**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S*.*NO*.*** | **PRACTICAL** | **DATE** | **SIGNATURE** |
| 1 | Write a program to display a file page wise assuming a page has 10 lines and each line has 80 characters |  |  |
| 2. | . Write a Program which converts all the small case letters in a file into appropriate capital letters. |  |  |
| 3. | write a program to print the details of the system (use uname sys call) |  |  |
| 4. | write a program which will print the list of environment variable and also print the value of the PATH system variable |  |  |
| 5. | Write a program to print current (soft) limit and maximum (Hard) limits of all resources |  |  |
| 6. | Write a program with an exit handler that outputs CPU usage |  |  |
| 7. | Write a program that prints it’s & it’s parent’s process ID. |  |  |
| 8. | Write a program that prints out various user & group ID’s. |  |  |
| 9. | Write a program which uses fork to create a child process& then parent & child print their respective process ID’s |  |  |
| 10. | Write a program that creates a chain of n processes, where n is a command line argument. |  |  |
| 11. | Write a program that creates a fan of n processes where n is passed as a command line argument. |  |  |
| 12. | Write a program to show that same opened file can be shared by both parent and child processes |  |  |
| 13. | Write a program that creates a child process to run ls – l |  |  |
| 14. | write a program to create a zombie child and find its status using system(ps) command |  |  |
| 15. | Write a program to copy a file. |  |  |
| 16. | Write a program for which output is automatically directed to a named file rather than on to the console |  |  |
| 17. | Write a program that redirects standards output to the file my.file (or Write a program that do the following operation cat XYZ > myfile).{ This question is similar to the previous question with the difference that here we will be using dup2 rather than dup } |  |  |
| 18. | write a program to create a empty directory using system calls |  |  |
| 19. | Write a program to remove a directory using system call |  |  |
| 20. | Write a program to output current working directory |  |  |
| 21. | Write a program to list files in a directory. |  |  |
| 22. | Write a program that returns true if a given file is a directory & false otherwise |  |  |
| 23. | Write a program that can display the type of a given file like regular, directory etc |  |  |
| 24. | Write a program to display the permission of a given file |  |  |
| 25. | Write a program to execute the equivalent of ls –l | sort –n +4 |  |  |
| 26. | Write a program to handle SIGUSR1 and SIGUSR2 signal |  |  |
| 27. | Write a program which suspends itself till it receives a SIGALARM signal |  |  |
| 28. | Write a program which prints the seconds part of current time whenever the SIGALRM signal is received by the program. |  |  |
| 29. | Write a Program to print the entries of passwd file for a given user name or user ID |  |  |
| 30. | Program to print all the information of file /etc/group for a given group name or group ID |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Q1. Write a program to display a file page wise assuming a page has 10 lines and each line has 80 characters

// To display a file page wise

#include<ctype.h>

#include<stdio.h>

#include<stdlib.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<error.h>

#include<ctype.h>

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

{

char buff[10][80];

int i,j;

char k;

FILE \*fp;

if(argc!=2)

{

fprintf(stderr,"Usage: ./a.out file name\n");

exit(1);

}

fp=fopen(argv[1],"r");

while(!feof(fp)){

for(i=0;i<10;i++)

for(j=0;j<80;j++)

buff[i][j]='\0';

for(i=0;i<10;i++)

fgets(buff[i],80,fp);

for(i=0;i<10;i++)

printf("%s",buff[i]);

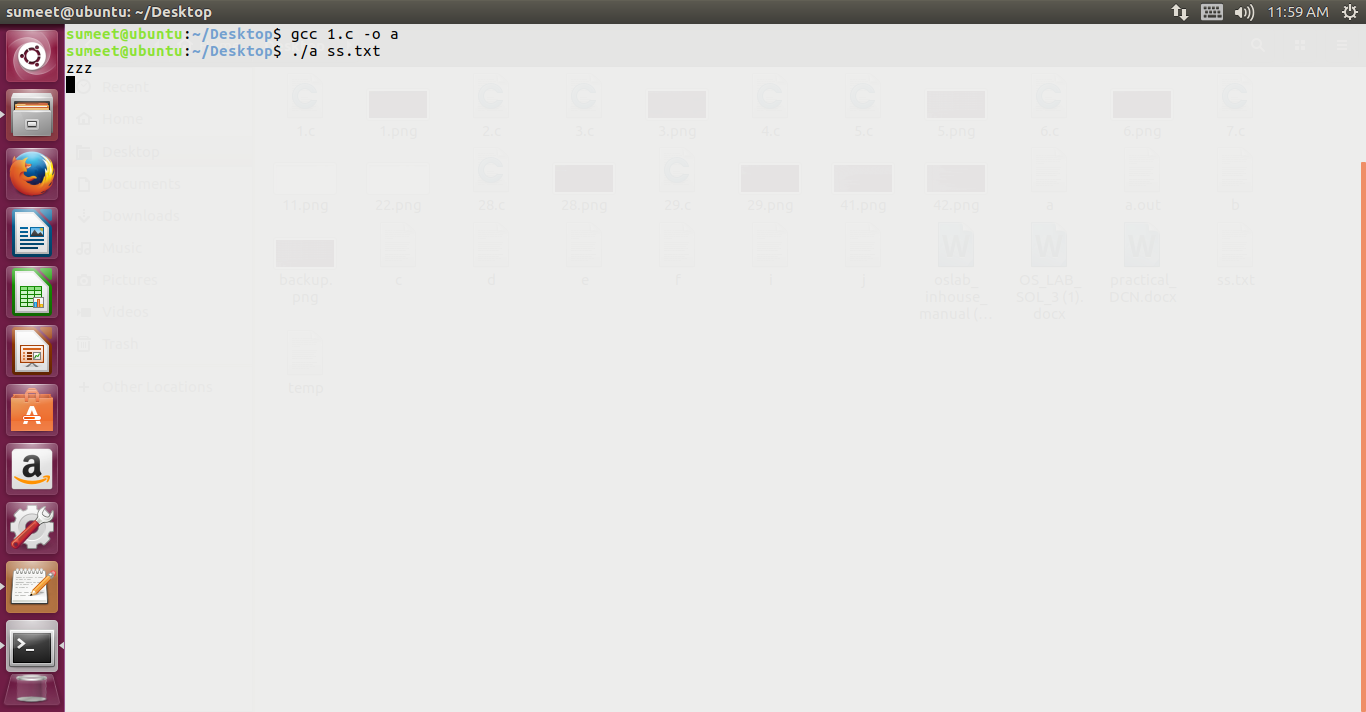
scanf("%c", &k);

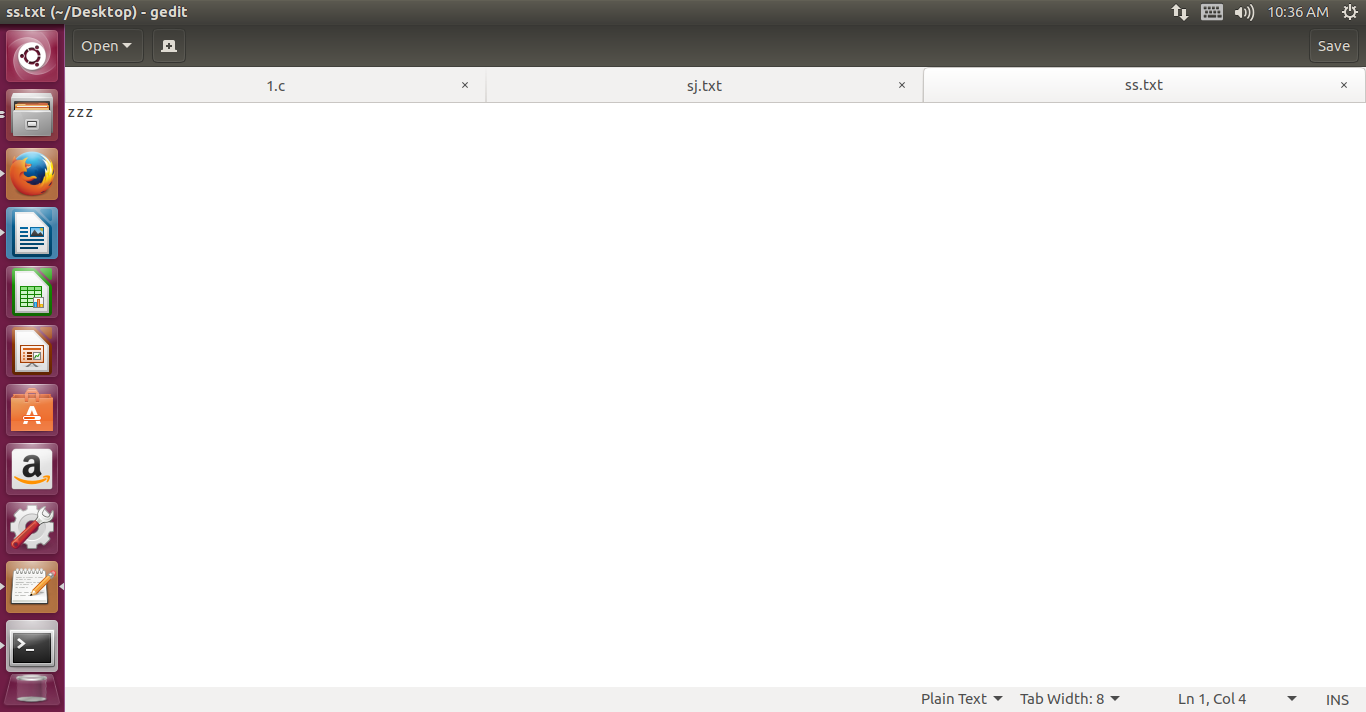
}

fclose(fp);

}

Output:



Q2. Write a Program which converts all the small case letters in a file into appropriate capital letters.

#include<ctype.h>

#include<stdio.h>

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<signal.h>

#include<time.h>

#include<error.h>

#include<ctype.h>

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

{

FILE \*fp, \*ft;

char ch;

if(argc!=2)

{

fprintf(stderr,"Usage: ./a.out file name\n");

exit(1);

}

fp=fopen(argv[1],"r");

if(fp==NULL)

{

printf("can't open file");

exit(1);

}

ft=fopen("temp","w");

while(!feof(fp))

{

ch=fgetc(fp);

if(ch>=97&&ch<=122)

ch=ch+'A' - 'a';

fputc(ch,ft);

}

fclose(ft);

fclose(fp);

ft=fopen("temp","r");

fp=fopen(argv[1],"w");

if(ft!=NULL)

{

while(!feof(ft))

{

ch=fgetc(ft);

fputc(ch,fp);

}

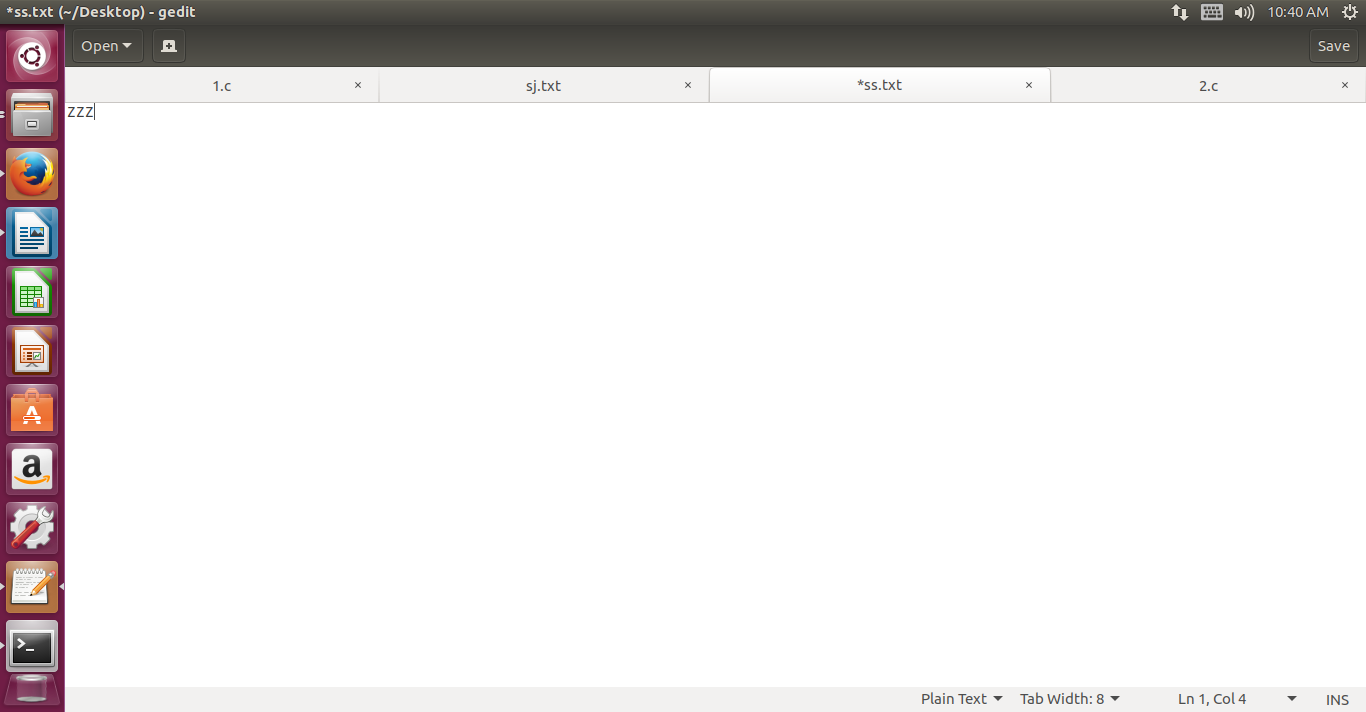
}

else

printf("Error in opening file");

}

Output:

Q3. write a program to print the details of the system (use uname sys call)

// compare your result with uname command

// do man 2 uname to understand the uname() function and structure //utsname

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<signal.h>

#include<time.h>

#include<sys/utsname.h>

main()

{

struct utsname u;

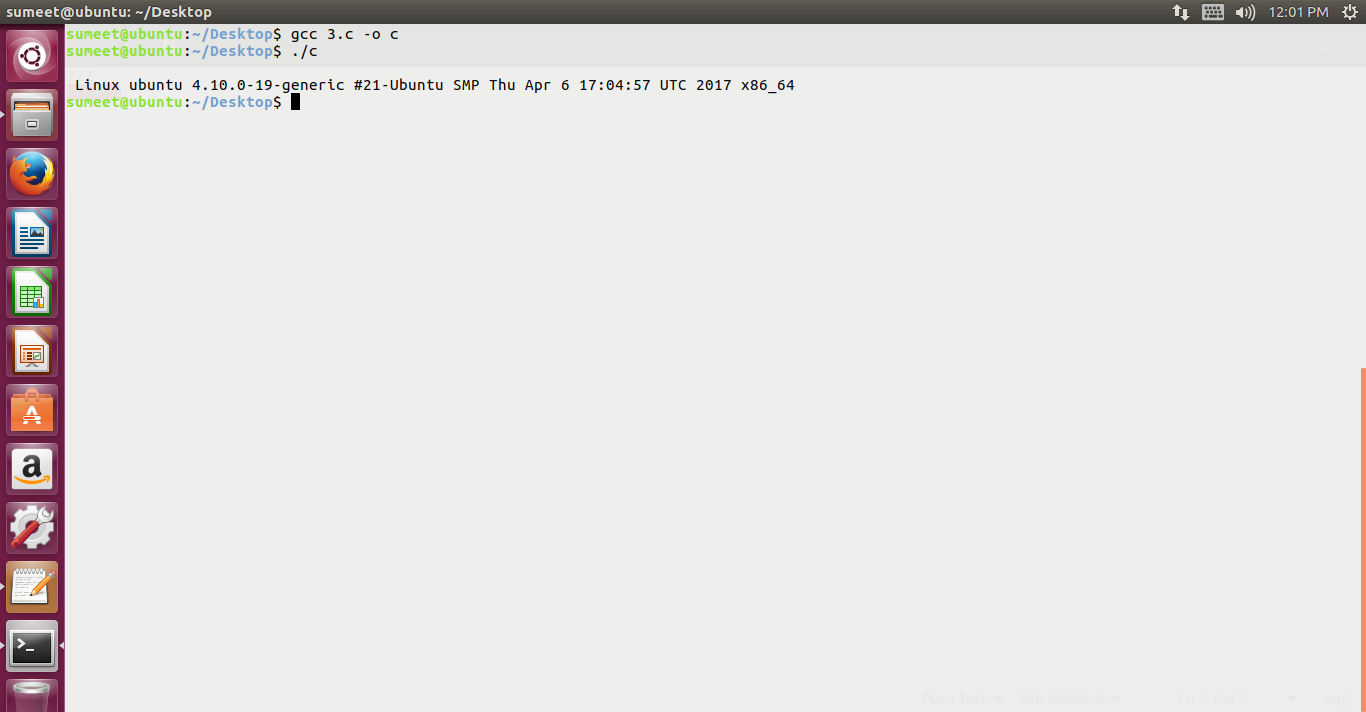
if(uname(&u)!=0)

fprintf(stderr, "Uname Error");

printf("\n %s %s %s %s %s\n",u.sysname,u.nodename,u.release,u.version,u.machine);

}

Output:

Q4. write a program which will print the list of environment variable and also print the value of the PATH system variable

//To print all environment variables

#include<stdio.h>

#include<stdlib.h>

extern char \*\*environ;// the external variable environ points to the //process environment list when the process begins executing.

// do man environ

int main(void)

{

int i;

char \*path;

printf("The environment list follows: \n");

for(i=0;environ[i] != NULL; i++)

printf("environ[%d]: %s\n", i, environ[i]);

if ((path =getenv("PATH")) == NULL)// do man getenv

printf("PATH environment variable not set\n");

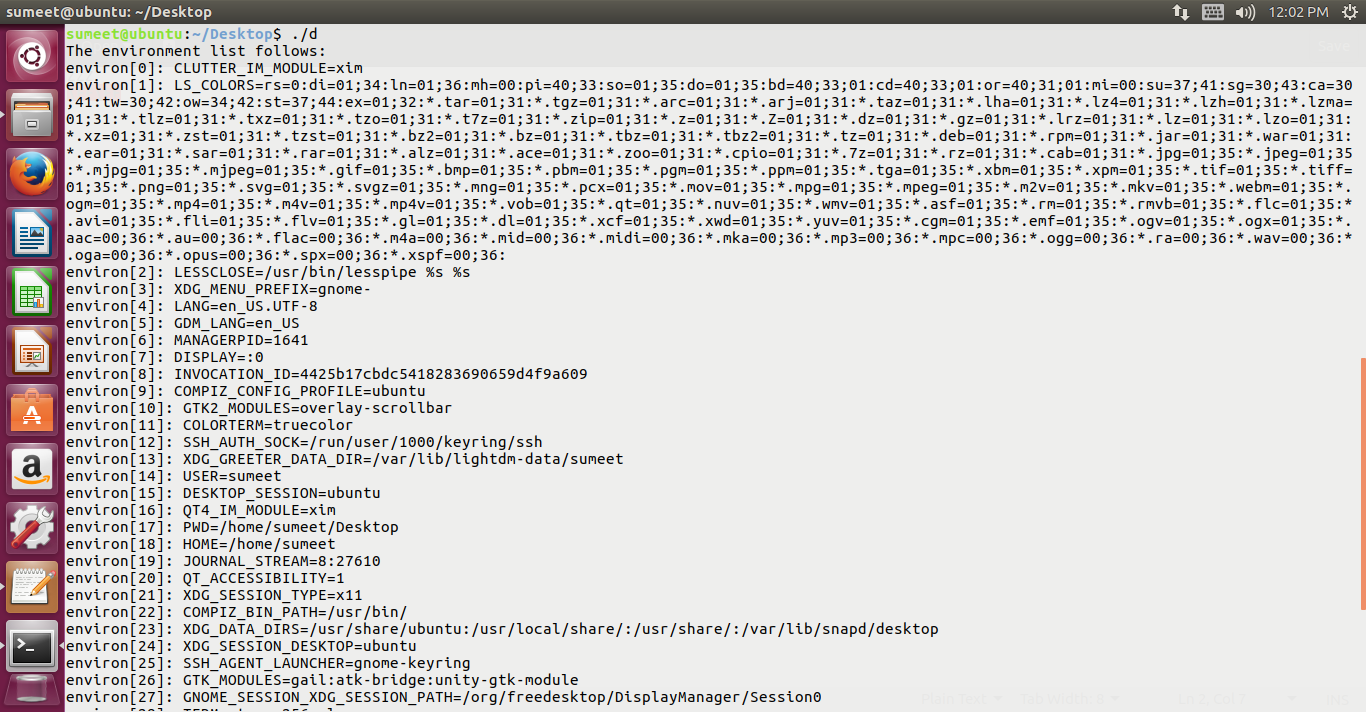
else

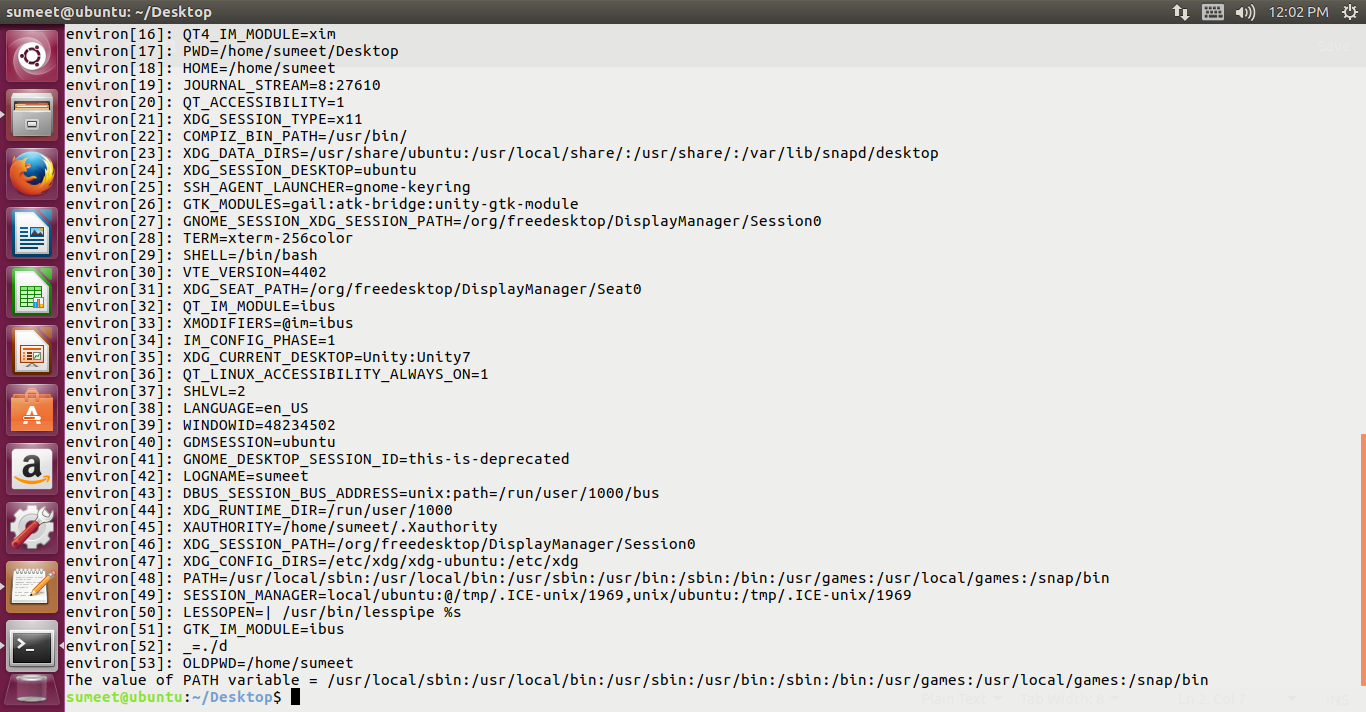
printf("The value of PATH variable = %s\n", path);

return 0;

}

Output:



Q5. Write a program to print current (soft) limit and maximum (Hard) limits of all resources

// To print current and max limits of all resources // pg 180 WR stevens

// do man getrlimit for more details

// -1 may mean no limit set for the resource ie the limit is infinity

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/resource.h>

#include<sys/time.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<limits.h>

#include<unistd.h>

main()

{

struct rlimit rl;

int i;

printf("\n Resources Name \t Current Limit \tMax Limit \t");

for(i=0;i<=10;i++)

{

if(getrlimit(i, &rl)<0)

{

printf("Error in grelimit\n");

exit(1);

}

switch(i)

{

case RLIMIT\_CPU :

printf("\nRLIMIT\_CPU\t%d\t\t%d",rl.rlim\_cur,rl.rlim\_max);

break;

case RLIMIT\_DATA:

printf("\nRLIMIT\_DATA\t%d\t\t%d",rl.rlim\_cur,rl.rlim\_max);

break;

case RLIMIT\_FSIZE:

printf("\nRLIMIT\_FSIZE\t%d\t\t%d",rl.rlim\_cur,rl.rlim\_max);

break;

case RLIMIT\_MEMLOCK:

printf("\nRLIMIT\_MEMLOCK\t%d\t\t%d",rl.rlim\_cur,rl.rlim\_max);

break;

case RLIMIT\_NOFILE:

printf("\nRLIMIT\_NOFILE\t%d\t\t%d",rl.rlim\_cur,rl.rlim\_max);

break;

case RLIMIT\_NPROC:

printf("\nRLIMIT\_NPROC\t%d\t\t%d",rl.rlim\_cur,rl.rlim\_max);

break;

/\*case RLIMIT\_OFILE:

printf(Â"\nRLIMIT\_OFILE\t%d\t\t%dÂ",rl.rlim\_cur,rl.rlim\_max);

break;\*/

case RLIMIT\_RSS:

printf("\nRLIMIT\_RSS\t%d\t\t%d",rl.rlim\_cur,rl.rlim\_max);

break;

case RLIMIT\_STACK:

printf("\nRLIMIT\_STACK\t%d\t\t%d",rl.rlim\_cur,rl.rlim\_max);

break;

case RLIMIT\_LOCKS:

printf("\nRLIMIT\_LOCK\t%d\t\t%d",rl.rlim\_cur,rl.rlim\_max);

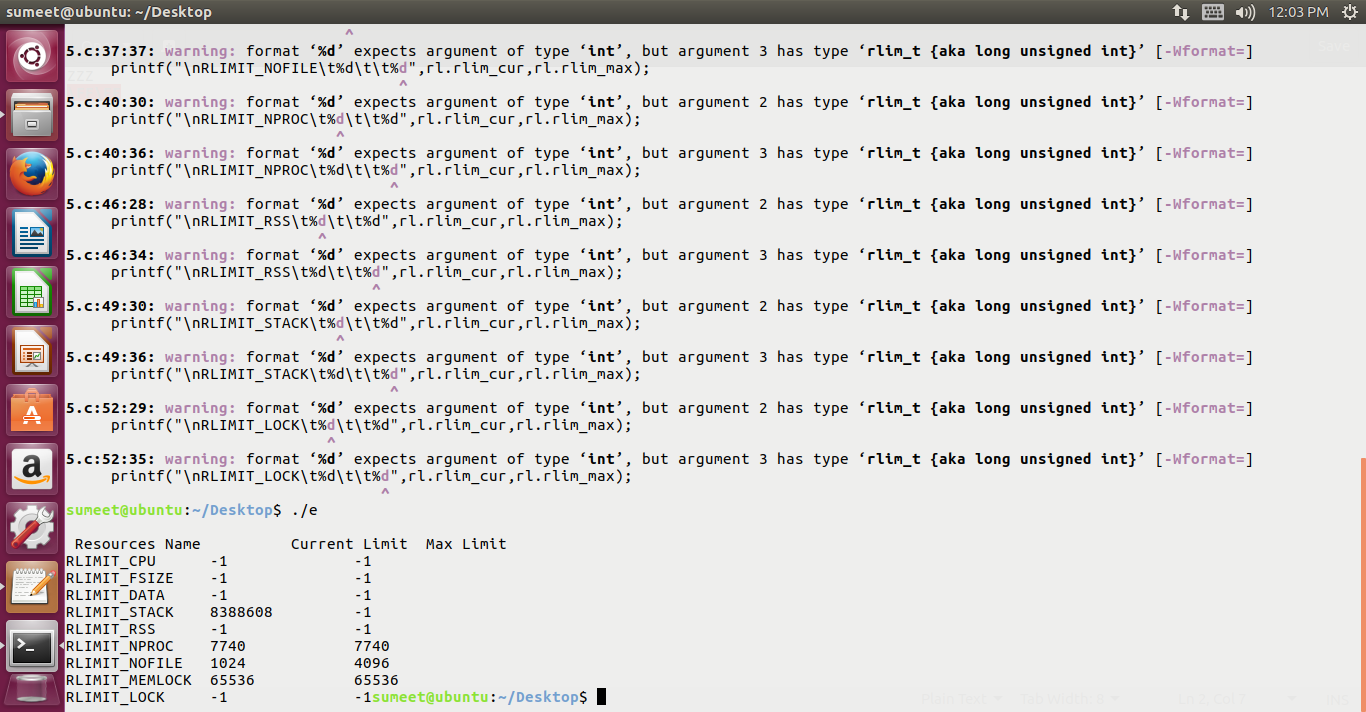
break;

}

}

}

Output:

Q6. Write a program with an exit handler that outputs CPU usage.

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/times.h>

#include <time.h> //modified

static void showtimes(void)

{

time\_t time1, time2;

time\_t time\_dif;

time1 = time(NULL); //man 2 time ; time(NULL) returns current time in

//seconds

printf("time1 : %ld",time1);

sleep(5); // man 3 sleep

time2 = time(NULL);

printf("time2 : %ld",time2);

time\_dif = difftime(time2,time1); // man difftime

printf("the showtime slept for: %ld seconds\n",time\_dif);

}

int main(void)

{

if (atexit(showtimes)) // man atexit

{

fprintf(stderr, "Failed to install showtimes exit handler\n");

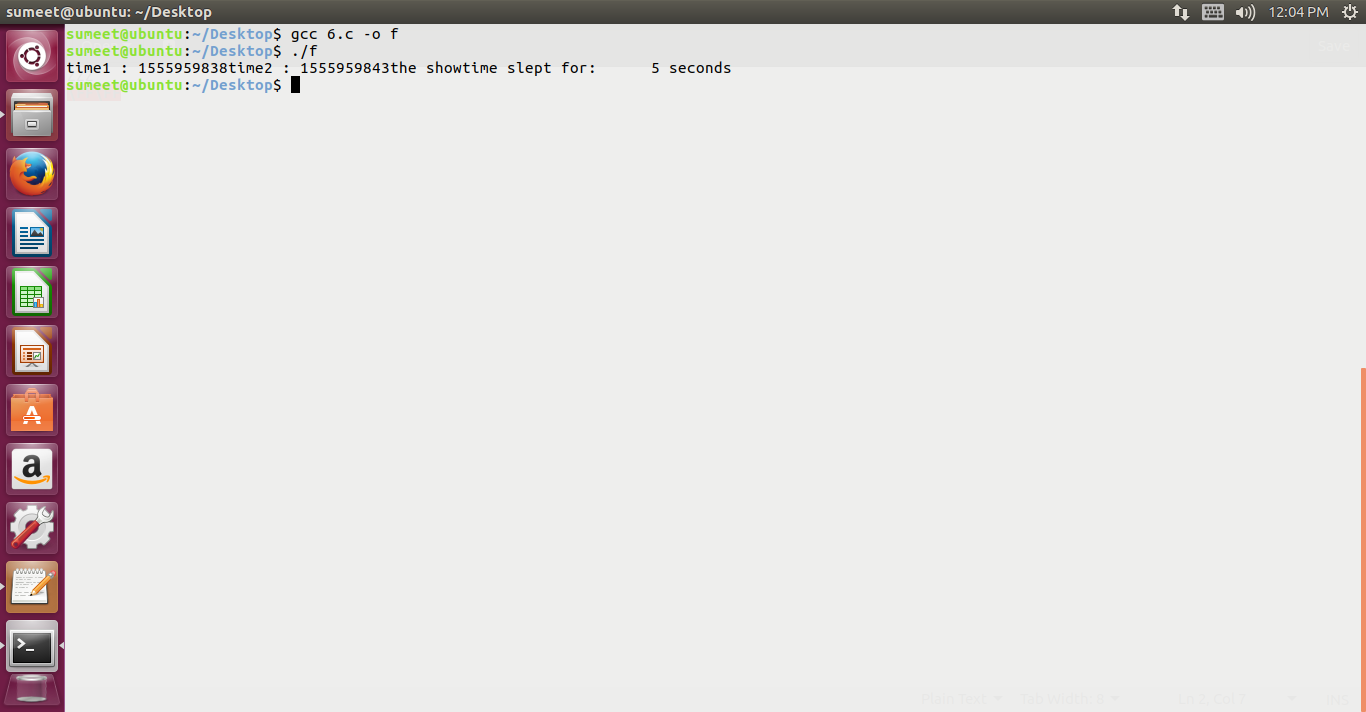
return 1;

}

/\* rest of main program goes here \*/

return 0;

}

Output: 

Q7. Write a program that prints it’s & it’s parent’s process ID.

#include <unistd.h>

#include<stdio.h>

int main (void)

{

printf("I am process %ld\n", (long)getpid()); //man getpid

printf("My parent is %ld\n", (long)getppid());

return 0;

}

Output:



Q8. Write a program that prints out various user & group ID’s.

#include <stdio.h>

#include <unistd.h>

int main(void)

{ // man getuid, man getid

printf("My real user ID is %5ld\n", (long)getuid());

printf("My effective user ID is %5ld\n", (long)geteuid());

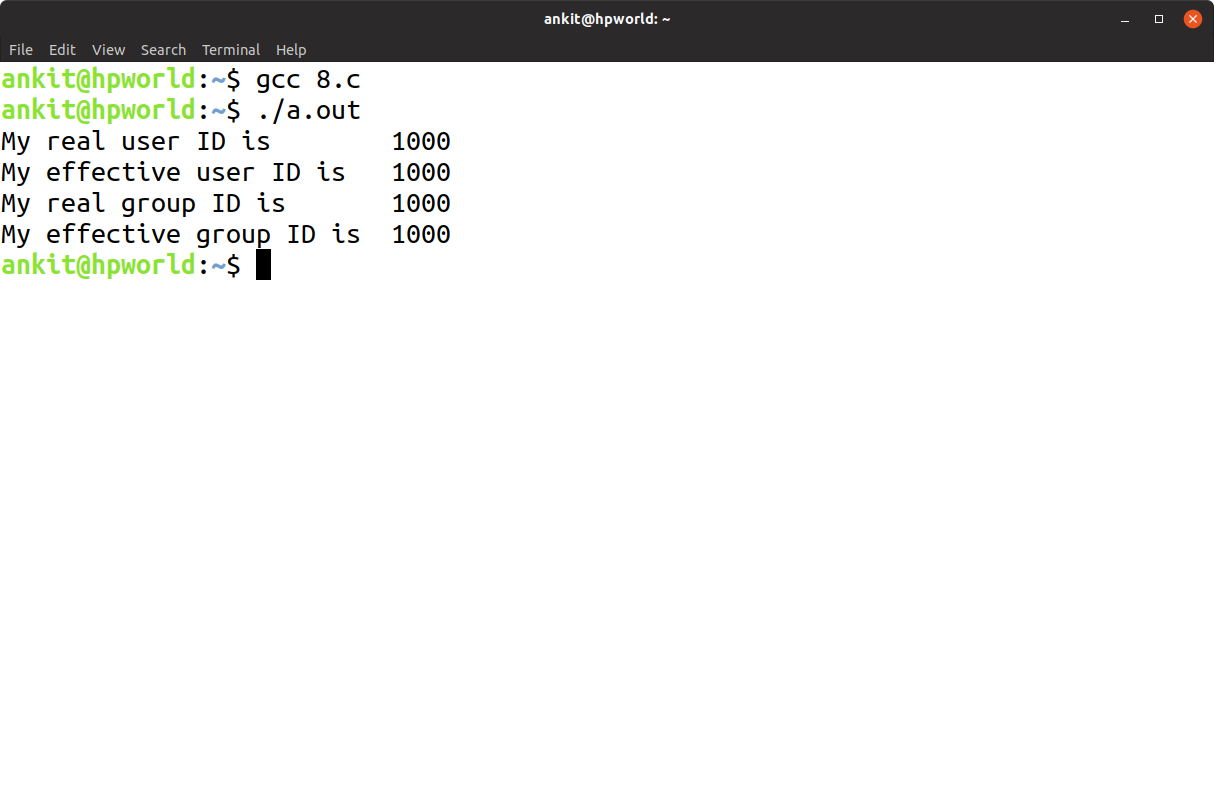
printf("My real group ID is %5ld\n", (long)getgid());

printf("My effective group ID is %5ld\n", (long)getegid());

return 0;

}

OUTPUT:



Q9. Write a program which uses fork to create a child process& then parent & child print their respective process ID’s

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main(void)

{

pid\_t childpid;

childpid = fork();

if (childpid == -1)

{

perror("Failed to fork");

return 1;

}

if (childpid == 0) /\* child code \*/

printf("I am child %ld\n", (long)getpid());

else /\* parent code \*/

printf("I am parent %ld\n", (long)getpid());

return 0;

}

OUTPUT:



Q10. Write a program that creates a chain of n processes, where n is a command line argument.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

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

{

pid\_t childpid = 0;

int i, n;

if (argc != 2) /\* check for valid number of command-line arguments \*/

{

fprintf(stderr, "Usage: %s processes\n", argv[0]);

return 1;

}

n = atoi(argv[1]);

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

if (childpid = fork()) // man fork

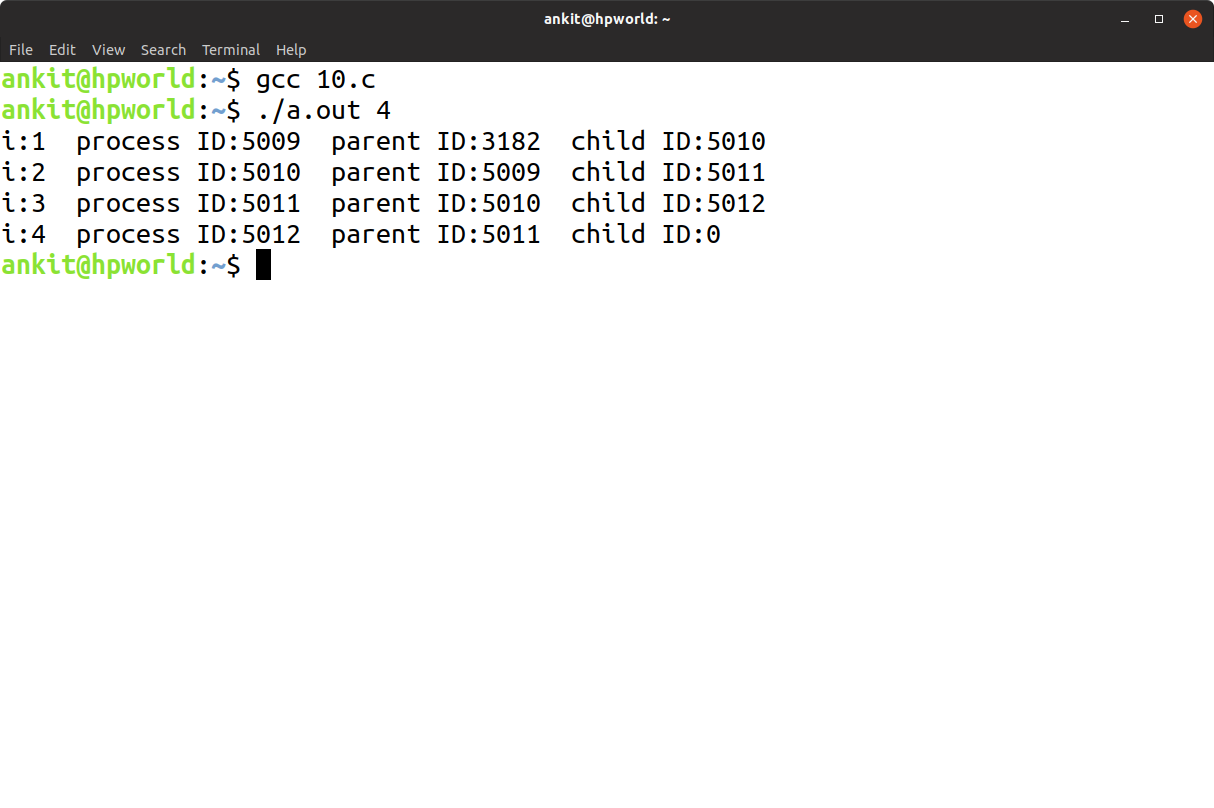
break;

fprintf(stderr, "i:%d process ID:%ld parent ID:%ld child ID:%ld\n",

i, (long)getpid(), (long)getppid(), (long)childpid);

return 0;

}

OUTPUT:

Q11. Write a program that creates a fan of n processes where n is passed as a command line argument.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

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

{

pid\_t childpid = 0;

int i, n;

if (argc != 2) /\* check for valid number of command-line arguments \*/

{

fprintf(stderr, "Usage: %s processes\n", argv[0]);

return 1;

}

n = atoi(argv[1]);

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

if ((childpid = fork()) <= 0)

break;

fprintf(stderr, "i:%d process ID:%ld parent ID:%ld child ID:%ld\n",

i, (long)getpid(), (long)getppid(), (long)childpid);

return 0;

}

OUTPUT:



Q12. Write a program to show that same opened file can be shared by both parent and child processes.

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include <unistd.h>

#include <sys/wait.h>

int main()

{

FILE \*fp;

int fd;

char ch;

fp=fopen("test","w");

fprintf(fp,"%s\n","This line is written by PARRENT PROCESS");

fflush(NULL);

fd=fork();

if(fd < 0)

{

printf("Fork Error");

exit(1);

}

if(fd == 0)

{

fprintf(fp,"%s","This line is written by CHILD PROCESS\n");

fclose(fp);

fp=fopen("test","r");

rewind(fp);

while(!feof(fp))

printf("%c",getc(fp));

}

if(fd > 0)

{ // man 2 wait

if (fd != wait(NULL)) /\* parent code \*/

{

perror("Parent failed to wait due to signal or error");

return 1;

}

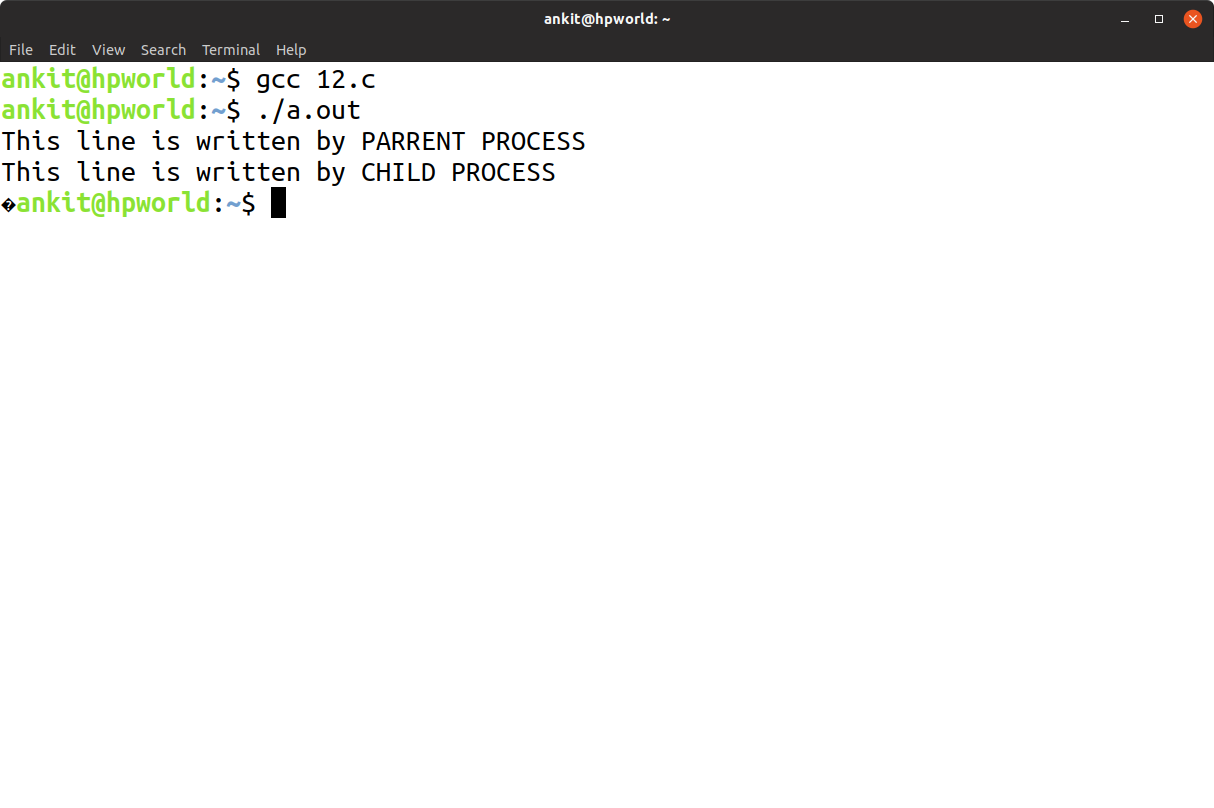
}

fclose(fp);

return 0;

}

OUTPUT:



Q13. Write a program that creates a child process to run ls – l

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main(void)

{

pid\_t childpid;

childpid = fork();

if (childpid == -1)

{

perror("Failed to fork");

return 1;

}

if (childpid == 0)

{ /\* child code \*/

execl("/bin/ls", "ls", "-l", NULL); // man 3 exec

perror("Child failed to exec ls");

return 1;

}

if (childpid != wait(NULL)) /\* parent code \*/

{

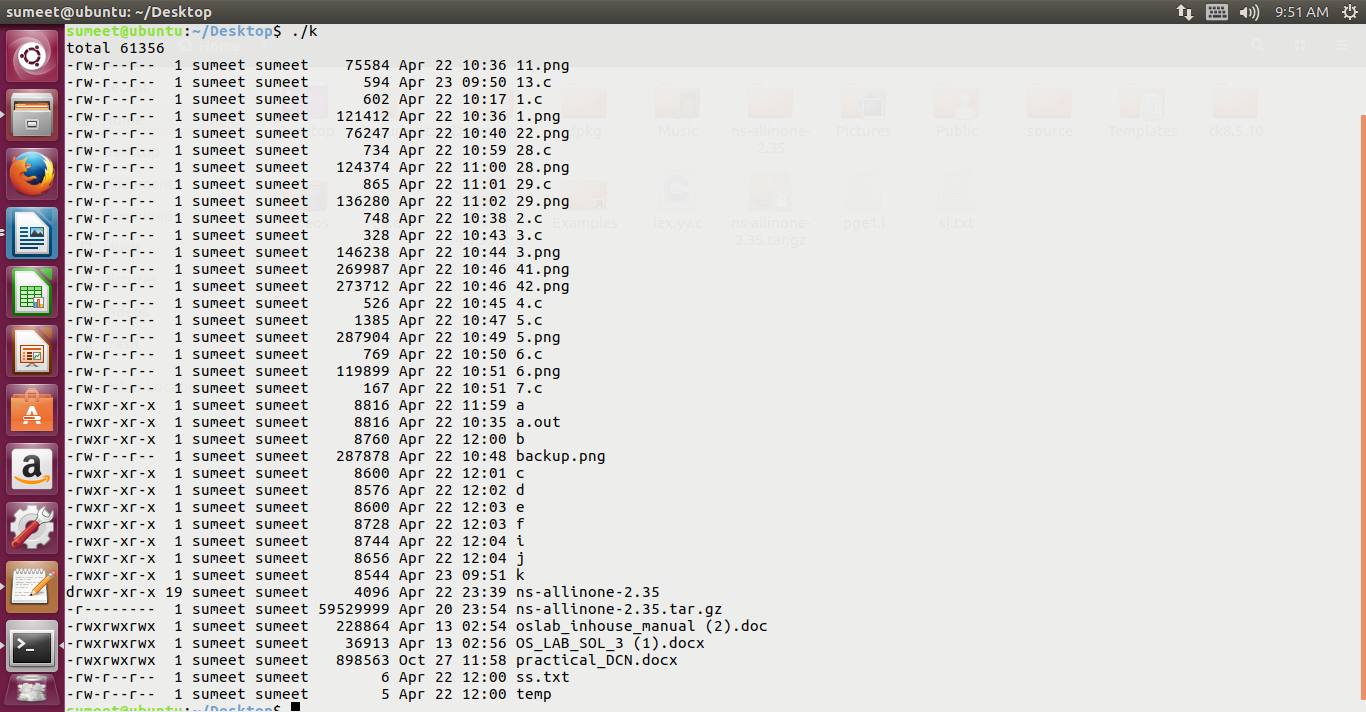
perror("Parent failed to wait due to signal or error");

return 1;

}

return 0;

}



Q14. write a program to create a zombie child and find its status using system(ps) command

//DEFUNCT MEANS ZOMBIE

// Zombie process is a process that has terminated, but whose parent has not yet waited for it. so parent's parent will become parent of this process

// also remember if parent process dies before child process then init process (process id = 1) becomes the parent of the executing child process

#include<stdio.h>

#include<stdlib.h>

#include <unistd.h>

#include <signal.h>

main()

{

int fd;

if((fd=fork())<0)

{

printf("error in creating child");

exit(1);

}

if(fd==0)

kill(getpid(),SIGKILL);

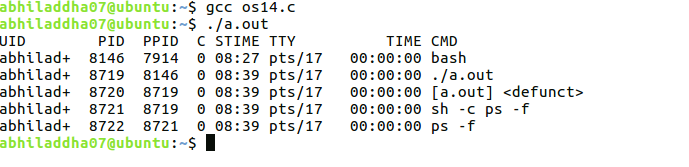
else

sleep(2);

system("ps -f");

}

Output:



Q15. Write a program to copy a file.

// the copyfile.c function copies a file fromfd to tofd

#include <errno.h>

#include <unistd.h>

#define BLKSIZE 1024

int copyfile(int fromfd, int tofd)

{

char \*bp;

char buf[BLKSIZE];

int bytesread;

int byteswritten = 0;

int totalbytes = 0;

for ( ; ; )

{

while (((bytesread = read(fromfd, buf, BLKSIZE)) == -1) &&

(errno == EINTR)) ; /\* handle interruption by signal \*/

if (bytesread <= 0) /\* real error or end-of-file on fromfd \*/

break;

bp = buf;

while (bytesread > 0)

{

while(((byteswritten = write(tofd, bp, bytesread)) == -1 ) &&

(errno == EINTR)) ; /\* handle interruption by signal \*/

if (byteswritten < 0) /\* real error on ––tofd \*/

break;

totalbytes += byteswritten;

bytesread -= byteswritten;

bp += byteswritten;

}

if (byteswritten == -1) /\* real error on tofd \*/

break;

}

return totalbytes;

}

//the main program to copy a file

#include <fcntl.h>

#include <stdio.h>

//#include "restart.h"

#define READ\_FLAGS O\_RDONLY

#define WRITE\_FLAGS (O\_WRONLY | O\_CREAT | O\_EXCL)

#define WRITE\_PERMS (S\_IRUSR | S\_IWUSR)

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

{

int bytes;

int fromfd, tofd;

if (argc != 3)

{

fprintf(stderr, "Usage: %s from\_file to\_file\n", argv[0]);

return 1;

}

if ((fromfd = open(argv[1], READ\_FLAGS)) == -1)

{

perror("Failed to open input file");

return 1;

}

if ((tofd = open(argv[2], WRITE\_FLAGS, WRITE\_PERMS)) == -1)

{

perror("Failed to create output file");

return 1;

}

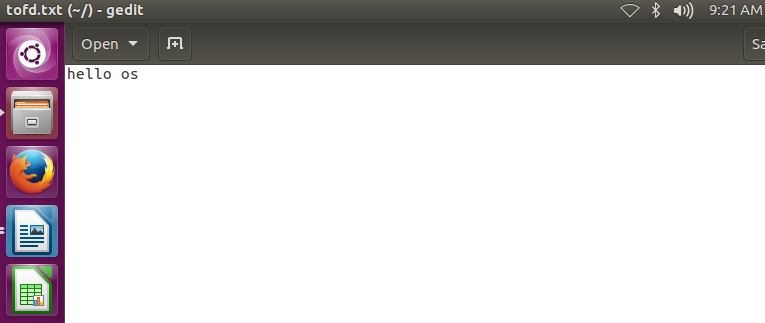
bytes = copyfile(fromfd, tofd);

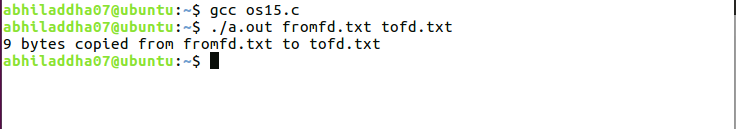
printf("%d bytes copied from %s to %s\n", bytes, argv[1], argv[2]);

return 0; /\* the return closes the files \*/

}

Output:





Q16. Write a program for which output is automatically directed to a named file rather than on to the console

/\*Program to create a file using dup fun (redirect output to some existing file.)\*/

#include<stdio.h>

#include<stdlib.h>

#include<fcntl.h>

int main()

{

int fd;

if((fd=open("test1",O\_WRONLY|O\_CREAT))<0)

{

printf("Error in opening file..\n");

exit(1);

}

close(1);

dup(fd);

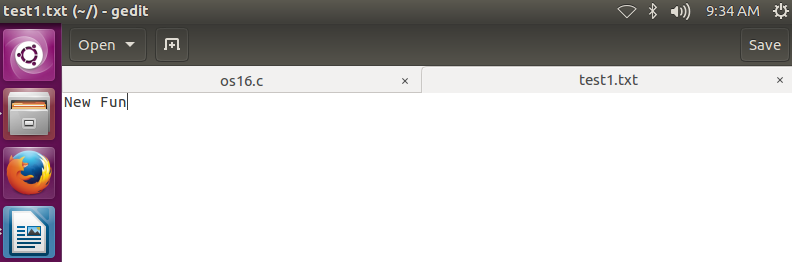
printf("New Fun");

close(fd);

return (0);

}

Output:



Q17. Write a program that redirects standards output to the file my.file (or Write a program that do the following operation cat XYZ > myfile).{ This question is similar to the previous question with the difference that here we will be using dup2 rather than dup }

#include <fcntl.h>

#include <stdio.h>

#include <sys/stat.h>

#include <unistd.h>

//#include "restart.h"

#define CREATE\_FLAGS (O\_WRONLY | O\_CREAT | O\_APPEND)

#define CREATE\_MODE (S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH)

int main(void)

{

int fd;

fd = open("my.file", CREATE\_FLAGS, CREATE\_MODE);

if (fd == -1)

{

perror("Failed to open my.file");

return 1;

}

if (dup2(fd, STDOUT\_FILENO) == -1)

{

perror("Failed to redirect standard output");

return 1;

}

if (close(fd) == -1)

{

perror("Failed to close the file");

return 1;

}

if (write(STDOUT\_FILENO, "OK", 2) == -1)

{

perror("Failed in writing to file");

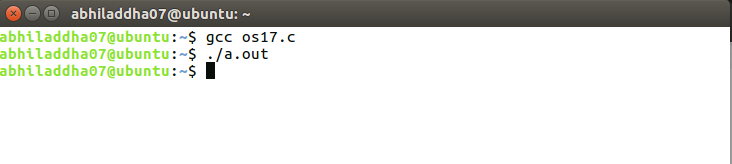
return 1;

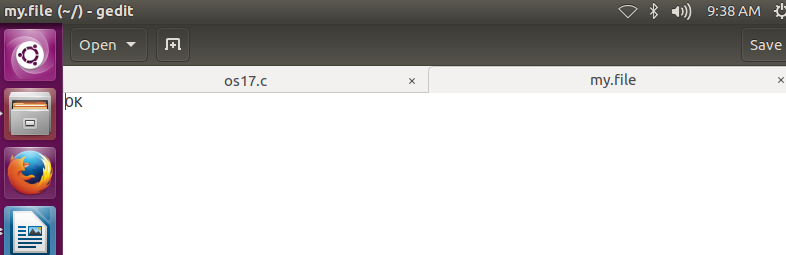
}

return 0;

}

Output:





Q18. write a program to create a empty directory using system calls

// Program to implement mkdir command using system calls

#include<stdio.h>

#include<stdlib.h>

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

{

if(argc!=2)

{

printf("Usages: ./a.out directory");

exit(1);

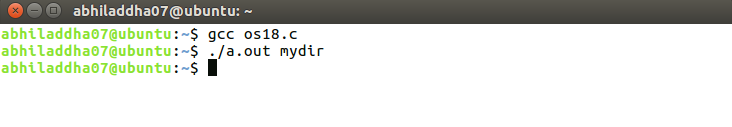
}

if(mkdir(argv[1],744)!=0)

printf("Error in Making Directory");

}

Output:



Q19. Write a program to remove a directory using system call

// Program to implement rmdir command using system calls

#include<stdio.h>

#include<stdlib.h>

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

{

if(argc!=2)

{

fprintf(stderr,"Too Less Arguments");

exit(1);

}

// remove() can be used to remove a name from the file system.so

//remove can be used to remove files and directories. for using

//remove()we need to include <stdio.h>.

// remove basically calls unlink() for files and rmdir() for

//directories

// we can also use unlink() or rmdir() for removing files or

//directories respectively directly inplace of remove(). but

//remember to include the header file <unistd.h> if you are using

//unlink() or rmdir()

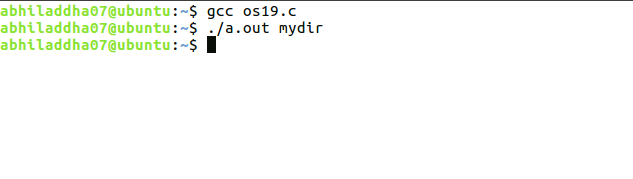
if(remove(argv[1])!=0)

fprintf(stderr,"Error in Removing Directory");

exit(1);

}

Output:



Q20. Write a program to output current working directory

#include <stdio.h>

#include <unistd.h>

#ifndef PATH\_MAX

#define PATH\_MAX 255

#endif

int main(void)

{

char mycwd[PATH\_MAX];

if (getcwd(mycwd, PATH\_MAX) == NULL)

{

perror("Failed to get current working directory");

return 1;

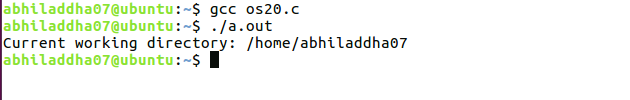
}

printf("Current working directory: %s\n", mycwd);

return 0;

}

Output:



Q21. Write a program to list files in a directory.

#include <dirent.h>

#include <errno.h>

#include <stdio.h>

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

{

struct dirent \*direntp;

DIR \*dirp;

if (argc != 2)

{

fprintf(stderr, "Usage: %s directory\_name\n", argv[0]);

return 1;

}

if ((dirp = opendir(argv[1])) == NULL)

{

perror ("Failed to open directory");

return 1;

}

while ((direntp = readdir(dirp)) != NULL)

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

while ((closedir(dirp) == -1) && (errno == EINTR)) ;

return 0;

}

Output:



Q22. Write a program that returns true if a given file is a directory & false otherwise.

#include <stdio.h>

#include <time.h>

#include <sys/stat.h>

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

{

struct stat statbuf;

if (stat(argv[1], &statbuf) == -1)

{

perror ("Failed to get status of file/directory");

return 1;

}

else

{

if (S\_ISDIR(statbuf.st\_mode))

printf("%s : is a directory\n",argv[1]);

else

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

}

return 0;

}

Output:



Q23. Write a program that can display the type of a given file like regular, directory etc

/\*Write C Program (Using only system calls)

1. That can display the type of a given file like regular, directory.\*/

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<fcntl.h>

#include<sys/stat.h>

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

{

struct stat statbuff;

int check;

if(argc!=2)

{

printf("Can accept only two arguments");

exit(1);

}

check=stat(argv[1], &statbuff);

if(check==0)

{

if(S\_ISREG(statbuff.st\_mode))

printf("Regular FIle\n");

else if(S\_ISDIR(statbuff.st\_mode))

printf("Directory\n");

else if(S\_ISCHR(statbuff.st\_mode))

printf("Char Device\n");

else

printf("Other File\n");

}

}

Output:



Q24. Write a program to display the permission of a given file

/\*Program that display file permission of a given file.\*/

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<fcntl.h>

#include<sys/stat.h>

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

{

struct stat statbuff;

int check;

if(argc!=2)

{

printf("Can Accept only two arguments");

exit(1);

}

check=stat(argv[1], &statbuff);

if(check==0)

{

//check Permission for Owner

if((statbuff.st\_mode & S\_IRUSR)==S\_IRUSR)

printf("Owner has Read Permission\n");

if((statbuff.st\_mode & S\_IWUSR)==S\_IWUSR)

printf("Owner has Write Permission\n");

if((statbuff.st\_mode & S\_IXUSR)==S\_IXUSR)

printf("Owner has Execute Permission\n");

// check Permission for Group

if((statbuff.st\_mode & S\_IRGRP)==S\_IRGRP)

printf("Group has Read Permission\n");

if((statbuff.st\_mode & S\_IWGRP)==S\_IWGRP)

printf("Group has Write Permission\n");

if((statbuff.st\_mode & S\_IXGRP)==S\_IXGRP)

printf("Group has Execute Permission\n");

// check Permission for Others

if((statbuff.st\_mode & S\_IROTH)==S\_IROTH)

printf("Others has Read Permission\n");

if((statbuff.st\_mode & S\_IWOTH)==S\_IWOTH)

printf("Others has Write Permission\n");

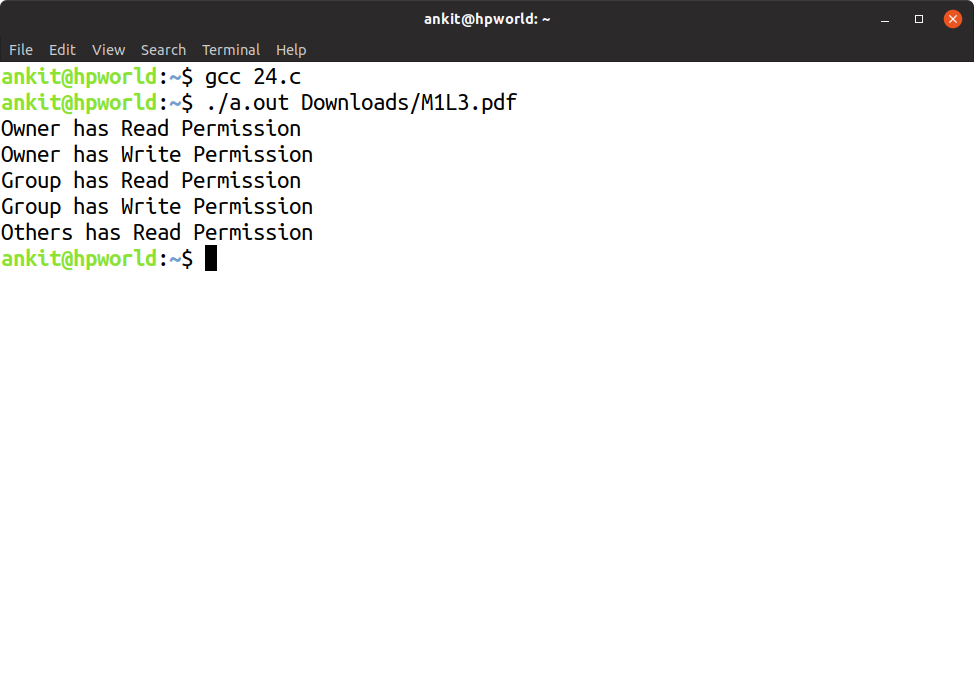
if((statbuff.st\_mode & S\_IXOTH)==S\_IXOTH)

printf("Others has Executed Permission\n");

}

}

Output:



Q25. Write a program to execute the equivalent of ls –l | sort –n +4

#include <errno.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main(void)

{

pid\_t childpid;

int fd[2];

if ((pipe(fd) == -1) || ((childpid = fork()) == -1))

{

perror("Failed to setup pipeline");

return 1;

}

if (childpid == 0) /\* ls is the child \*/

{

if (dup2(fd[1], STDOUT\_FILENO) == -1)

perror("Failed to redirect stdout of ls");

else if ((close(fd[0]) == -1) || (close(fd[1]) == -1))

perror("Failed to close extra pipe descriptors on ls");

else

{

execl("/bin/ls", "ls", "-l", NULL);

perror("Failed to exec ls");

}

return 1;

}

if (dup2(fd[0], STDIN\_FILENO) == -1) /\* sort is the parent \*/

perror("Failed to redirect stdin of sort");

else if ((close(fd[0]) == -1) || (close(fd[1]) == -1))

perror("Failed to close extra pipe file descriptors on sort");

else

{

execl("/bin/sort", "sort", "-n", "+4", NULL);

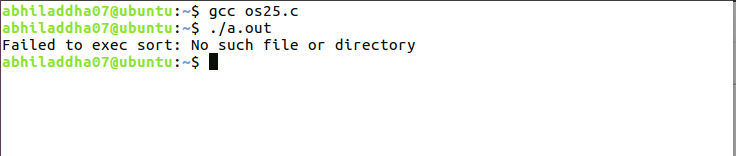
perror("Failed to exec sort");

}

return 1;

}

Output:



Q26. Write a program to handle SIGUSR1 and SIGUSR2 signal

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<signal.h>

#include<unistd.h>

void fun(int);

main()

{

char a[200];

if((signal(SIGUSR1,fun))==SIG\_ERR)

{

printf("Handler not registered\n");

exit(1);

}

if((signal(SIGUSR2,fun))==SIG\_ERR)

{

printf("Handler not registered\n");

exit(1);

}

while(1)

pause(); // include <unistd.h>

}

void fun(int i)

{

if(i==SIGUSR1)

{

printf("SIGUSR1 INTRRUPT");

fflush(NULL);

}

else if(i==SIGUSR2)

{

printf("SIGUSR2 INTRRUPT");

fflush(NULL);

}

//raise(SIGKILL);

}

Output:



Q27. Write a program which suspends itself till it receives a SIGALARM signal

//Program to write own sleep command using alarm and pause

// do man 2 alarm to know about the function alarm()

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<signal.h>

#include<unistd.h>

void sig\_alrm(int);

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

{

if((signal(SIGALRM, sig\_alrm))==SIG\_ERR)

printf("Not Registered");

alarm(5);

pause();

}

void sig\_alrm(int sig)

{

if(sig==SIGALRM)

printf("Wake Up");

}

Output:



Q28. Write a program which prints the seconds part of current time whenever the SIGALRM signal is received by the program.

// After two seconds print TM\_SEC fileld of tm structure using alarm

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<signal.h>

#include<time.h>

#include<unistd.h>

void sig\_hand(int);

main()

{

int i=1;

pid\_t pid;

if(signal(SIGALRM,sig\_hand)==SIG\_ERR)

{

printf("Not Registered");

}

while(i<=5)

{

i++;

pid=getpid();

sleep(2);

kill(pid,SIGALRM);

}

}

void sig\_hand(int sig)

{

struct tm \*t; // this structure defination can be got by man //localtime

time\_t tt; // used for time in seconds

if(sig==SIGALRM)

{

tt=time(NULL); //returns current time in sec since epoc (do man 2 time)

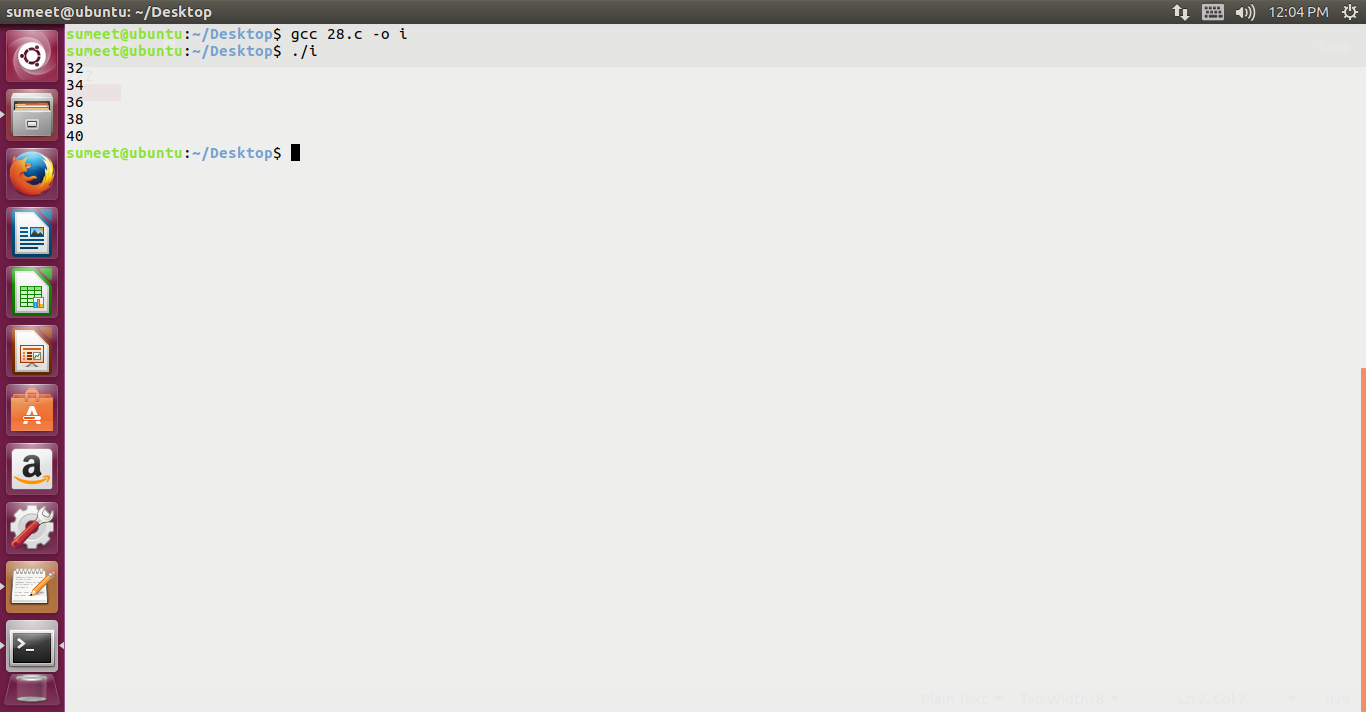
t=localtime(&tt); // break down time in hr, min, sec, etc (do man localtime)

printf("%d\n",t->tm\_sec);// new line is necessary here

}

}

Output:

Q29. Write a Program to print the entries of passwd file for a given user name or user ID

// do man getpwnam to understand getpwnam() function and passwd structure

// also do cat /etc/passwd file and check what all information is given in

// this file

#include<pwd.h>

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<signal.h>

#include<time.h>

#include<error.h>

#include<ctype.h>

main()

{

char u\_name[10];

char ch;

uid\_t u\_id;

struct passwd \*p;

printf("Enter Your Choice\n");

printf("Whether you want to enter UNAME or UID?(N or I)");

scanf("%c",&ch);

if((ch == 'N')|| (ch == 'n'))

{

printf("Enter UNAME");

scanf("%s",u\_name);

p=getpwnam(u\_name);

printf("\n%s\n %s\n %d\n %d\n %s\n %s\n %s\n", p->pw\_name, p->pw\_passwd, p->pw\_uid,p->pw\_gid,p->pw\_gecos, p->pw\_dir, p->pw\_shell);

}

else if((ch == 'I' || 'i'))

{

printf("Enter UID");

scanf("%d",&u\_id);

p= **getpwuid (u\_id);**

printf("\n%s %s %d %d %s %s %s\n", p->pw\_name, p->pw\_passwd, p->pw\_uid,p->pw\_gid,p->pw\_gecos, p->pw\_dir, p->pw\_shell);

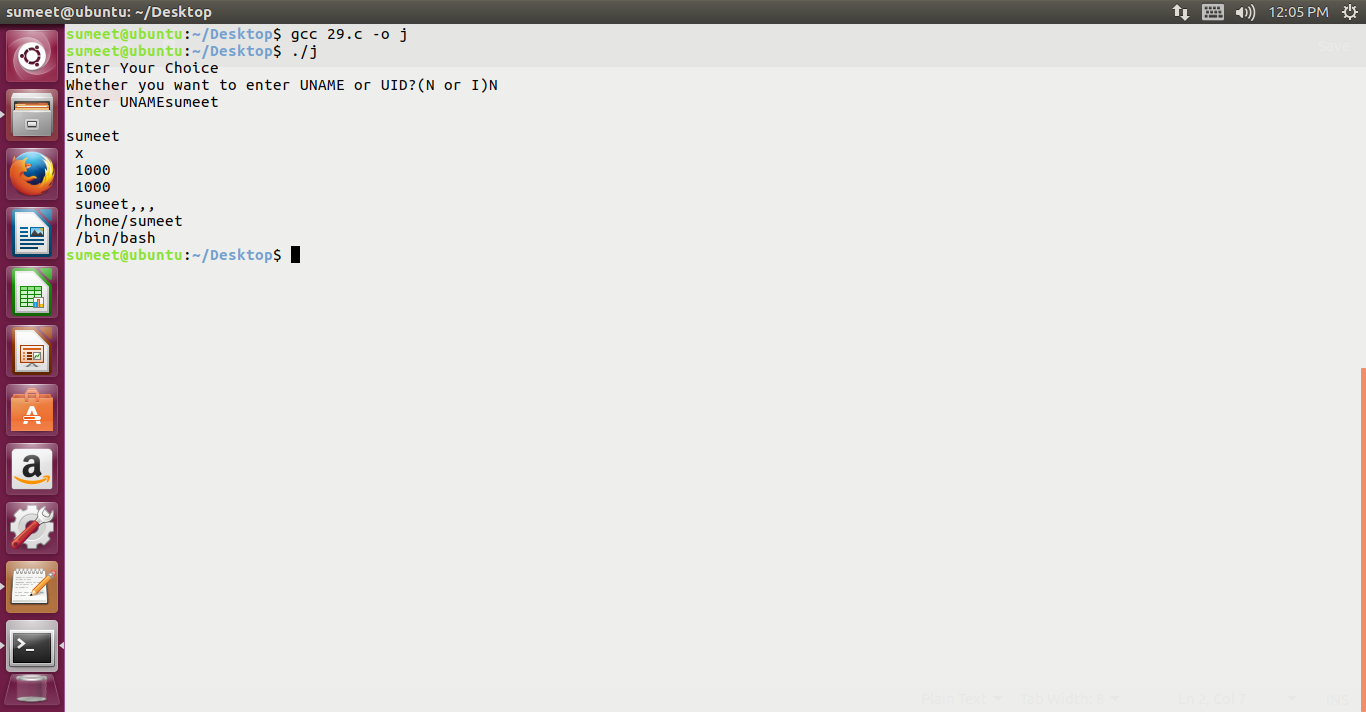
}

else

printf("Wrong Choice");

}

Output:

Q30. Program to print all the information of file /etc/group for a given group name or group ID

// do man getgrnam to get the information about getgrnam() function and

// structure group

// also do cat /etc/group and check what all information is written here

#include<grp.h>

#include<pwd.h>

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<signal.h>

#include<time.h>

#include<error.h>

#include<ctype.h>

main()

{

char g\_name[10];

gid\_t gid;

char ch;

struct group \*g;

printf("Enter Your Choice: \n Enter Group Name(N) \n Enter Group ID (I)\n");

printf("Enter Choice");

scanf("%c",&ch);

switch(ch)

{

case 'N':

case 'n':

printf("Enter GNAME:");

scanf("%s",g\_name);

g=getgrnam(g\_name);

printf("\n %s %s %d\n", g->gr\_name, g->gr\_passwd, g->gr\_gid);

break;

case 'I':

case 'i':

printf("Enter GID:");

scanf("%d",&gid);

g=getgrgid(gid);

printf("\n %s %s %d\n", g->gr\_name, g->gr\_passwd, g->gr\_gid);

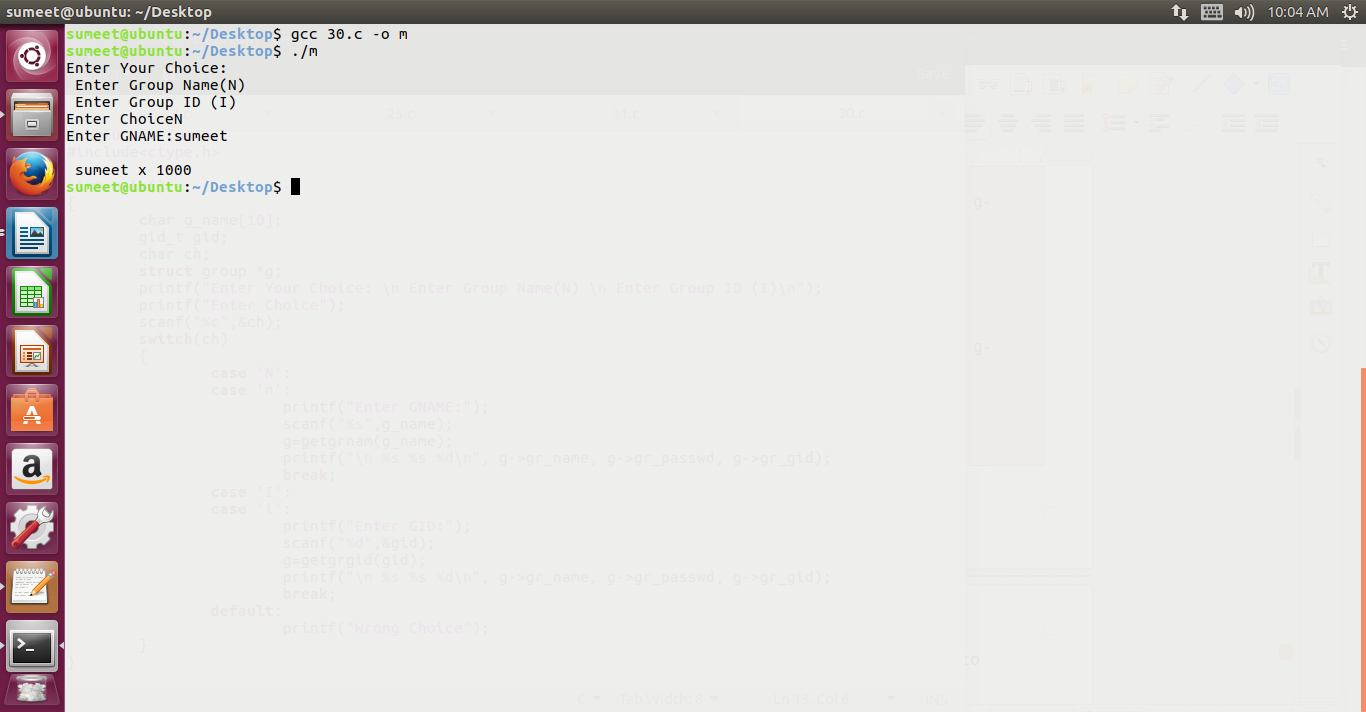
break;

default:

printf("Wrong Choice");

}

}



* Q31. Reads what is written to a named pipe & writes it to standard output.
* Q32. Write an informative message to a named pipe

// server

#include <errno.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/stat.h>

//#include "restart.h"

//#include <errno.h>

//#include <unistd.h>

#define BLKSIZE 1024

#define FIFOARG 1

#define FIFO\_PERMS (S\_IRWXU | S\_IWGRP| S\_IWOTH)

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

int requestfd;

if (argc != 2) { /\* name of server fifo is passed on the command line \*/

fprintf(stderr, "Usage: %s fifoname > logfile\n", argv[0]);

return 1;

}

/\* create a named pipe to handle incoming requests \*/

if ((mkfifo(argv[FIFOARG], FIFO\_PERMS) == -1) && (errno != EEXIST)) {

perror("Server failed to create a FIFO");

return 1;

}

/\* open a read/write communication endpoint to the pipe \*/

if ((requestfd = open(argv[FIFOARG], O\_RDWR)) == -1) {

perror("Server failed to open its FIFO");

return 1;

}

copyfile(requestfd, STDOUT\_FILENO);

return 1;

}

int copyfile(int fromfd, int tofd) {

char \*bp;

char buf[BLKSIZE];

int bytesread;

int byteswritten = 0;

int totalbytes = 0;

for ( ; ; ) {

while (((bytesread = read(fromfd, buf, BLKSIZE)) == -1) &&

(errno == EINTR)) ; /\* handle interruption by signal \*/

if (bytesread <= 0) /\* real error or end-of-file on fromfd \*/

break;

bp = buf;

while (bytesread > 0) {

while(((byteswritten = write(tofd, bp, bytesread)) == -1 ) &&

(errno == EINTR)) ; /\* handle interruption by signal \*/

if (byteswritten < 0) /\* real error on tofd \*/

break;

totalbytes += byteswritten;

bytesread -= byteswritten;

bp += byteswritten;

}

if (byteswritten == -1) /\* real error on tofd \*/

break;

}

return totalbytes;

}

// client

#include <errno.h>

#include <fcntl.h>

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <unistd.h>

#include <sys/stat.h>

//#include "restart.h"

#define FIFOARG 1

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

time\_t curtime;

int len;

char requestbuf[PIPE\_BUF];

int requestfd;

if (argc != 2) { /\* name of server fifo is passed on the command line \*/

fprintf(stderr, "Usage: %s fifoname\n", argv[0]);

return 1;

}

if ((requestfd = open(argv[FIFOARG], O\_WRONLY)) == -1) {

perror("Client failed to open log fifo for writing");

return 1;

}

curtime = time(NULL);

snprintf(requestbuf, PIPE\_BUF, "%d: %s", (int)getpid(), ctime(&curtime));

len = strlen(requestbuf);

if (write(requestfd, requestbuf, len) != len) {

perror("Client failed to write");

return 1;

}

close(requestfd);

return 0;

}