Name-Maroti Bhise

Roll No – 45

Q1. Create a file with hole in it

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

int fd = open("file\_with\_hole.txt", O\_WRONLY | O\_CREAT, 0644);

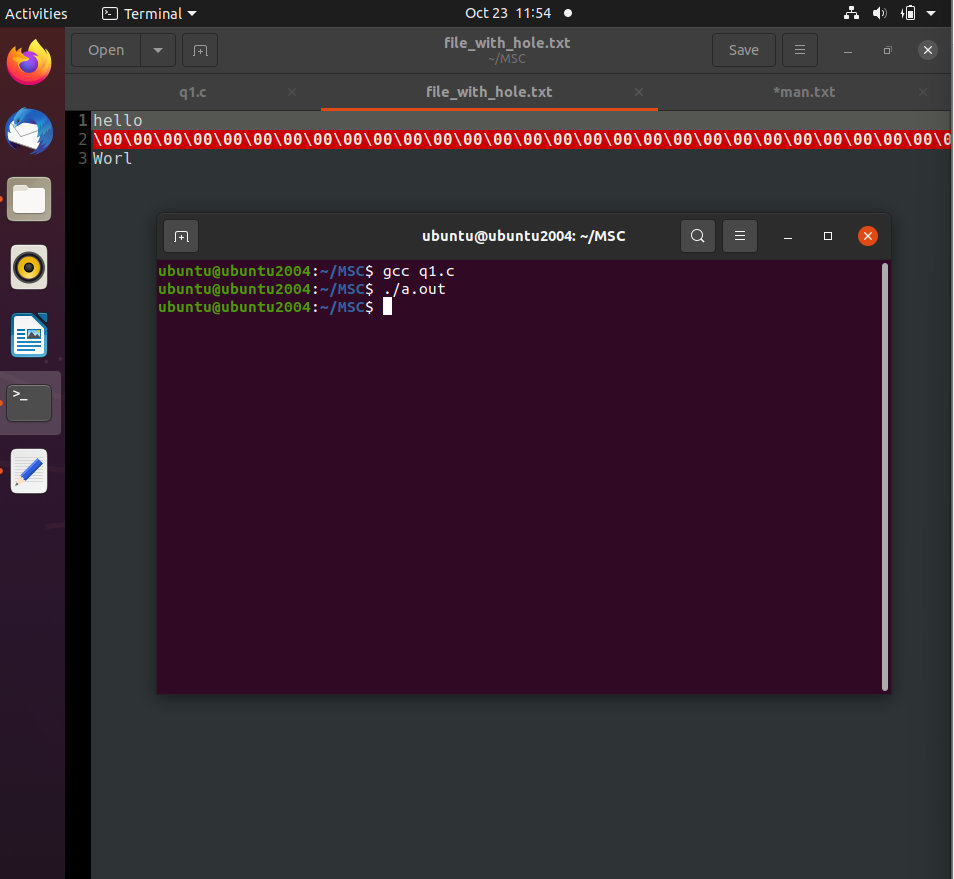
if (fd < 0) { perror("open"); return 1;

}

write(fd, "Hello", 5); lseek(fd, 1000, SEEK\_CUR); write(fd, "World", 5);

close(fd);

return 0; }



Q2 Take multiple files as Command Line Arguments and print their inode number

//inode of file—file name---inodes.c

#include<stdio.h>

#include<stdlib.h>

#include<sys/stat.h>

int main(int argc,char \*argv[]){

struct stat ft;

for(int i=1;i<argc;i++)

{

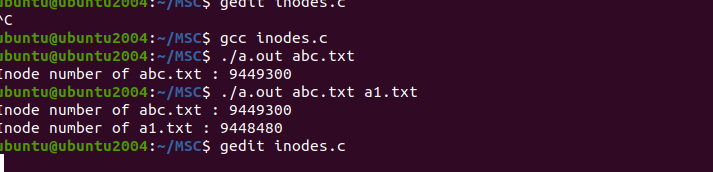
if(stat(argv[i],&ft)==0){

printf("Inode number of %s : %lu\n",argv[i],ft.st\_ino);

}

}

}



Q3 . Write a C program to find file properties such as inode number, number of hard link, File permissions, File size, File access and modification time and so on of a given file using stat() system call.

File name—file\_prop.c

//print the properties of file

#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <time.h>

void printpermissions(mode\_t mode)

{

printf("file permissions: ");

printf((mode & S\_IRUSR) ? "r" : "-");

printf((mode & S\_IWUSR) ? "w" : "-");

printf((mode & S\_IXUSR) ? "x" : "-");

printf((mode & S\_IRGRP) ? "r" : "-");

printf((mode & S\_IWGRP) ? "w" : "-");

printf((mode & S\_IXGRP) ? "x" : "-");

printf((mode & S\_IROTH) ? "r" : "-");

printf((mode & S\_IWOTH) ? "w" : "-");

printf((mode & S\_IXOTH) ? "w" : "-");

}

int main(int argc,char \* argv[]){

struct stat ft;

if(stat(argv[1],&ft)==0)

{

printf("file: %s\n",argv[1]);

printf("Inode: %ld\n",ft.st\_ino);

printf("Size: %ld\n",ft.st\_size);

printf("Number of hard links : %ld\n",ft.st\_nlink);

printf("Access Time: %s\n",ctime(&ft.st\_atime));

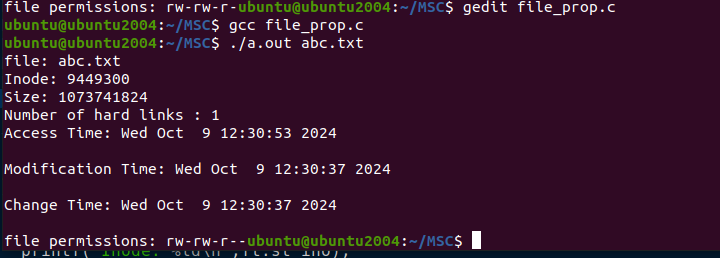
printf("Modification Time: %s\n",ctime(&ft.st\_mtime));

printf("Change Time: %s\n",ctime(&ft.st\_ctime));

printpermissions(ft.st\_mode);

}

}



Q4. Print the type of file where file name accepted through Command Line

// print type of file

//program to print type of file

#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

void printFileType(mode\_t mode)

{

if(S\_ISDIR(mode)) printf("File Type : Directory\n");

else if(S\_ISREG(mode)) printf("File Type : Regular\n");

else if(S\_ISCHR(mode)) printf("File Type : Character\n");

else if(S\_ISBLK(mode)) printf("File Type : Block device\n");

else printf("File Type is unknown");

}

int main(int argc,char \*argv[])

{

if(argc>1)

{

struct stat ft;//file discriptor

if(stat(argv[1],&ft)==0)

{

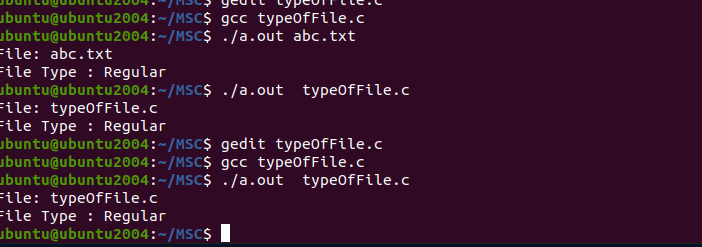
printf("File: %s\n",argv[1]);

printFileType(ft.st\_mode);

}

}

}



Q5. Write a C program to find whether a given file is present in current directory or not.

File Name-Check\_file.c

//Check the file present in the directory or not

#include<stdio.h>

#include<stdlib.h>

#include<sys/stat.h>

int main(int argc,char \*argv[])

{

if(argc>1)

{

struct stat ft;

if(stat(argv[1],&ft)==0)

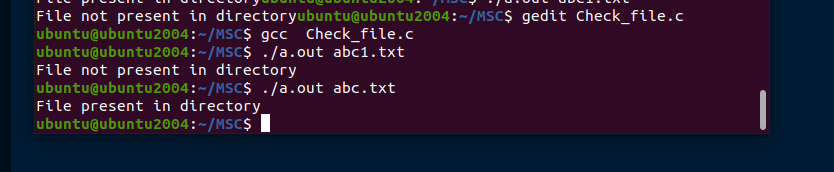
printf("File present in directory\n");

else

printf("File not present in directory\n");

}

}



Q6. Write a C program that a string as an argument and return all the files that begins with that name in the current directory. For example > ./a.out foo will return all file names that begins with foo

//File Name -Start

// list the file that start with given string

#include<stdio.h>

#include<string.h>

#include<sys/stat.h>

#include<dirent.h>

int main(int argc,char \*argv[])

{

if(argc>1)

{

DIR \* dir=opendir("."); //open current directory

if(!dir) printf("Error");

struct dirent \* entry;

int flag=0;

while((entry= readdir(dir))){

if(strncmp(entry->d\_name,argv[1],strlen(argv[1]))==0)

{

flag++;

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

}

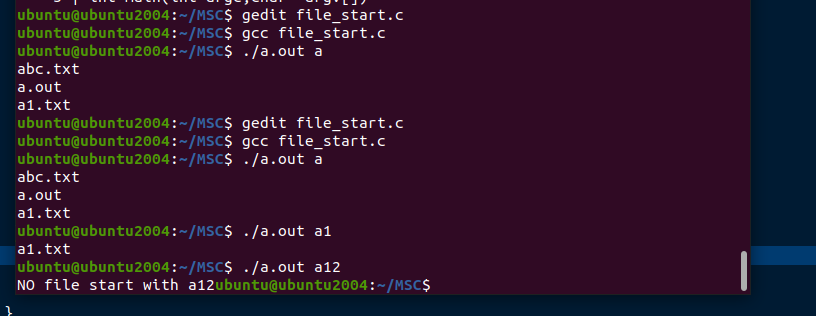
}

if(flag==0)

printf("NO file start with %s",argv[1]);

}

}



Q7. Read the current directory and display the name of the files, no of files in current directory

File Name- files\_list.c

//print the list of file present in current directory.

#include<stdio.h>

#include<dirent.h>

int main()

{

DIR \* dir;

struct dirent \* entry;

int count=0;

dir=opendir("."); //open current direcotry

if(!dir){

printf("error");

return 0;

}

printf("Files in the directory:\n");

while( (entry=readdir(dir)) !=NULL){

if(entry->d\_name[0]!='.')

{

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

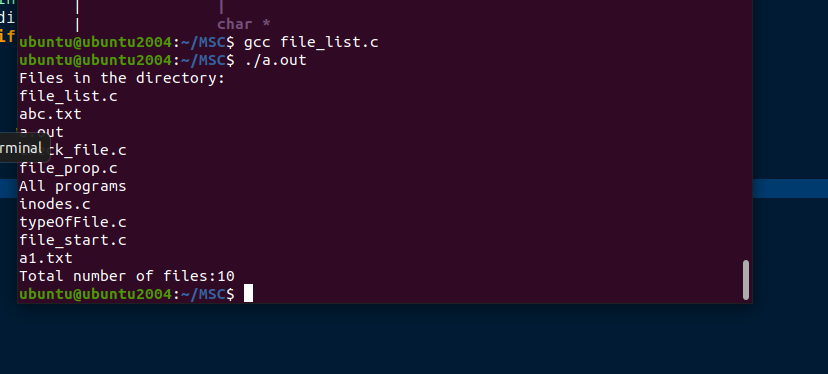
count++;

}

}

printf("Total number of files:%d\n",count);

}



Q8. Write a C program which receives file names as command line arguments and display SPPU M.Sc. Computer Science Syllabus 2023-24 18 those filenames in ascending order according to their sizes. I) (e.g $ a.out a.txt b.txt c.txt, …)

//display the files in ascending order of file size.

#include<stdio.h>

#include<sys/stat.h>

#include<string.h>

struct files

{

char name[30];

long int size;

}f[20];

int main(int argc,char \* argv[])

{

if(argc<2)//if command line does not have argument.

{

printf("No argument in command line arguemnt");

return 0;

}

struct stat st;

for(int i=1;i<argc;i++)

{

if(stat(argv[i],&st)==0){

strncpy(f[i-1].name,argv[i],strlen(argv[i]));

f[i-1].size=st.st\_size;

}

else return 1;

}

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

printf("%s\n",f[i].name);

printf("hello");

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

for(int j=i+1;j<argc-1;j++)

if(f[i].size>f[j].size)

{

struct files temp=f[i];

f[i]=f[j];

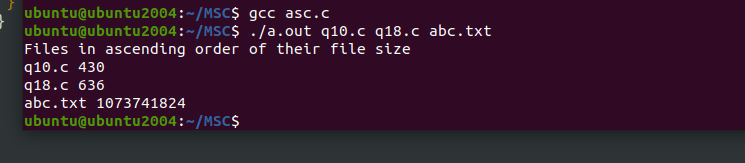
f[j]=temp;

}

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

printf("%s %ld\n",f[i].name,f[i].size);

}



Q9. Display all the files from current directory which are created in particular month

//display the files which are created in particular month

#include<stdio.h>

#include<dirent.h>

#include<sys/stat.h>

#include<time.h>

#include<stdlib.h>

void display\_files(int month,int year)

{

DIR \*dir;

struct dirent \*entry;//to store the file entry

struct stat ft;//store file info

struct tm \*tm\_info;//to handle time

dir=opendir(".");//open directory

printf("\nFiles created in %d %d:\n",month,year);

while((entry=readdir(dir))!=NULL)//read each entry in the directory

{

if(entry->d\_type==DT\_REG){ //check if it is a regular file

if(stat(entry->d\_name,&ft)==0){//info retrived successfully

tm\_info=localtime(&ft.st\_mtime);//get last modified time

if(tm\_info->tm\_year + 1900==year && tm\_info->tm\_mon + 1==month)

{

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

}

}

}

}

closedir(dir);

}

int main()

{

int month;

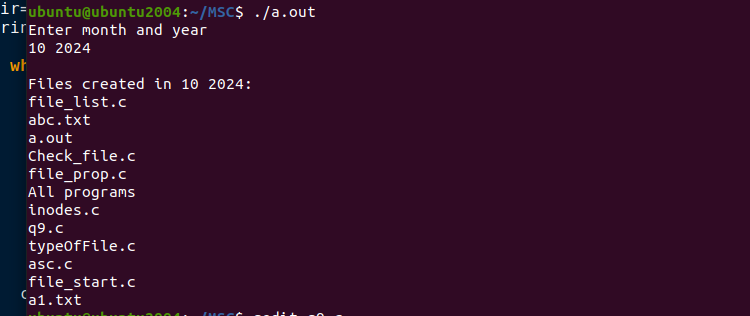
int year;

printf("Enter month and year\n");// input should be in the form of 10 2024

scanf("%d%d",&month,&year);

display\_files(month,year);

}



q10.Display all the files from current directory whose size is greater that n Bytes Where n is accept from user

#include<stdio.h>

#include<stdlib.h>

#include<dirent.h>

#include<sys/stat.h>

int main()

{

int n;

printf("Enter number of size\n");

scanf("%d",&n);

DIR \*dir;

struct dirent \* entry;

struct stat ft;

dir=opendir(".");

while((entry=readdir(dir))!=NULL)

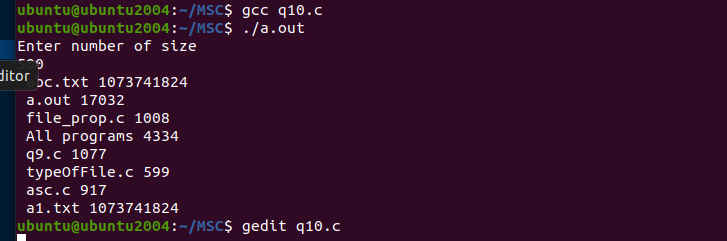
if(entry->d\_type==DT\_REG)

if(stat(entry->d\_name,&ft)==0)

if(ft.st\_size>n)

printf(" %s %ld\n",entry->d\_name,ft.st\_size);

}



q11. Write a C Program that demonstrates redirection of standard output to a file.ge

// cprogram that redirect standard out put to a file.

#include<stdio.h>

#include<fcntl.h>

#include<unistd.h>

#include<stdlib.h>

int main(){

int fd=open("abcd.txt",O\_WRONLY);

if(fd<0)

{ printf("error"); return 0; }

if(dup2(fd,STDOUT\_FILENO)<0)

{

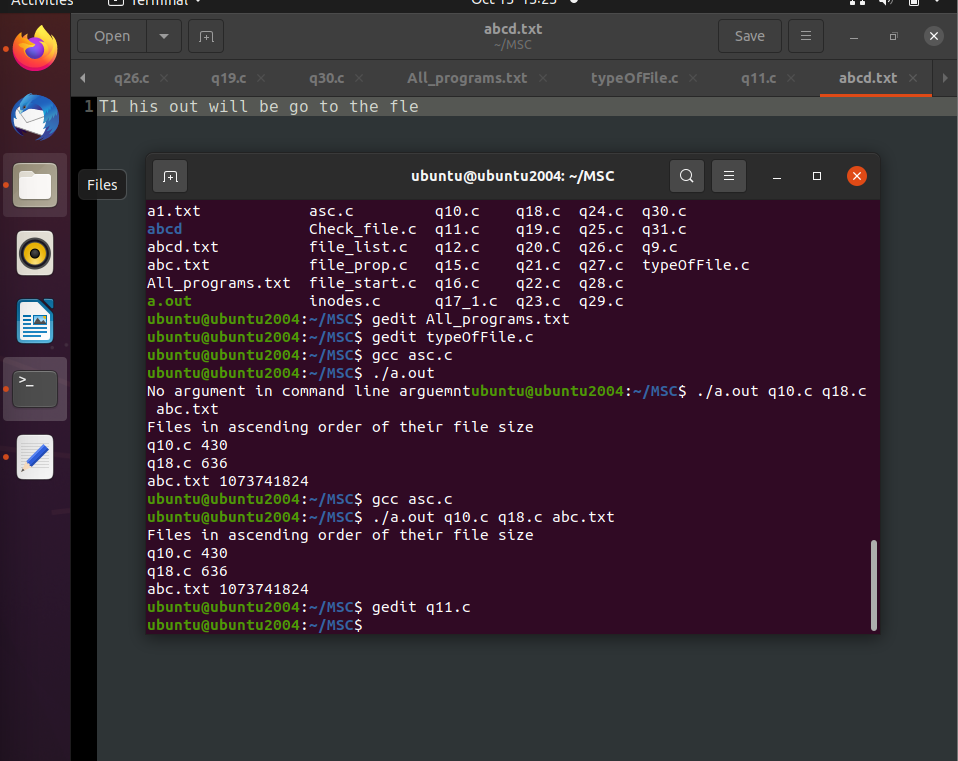
printf("error 2"); return 0;

}

close(fd);

printf("T1 his out will be go to the fle");

}



q12. Write a C program that will only list all subdirectories in alphabetical order from

current directory

// c program that will only list all subdirectories in alphabetical order from current directory

#include<stdio.h>

#include<dirent.h>

#include<string.h>

int main()

{

DIR \* dir;

struct dirent \* entry;

dir=opendir(".");//open current directories

printf("subdirectories in current directory\n");

while((entry=readdir(dir))!=NULL)

{

if(entry->d\_type==DT\_DIR &&

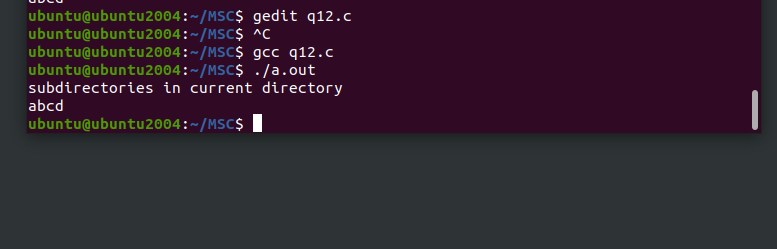
strcmp(entry->d\_name,"." ) &&

strcmp(entry->d\_name,".."))

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

}

}



//q13. Write a C program that redirects standard output to a file output.txt. (use of dup and open system call)

// cprogram that redirect standard out put to a file.

#include<stdio.h>

#include<fcntl.h>

#include<unistd.h>

#include<stdlib.h>

int main(){

int fd=open(".txt",O\_WRONLY);

if(fd<0)

{ printf("error"); return 0; }

if(dup2(fd,STDOUT\_FILENO)<0)

{

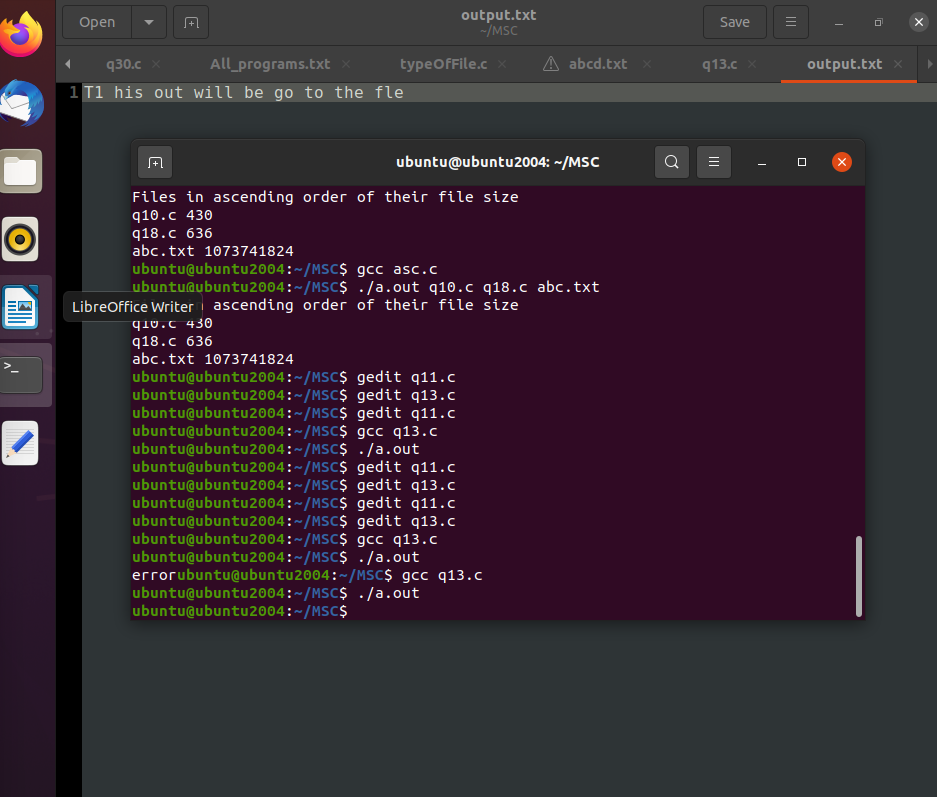
printf("error 2"); return 0;

}

close(fd);

printf("T1 his out will be go to the file");

}



//q14.Write a C program to Identify the type (Directory, character device, Block device, Regular file, FIFO or pipe, symbolic link or socket) of given file using stat() system call

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

void printFileType(mode\_t mode)

{  
 if(S\_ISREG(mode)) printf(“File Type: Regular\n”);

else if(S\_ISCHR(mode)) printf(“File Type: Character\n”);

else if(S\_ISBLK(mode)) printf(“File Type: Block device\n”);

else printf(“File Type is unknown”);

}

int main()  
 {

if(argc>1)

{

Struct stat ft;//file descriptor

if(stat(argv[i],&ft)==0){

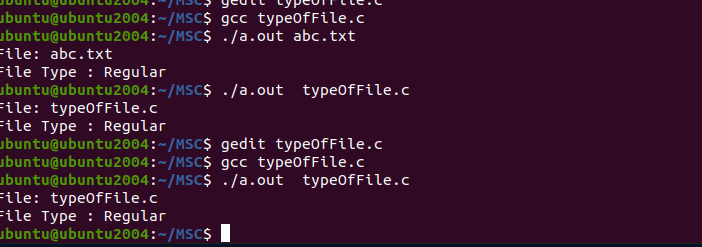
printf(“File: %s\n”,argv[1]);

printFileType(ft.st\_mode);

}

}

}



//q15. Generate parent process to write unnamed pipe and child will read from it

#include<stdio.h>

#include<unistd.h>

#include<string.h>

#define BS 100

int main()

{

int pipe\_fd[2];//file discriptor for pipe

pid\_t pid;

char buffer[BS];

pipe(pipe\_fd);//create pipe

pid=fork();

if(pid>0)//parent process

{

close(pipe\_fd[0]);//close read end

char \*msg="This is parent process";

write(pipe\_fd[1],msg,strlen(msg)+1);

close(pipe\_fd[1]);

//wait(NULL);

}else

{

close(pipe\_fd[1]);

read(pipe\_fd[0],buffer,BS);

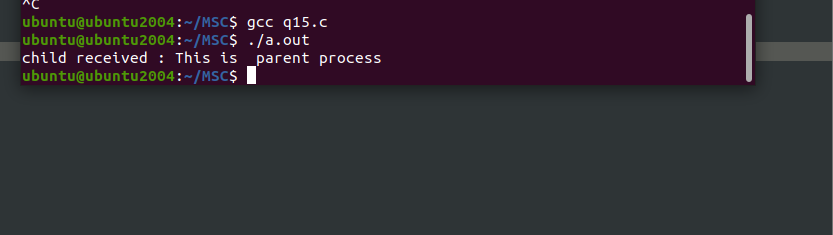
printf("child received : %s\n",buffer);

close(pipe\_fd[0]);

}

return 0;

}



//q16. Handle the two-way communication between parent and child processes using pipe

// two way communication

#include<stdio.h>

#include<unistd.h>

#include<string.h>

#define BS 100

int main()

{

int pipe1[2],pipe2[2];

pid\_t pid;

char buffer[BS];

pipe(pipe1);//create pipe

pipe(pipe2);

pid=fork();

if(pid>0)//parent process

{

close(pipe1[0]);//close read end

close(pipe2[1]);//close write end

char \*msg="This msg from parent process";

write(pipe1[1],msg,strlen(msg)+1);

read(pipe2[0],buffer,BS);

printf("Parent received : %s\n",buffer);

}else

{

close(pipe1[1]);

close(pipe2[0]);

read(pipe1[0],buffer,BS);

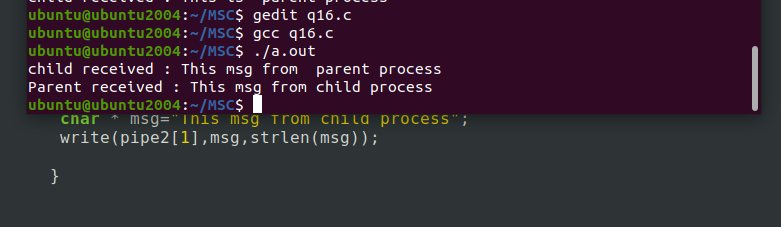
printf("child received : %s\n",buffer);

char \* msg="This msg from child process";

write(pipe2[1],msg,strlen(msg));

}

}



//q17. Demonstrate the use of atexit() function

//use of atexit() function

#include<stdio.h>

#include<stdlib.h>

void cleanup(){

printf("Cleanup function called.\n");

}

int main()

{

atexit(cleanup);

printf("Main function is runnig...\n");

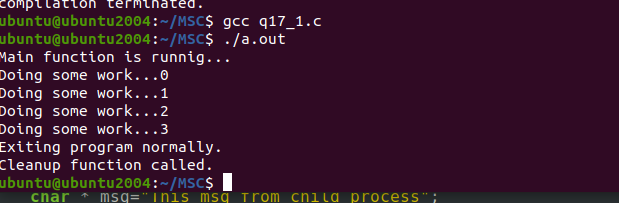
for(int i=0;i<=3;i++)

printf("Doing some work...%d\n",i);

printf("Exiting program normally.\n");

return 0;

}



//q18. Write a C program to demonstrates the different behaviour that can be seen with automatic, global, register, static and volatile variables (Use setjmp() and longjmp() system call).

//automatic, global, register, static an volatile variables using setjamp() and longjump() functions.

#include<stdio.h>

#include <setjmp.h>

jmp\_buf env;

int global\_var=10;

static int static\_var=20;

volatile int vol\_var=30;

void func()

{

auto int auto\_var=40;

register int reg\_var=50;

printf("In func:\n");

printf("Auto: %d, Register : %d, Static : %d , Volatile: %d, Global: %d\n",auto\_var,reg\_var,static\_var,vol\_var,global\_var);

longjmp(env,1);

}

int main()

{

if(setjmp(env)==0)

func();

else

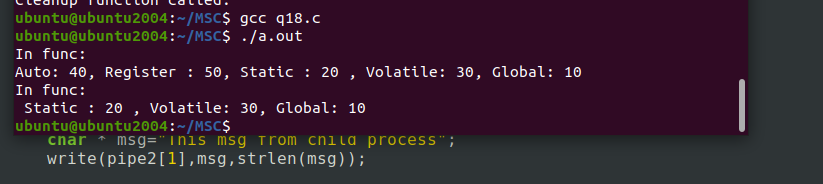
{

printf("In func:\n");

printf(" Static : %d , Volatile: %d, Global: %d\n",static\_var,vol\_var,global\_var);

}

}



//q19. Implement the following unix/linux command (use fork, pipe and exec system call) ls –l | wc –l

//implement ls -l|wc -l

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

int main()

{

int pipe1[2];

pid\_t pid1,pid2;

pipe(pipe1);

if((pid1=fork())==0)

{

dup2(pipe1[1],STDOUT\_FILENO);

close(pipe1[0]);

//close(pipe1[1]);

execlp("ls","ls","-l",NULL);

return 0;

}

if((pid2=fork())==0)

{

dup2(pipe1[0],STDIN\_FILENO);

close(pipe1[1]);

execlp("wc","wc","-l",NULL);

return 0;

}

close(pipe1[0]);

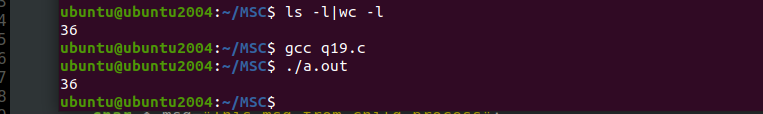
close(pipe1[1]);

wait(NULL);

wait(NULL);

return 0;

}



//q20. Write a C program to create „n‟ child processes. When all „n‟ child processes terminates, Display total cumulative time children spent in user and kernel mode

//n child process

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<sys/resource.h>

int main()

{

int n;

float total\_user\_time=0,total\_system\_time=0;

printf("Enter a number");

scanf("%d",&n);

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

if(fork()==0)

{ for(int j=0;j<1e7;j++);//busy

return 0;

}

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

wait(NULL);

struct rusage usage;

getrusage(RUSAGE\_CHILDREN, &usage);

total\_user\_time+=usage.ru\_utime.tv\_sec+usage.ru\_utime.tv\_usec/1e6;

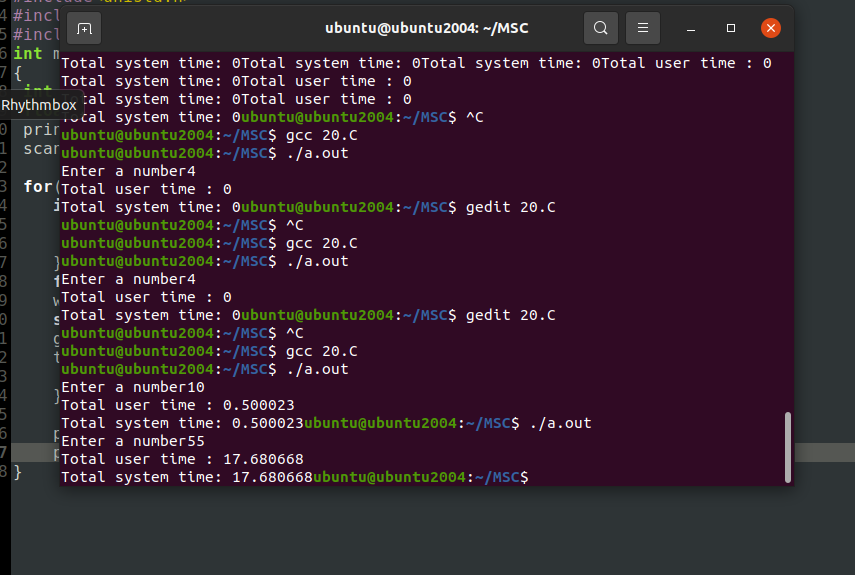
total\_system\_time+=usage.ru\_utime.tv\_sec+usage.ru\_utime.tv\_usec/1e6;

}

printf("Total user time : %f",total\_user\_time);

printf("\nTotal system time: %f",total\_system\_time);

}



//q21. Write a C program to create an unnamed pipe. The child process will write following three messages to pipe and parent process display it. Message1 = “Hello World” Message2 = “Hello SPPU” Message3 = “Linux is Funny”

#include<stdio.h>

#include<unistd.h>

int main()

{

int pip[2];

pid\_t pid;

char buffer[200];

pipe(pip);

pid=fork();

if (pid==0)

{

close(pip[0]);

write(pip[1],"Hello World \\n ",12);

write(pip[1],"Hello SPPU ",12);

write(pip[1],"Linux is funny ",12);

close(pip[1]);

}

else{

close(pip[1]);

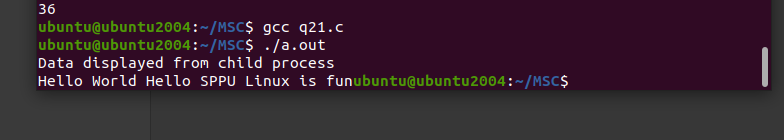
printf("Data displayed from child process\n");

while(read(pip[0],buffer,sizeof(buffer)>0))

printf("%s",buffer);

}

}



//q22. . Write a C program to get and set the resource limits such as files, memory associated with a processg

//q22 get and set resource limits such as files memory

#include<stdio.h>

#include<sys/resource.h>

#include<unistd.h>

int main()

{

struct rlimit rl;

getrlimit(RLIMIT\_NOFILE,&rl);

printf("Current max open files : soft limit =%lu, hard limit =%lu\n",rl.rlim\_cur,rl.rlim\_max);

rl.rlim\_cur=1024;

setrlimit(RLIMIT\_NOFILE,&rl);

printf("New max open files limit set to : %lu\n",rl.rlim\_cur);

getrlimit(RLIMIT\_AS,&rl);

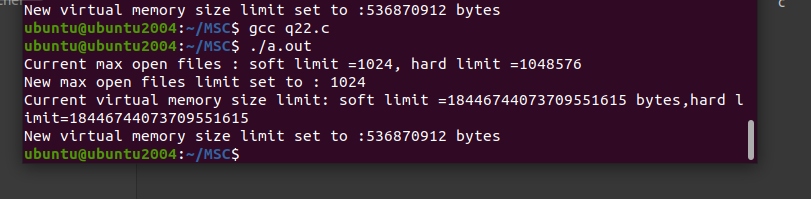
printf("Current virtual memory size limit: soft limit =%lu bytes,hard limit=%lu\n",rl.rlim\_cur,rl.rlim\_max);

rl.rlim\_cur=1024\*1024\*512;

setrlimit(RLIMIT\_AS,&rl);

printf("New virtual memory size limit set to :%lu bytes\n",rl.rlim\_cur);

}



//q23. Write a program that illustrates how to execute two commands concurrently with a pipe

//execute two commands concurrently with a pipe

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

int main()

{

int pipe1[2];

pid\_t pid1,pid2;

pipe(pipe1);

if((pid1=fork())==0)

{

dup2(pipe1[1],STDOUT\_FILENO);

close(pipe1[0]);

//close(pipe1[1]);

execlp("ls","ls","-l",NULL);

return 0;

}

if((pid2=fork())==0)

{

dup2(pipe1[0],STDIN\_FILENO);

close(pipe1[1]);

execlp("wc","wc","-l",NULL);

return 0;

}

close(pipe1[0]);

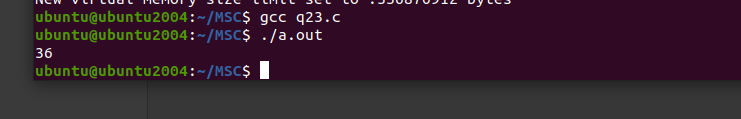
close(pipe1[1]);

wait(NULL);

wait(NULL);

return 0;

}



//q24. Write a C program that print the exit status of a terminated child processg

//To display the exit status of a terminated child process

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

int main()

{

pid\_t pid=fork();

if(pid==0)

{

printf("Child process is running...\n");

return 1;

}else

{

int status;

waitpid(pid,&status,0);

if(WIFEXITED(status))

printf("Child process exited staus : %d \n",WEXITSTATUS(status));

}

}



//q25. Write a C program that catches the ctrl-c (SIGINT) signal for the first time and display the appropriate message and exits on pressing ctrl-c again

//program to catch the ctrl-c(SIGINT) signall

#include<stdio.h>

#include<signal.h>

#include<unistd.h>

#include<stdlib.h>

int count=0;

void handle\_sigint(int sig)

{

count++;

if(count==1)

printf("Caught sigint! press Ctrl+c again to exit.\n");

else

{

printf("Exiting program...\n");

exit(0);

}

}

int main()

{

if(signal(SIGINT,handle\_sigint)==SIG\_ERR)

{

printf("Unable to register signal handler");

return 0;

}

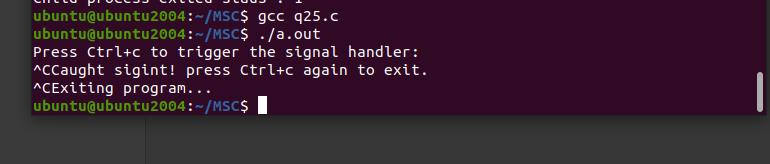
printf("Press Ctrl+c to trigger the signal handler: \n");

while(1)

pause();

return 0;

}



//q26. Write a C program which creates a child process and child process catches a signal SIGHUP, SIGINT and SIGQUIT. The Parent process send a SIGHUP or SIGINT signal after every 3 seconds, at the end of 15 second parent send SIGQUIT signal to child and child terminates by displaying message "My Papa has Killed me!!!”.

// program creates a child process and child process catches a signal SIGUP, SIGINT and SIQUITE.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<signal.h>

#include<sys/wait.h>

void handle\_sighup(int sig){ printf("Child received SIGHUP\n"); }

void handle\_sigint(int sig){ printf("Child received SIGINT\n"); }

void handle\_sigquit(int sig){ printf(" MY papa has killed me!!!"); exit(0); }

void main()

{

pid\_t pid=fork();

if(pid==0)

{

//set up signal handlers

signal(SIGHUP,handle\_sighup);

signal(SIGINT, handle\_sigint);

signal(SIGQUIT,handle\_sigquit);

while(1)

pause();

}else{//parent process

for(int i=1;i<=5;i++)

{

sleep(3);

if(i%2==1) kill(pid,SIGHUP);

else kill(pid,SIGINT);

}

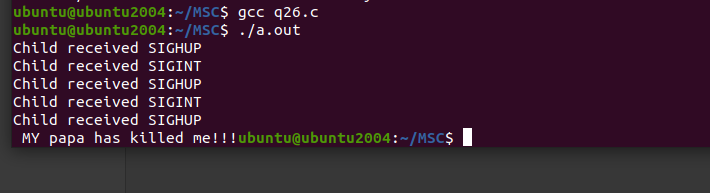
sleep(3);

kill(pid, SIGQUIT);//send sigquit to child

wait(NULL);

}

}



//q27. Write a C program to send SIGALRM signal by child process to parent process and parent process make a provision to catch the signal and display alarm is fired.(Use Kill, fork, signal and sleep system call)

//SIGALRM

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<signal.h>

void handle\_sigalrm(int sig){ printf("Alram is fired!\n"); exit(0); }

int main()

{

pid\_t pid= fork();//create a child process

if(pid==0)

{

sleep(2);

kill(getppid(),SIGALRM);

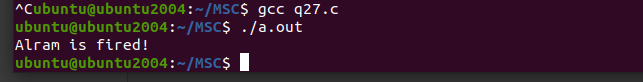
}else{

signal(SIGALRM,handle\_sigalrm);

while(1) pause();

}

}



//q28. Write a C program that illustrates suspending and resuming processes using signals.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <sys/types.h>

void handler(int signum) {

if (signum == SIGUSR1) {

printf("Child process received SIGUSR1: Suspending...\n");

raise(SIGSTOP); // Suspend the child process

} else if (signum == SIGCONT) {

printf("Child process resumed.\n");

}

}

int main() {

pid\_t pid = fork();

if (pid < 0) {

perror("Fork failed");

exit(1);

} else if (pid == 0) { // Child process

signal(SIGUSR1, handler); // Handle SIGUSR1

signal(SIGCONT, handler); // Handle SIGCONT

while (1) {

printf("Child process running...\n");

sleep(2); // Simulate work

}

} else { // Parent process

sleep(5); // Let the child run for a while

printf("Parent sending SIGUSR1 to suspend child.\n");

kill(pid, SIGUSR1); // Suspend child

sleep(5); // Wait before resuming

printf("Parent sending SIGCONT to resume child.\n");

kill(pid, SIGCONT); // Resume child

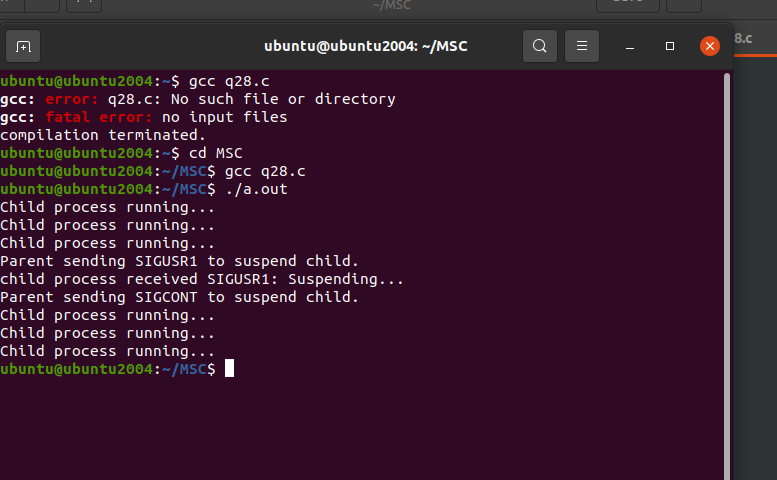
sleep(5); // Allow the child to run again

kill(pid, SIGKILL); // Terminate child

}

return 0;

}



//q29. 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 after every 3 seconds, at the end of 30 second parent send sigquit signal to child and child terminates my displaying message “My DADDY has Killed me!!!”.

// program creates a child process and child process catches a signal SIGUP, SIGINT and SIQUITE.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<signal.h>

#include<sys/wait.h>

void handle\_sighup(int sig){ printf("Child received SIGHUP\n"); }

void handle\_sigint(int sig){ printf("Child received SIGINT\n"); }

void handle\_sigquit(int sig){ printf(" MY DADDY has killed me!!!"); exit(0); }

void main()

{

pid\_t pid=fork();

if(pid==0)

{

//set up signal handlers

signal(SIGHUP,handle\_sighup);

signal(SIGINT, handle\_sigint);

signal(SIGQUIT,handle\_sigquit);

while(1)

pause();

}else{//parent process

for(int i=1;i<=10;i++)

{

sleep(3);

if(i%2==1) kill(pid,SIGHUP);

else kill(pid,SIGINT);

}

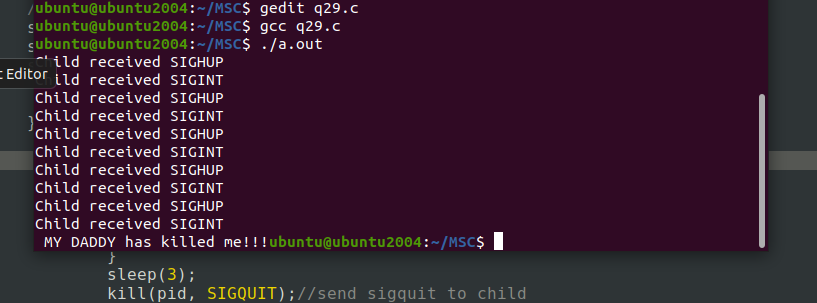
sleep(3);

kill(pid, SIGQUIT);//send sigquit to child

wait(NULL);

}

}



//q30. Write a C program to implement the following unix/linux command (use fork, pipe SPPU M.Sc. Computer Science Syllabus 2023-24 19 and exec system call). Your program should block the signal Ctrl-C and Ctrl-\ signal during the execution. i. Ls –l | wc –l

//implement ls -l|wc -l with blocking signals ctrl+c and ctrl+\

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<signal.h>

void block\_signals(){

sigset\_t mask;

sigemptyset(&mask);

sigaddset(&mask,SIGINT);

sigaddset(&mask,SIGQUIT);

sigprocmask(SIG\_BLOCK,&mask,NULL);

}

int main()

{

int pipe1[2];

pid\_t pid1,pid2;

pipe(pipe1);

block\_signals();

if((pid1=fork())==0)

{

dup2(pipe1[1],STDOUT\_FILENO);

close(pipe1[0]);

//close(pipe1[1]);

execlp("ls","ls","-l",NULL);

return 0;

}

if((pid2=fork())==0)

{

dup2(pipe1[0],STDIN\_FILENO);

close(pipe1[1]);

execlp("wc","wc","-l",NULL);

return 0;

}

close(pipe1[0]);

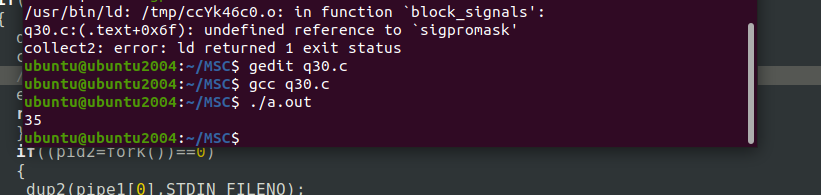
close(pipe1[1]);

wait(NULL);

wait(NULL);

return 0;

}



//q31. Write a C program which creates a child process to run linux/ unix command or any user defined program. The parent process set the signal handler for death of child signal and Alarm signal. If a child process does not complete its execution in 5 second then parent process kills child process

#include<stdio.h>

#include<stdlib.h>

#include<signal.h>

#include<signal.h>

#include<sys/wait.h>

#include<unistd.h>

pid\_t cpid;

void handle\_child(int sig){

int status;

waitpid(cpid,&status,0);

printf("Child proces terminated. status: %d\n",status);

exit(0);

}

void handle\_alarm(int sig)

{

printf("Child process taking too long. Terminating ...\n");

kill(cpid,SIGKILL);

}

int main()

{

cpid=fork();

if(cpid==0)

execlp("sleep","sleep","3",NULL);

else{

signal(SIGCHLD,handle\_child);

signal(SIGALRM,handle\_alarm);

alarm(5);

pause();

}

}

