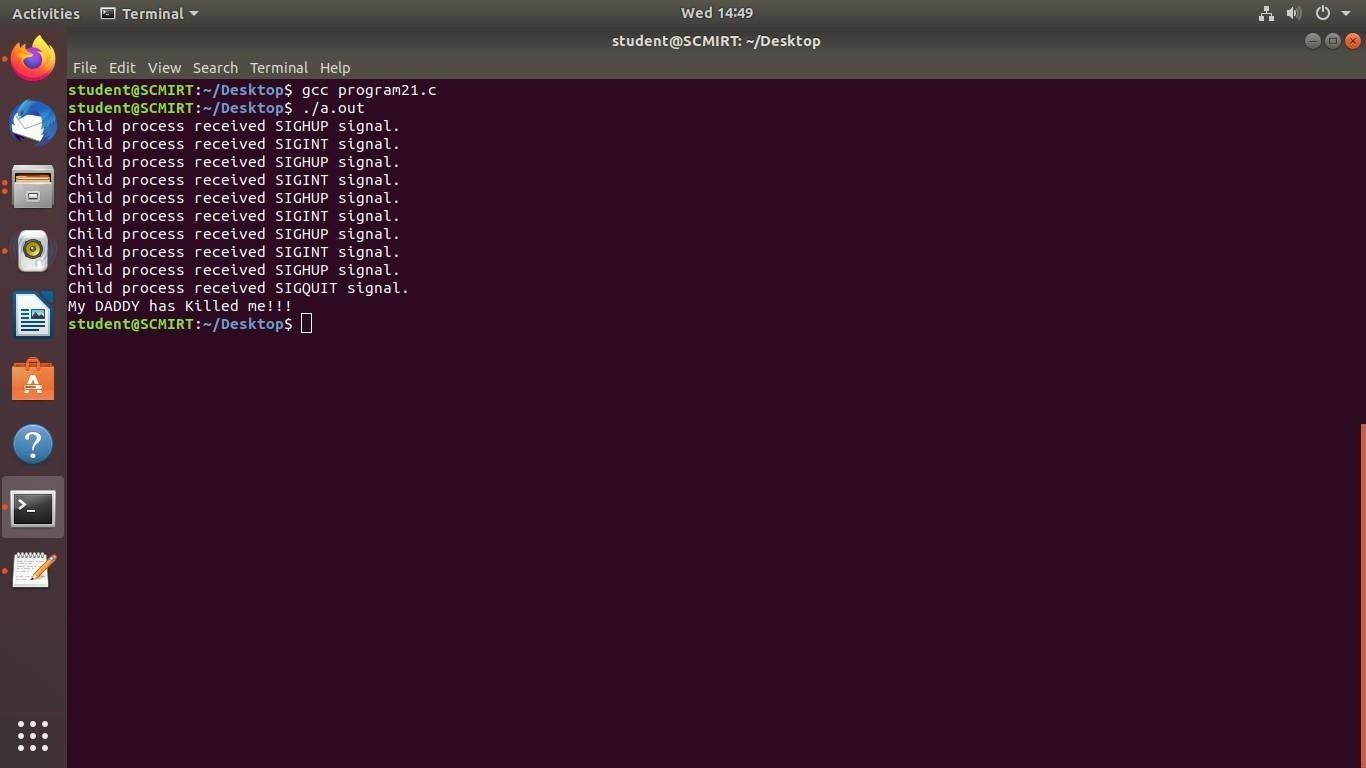
}

}

return EXIT\_SUCCESS;

}

**Output:**



16) **Write a C program that illustrates suspending and resuming processes using signals.**

Program:

#include <stdio.h> #include <stdlib.h>

#include

<unistd.h> #include <signal.h>

void sigint\_handler(int signum) {

printf("Caught signal %d (SIGINT)\n", signum);

} int main() { struct sigaction sa; sigemptyset(&sa.sa\_mask); sa.sa\_flags = 0; sa.sa\_handler = sigint\_handler; sigaction(SIGINT, &sa, NULL);

printf("Press Ctrl+C to suspend the program...\n"); while (1) { sleep(1);

printf("Still running...\n"); kill(getpid(), SIGSTOP); printf("Resuming...\n");

} return 0;

}

Output:

