Department of Electrical & Computer Engineering
North South University

Project Title: Process Managements

Course: CSE 323 – Operating system Design

Section – 02

Fall 2022

Group- 10

CONTRIBUTION IN PROJECT

Name	ID	Contribution
Zobaer Ahammod Zamil	2021796042	Project merge, Show all running processes, Terminate processes, Show the files and their size of a directory, Change directory.
Md. Nazmus Saqib	2021696642	Create file, Delete File, Read File, Show current Path
Fazley Rabbi	2022242042	Create directory, Delete directory

Submitted To:

Tarek Ibne Mizan (TIM)

Department of Electrical & Computer Engineering,
North South University

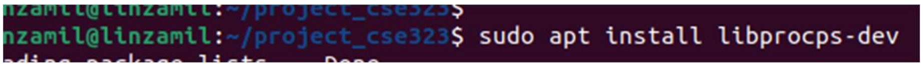
Process Managements

Objective:

- Check all running processes
- Create Child process
- Terminate any process if not needed
- Create and Delete Files / Directories
- Read files
- Show List of files and their size of current directory

Proposed Solutions:

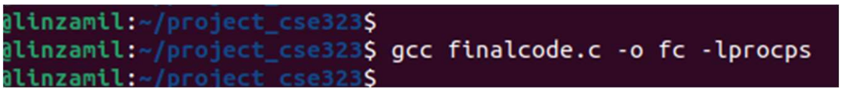
Firstly, install **“libprocps-dev”** by following this snapshot:



```
nzamil@linzamil:~/project_cse323$
nzamil@linzamil:~/project_cse323$ sudo apt install libprocps-dev
ading package lists... Done
```

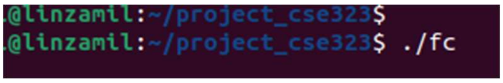
File name: finalcode.c

To compile this file, type following command:

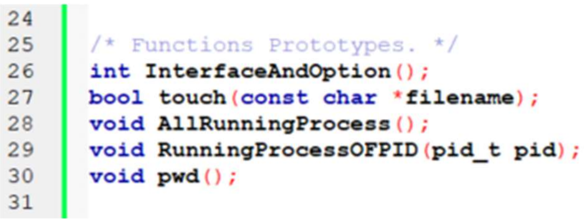


```
@linzamil:~/project_cse323$
@linzamil:~/project_cse323$ gcc finalcode.c -o fc -lprocps
@linzamil:~/project_cse323$
```

To execute, type following line:

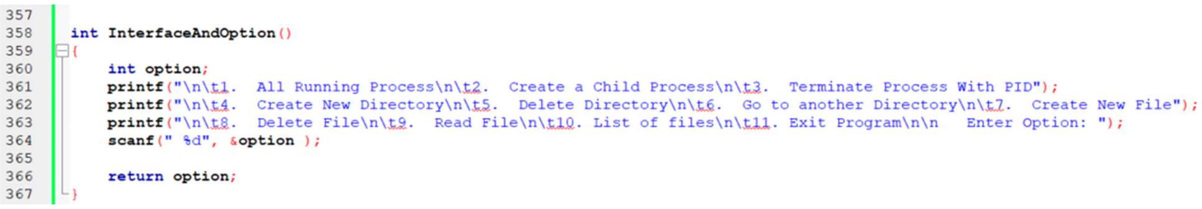


```
@linzamil:~/project_cse323$
@linzamil:~/project_cse323$ ./fc
```



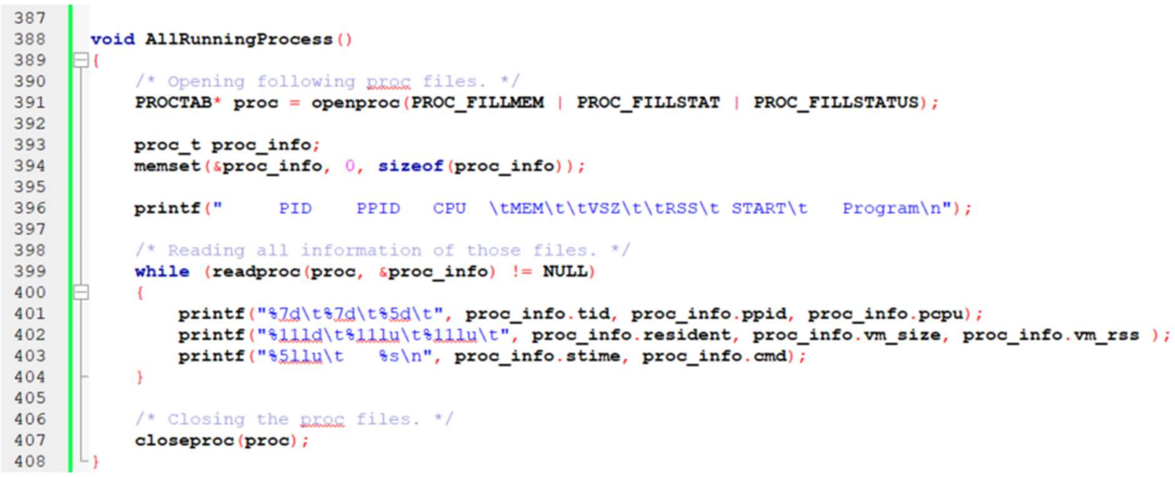
```
/* Functions Prototypes. */
int InterfaceAndOption();
bool touch(const char *filename);
void AllRunningProcess();
void RunningProcessOFPID(pid_t pid);
void pwd();
```

These are the function prototypes which are used in this program. It makes easy to call the functions whenever I need those.



```
int InterfaceAndOption()
{
    int option;
    printf("\n\t1.  All Running Process\n\t2.  Create a Child Process\n\t3.  Terminate Process With PID");
    printf("\n\t4.  Create New Directory\n\t5.  Delete Directory\n\t6.  Go to another Directory\n\t7.  Create New File");
    printf("\n\t8.  Delete File\n\t9.  Read File\n\t10. List of files\n\t11. Exit Program\n\n   Enter Option: ");
    scanf(" %d", &option );

    return option;
}
```

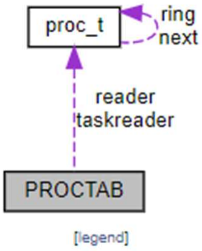
This is the interface which we will use for this program. It also shows the options/tasks which we can perform by using this program.

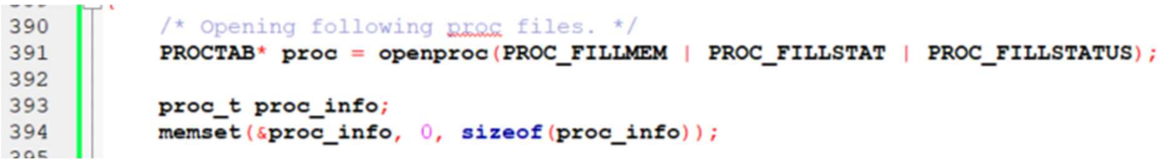


```c
void AllRunningProcess()
{
    /* Opening following proc files. */
    PROCTAB* proc = openproc(PROC_FILLMEM | PROC_FILLSTAT | PROC_FILLSTATUS);

    proc_t proc_info;
    memset(&proc_info, 0, sizeof(proc_info));

    printf("     PID    PPID   CPU  \tMEM\t\tVSZ\t\tRSS\t START\t   Program\n");

    /* Reading all information of those files. */
    while (readproc(proc, &proc_info) != NULL)
    {
        printf("%7d\t%7d\t%5d\t", proc_info.tid, proc_info.ppid, proc_info.pcpu);
        printf("%11ld\t%11lu\t%11lu\t", proc_info.resident, proc_info.vm_size, proc_info.vm_rss );
        printf("%5llu\t   %s\n", proc_info.stime, proc_info.cmd);
    }

    /* Closing the proc files. */
    closeproc(proc);
}
```

This code will give the information of all running process of my machine. Here **PROCTAB** structure is under **<proc/readproc.h>** header file which will help to read the information on current processes.

Collaboration diagram for PROCTAB:





```c
    /* Opening following proc files. */
    PROCTAB* proc = openproc(PROC_FILLMEM | PROC_FILLSTAT | PROC_FILLSTATUS);

    proc_t proc_info;
    memset(&proc_info, 0, sizeof(proc_info));
```

Using **openproc()** function I can get information on current processes

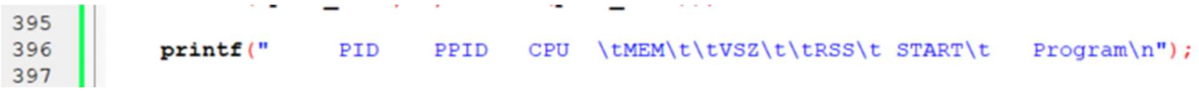
PROC_FILLMEM (read information from */proc/#pid/statm*),

PROC_FILLSTAT (read information from */proc/#pid/stat*),

PROC_FILLSTATUS (read information from */proc/#pid/status*).

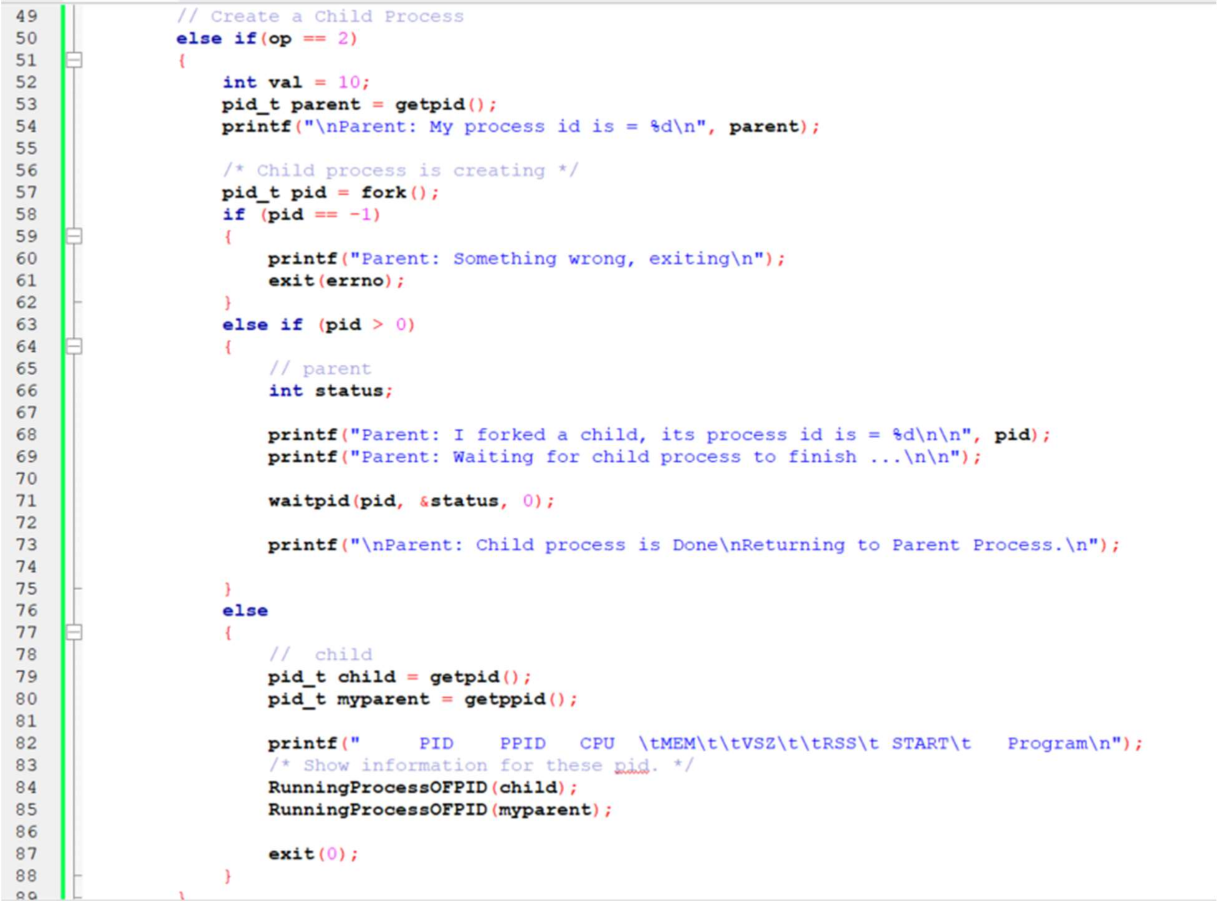
And store the information in **proc_info** which is a variable of **proc_t** structure while using **readproc()** function.

The **memset()** function sets the first count bytes of dest to the value c . The value of c is converted to an unsigned character. The **memset()** function returns a **pointer to dest**. It is also faster than a loop.



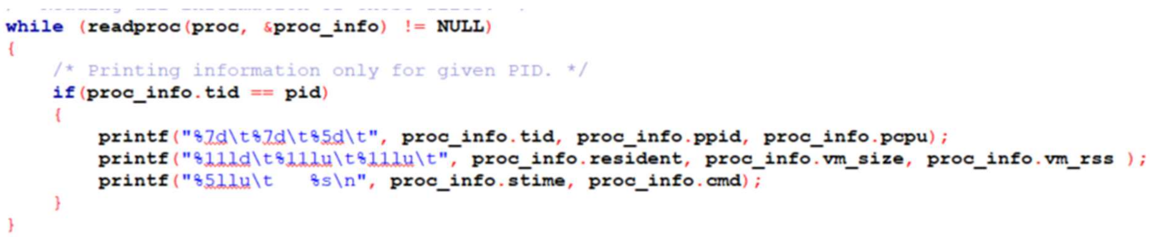
```
    printf("     PID     PPID    CPU   \tMEM\t\tVSZ\t\tRSS\t START\t    Program\n");
```

Then I show the PID, PPID, CPU, MEM, VSZ, RSS, START, Program command as output from reading the above stated files. PID is process id, PPID is parent id, MEM shows the utilization of memory for that process, VSZ is Virtual Memory Size which includes all memory that the process can access, including memory that is swapped out, memory that is allocated, but not used, and memory that is from shared libraries. RSS is Resident set size and is used to show how much memory is allocated to that process and is in RAM. It does not include memory that is swapped out. It does include memory from shared libraries as long as the pages from those libraries are actually in memory. It does include all stack and heap memory. After completing my task, I close those files using **closeproc()** function.



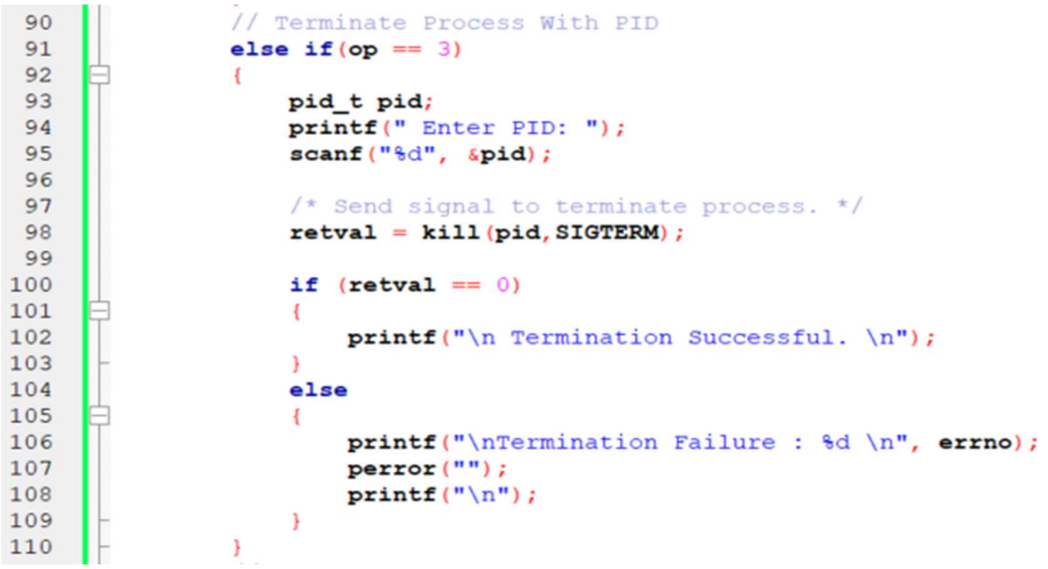
```
        // Create a Child Process
        else if(op == 2)
        {
            int val = 10;
            pid_t parent = getpid();
            printf("\nParent: My process id is = %d\n", parent);

            /* Child process is creating */
            pid_t pid = fork();
            if (pid == -1)
            {
                printf("Parent: Something wrong, exiting\n");
                exit(errno);
            }
            else if (pid > 0)
            {
                // parent
                int status;

                printf("Parent: I forked a child, its process id is = %d\n\n", pid);
                printf("Parent: Waiting for child process to finish ...\n\n");

                waitpid(pid, &status, 0);

                printf("\nParent: Child process is Done\nReturning to Parent Process.\n");

            }
            else
            {
                //  child
                pid_t child = getpid();
                pid_t myparent = getppid();

                printf("     PID     PPID    CPU   \tMEM\t\tVSZ\t\tRSS\t START\t    Program\n");
                /* Show information for these pid. */
                RunningProcessOFPID(child);
                RunningProcessOFPID(myparent);

                exit(0);
            }
        }
```

Then I create a child process using **fork()** and show the information about child and parent process alongside by calling **RunningProcessOFPID(pid_t pid)** function which only show the information for given pid. If **fork()** returns negative value then child process creation is unsuccessful. If it returns zero then creation of child process is successful. And if it returns positive value then it will go to parent or caller which contains process ID of newly created child process.



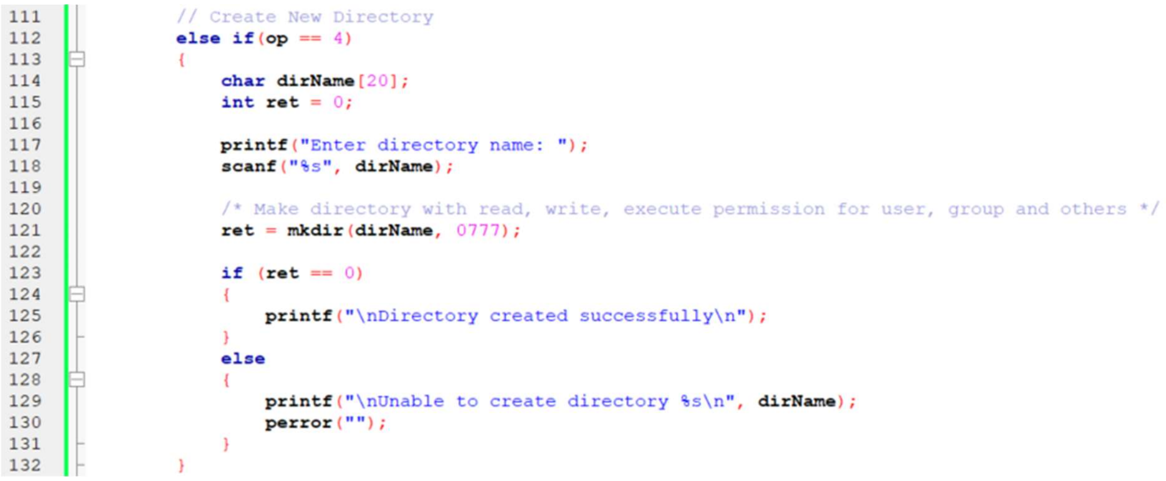
```
while (readproc(proc, &proc_info) != NULL)
{
    /* Printing information only for given PID. */
    if(proc_info.tid == pid)
    {
        printf("%7d\t%7d\t%5d\t", proc_info.tid, proc_info.ppid, proc_info.pcpu);
        printf("%11ld\t%11lu\t%11lu\t", proc_info.resident, proc_info.vm_size, proc_info.vm_rss );
        printf("%5llu\t   %s\n", proc_info.stime, proc_info.cmd);
    }
}
```

Parent process will wait until the child process done its work, after completing the assigned work to show information the child process will exit and parent process will start its work.



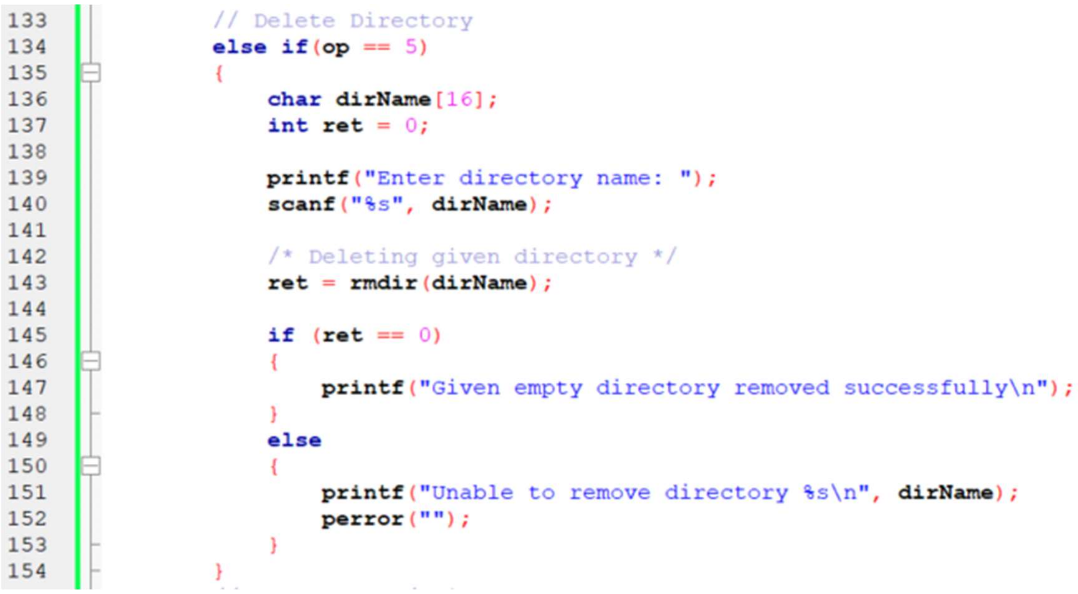
```
// Terminate Process With PID
else if(op == 3)
{
    pid_t pid;
    printf(" Enter PID: ");
    scanf("%d", &pid);

    /* Send signal to terminate process. */
    retval = kill(pid,SIGTERM);

    if (retval == 0)
    {
        printf("\n Termination Successful. \n");
    }
    else
    {
        printf("\nTermination Failure : %d \n", errno);
        perror("");
        printf("\n");
    }
}
```

This is to terminate any program with PID which we can get from **AllRunningProcess()** function. I will take the **PID** from user and terminate it using **kill()** function. Kill function will take pid and a signal as its parameter. There are different kinds of signals to terminate a process. The default and safest one is **SIGTERM**. Its signal number is **15**. If **kill()** function returns 0, then process termination is successful; otherwise, an error will occur and will be shown in terminal.



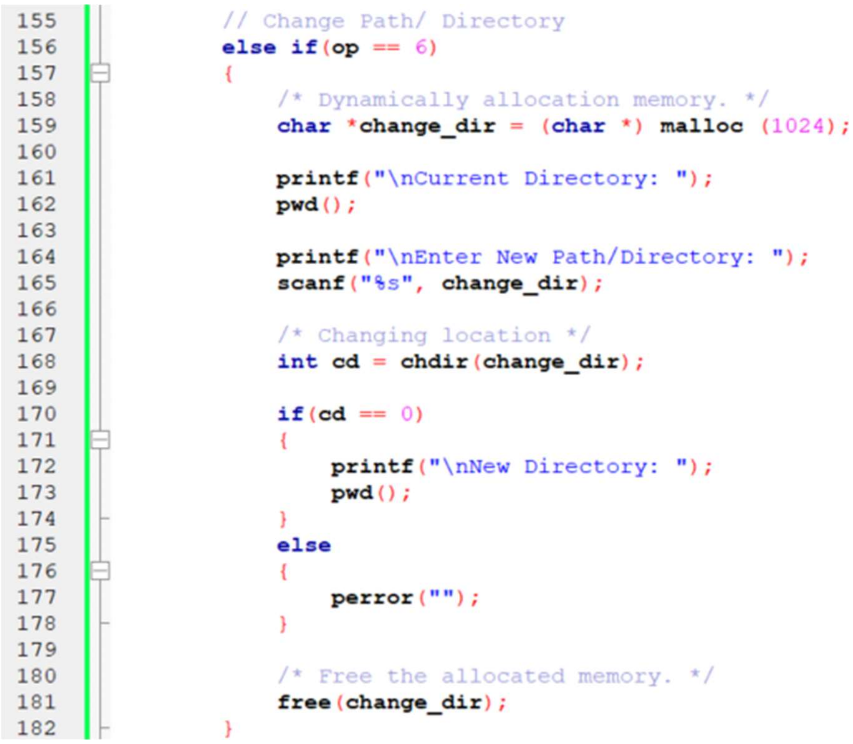
```
        // Create New Directory
        else if(op == 4)
        {
            char dirName[20];
            int ret = 0;

            printf("Enter directory name: ");
            scanf("%s", dirName);

            /* Make directory with read, write, execute permission for user, group and others */
            ret = mkdir(dirName, 0777);

            if (ret == 0)
            {
                printf("\nDirectory created successfully\n");
            }
            else
            {
                printf("\nUnable to create directory %s\n", dirName);
                perror("");
            }
        }
```

This code is to make new directory using **mkdir()** function given by user input. The **mkdir()** function to create a new directory with the name specified by the user and the permissions specified by the second argument (in this case, **“0777”** which allows the user, group, and others can read, write, and execute the directory). The return value of the **mkdir()** function is stored in the **“ret”** variable. If ret = = 0, then directory create successfully, otherwise an error message will be shown.



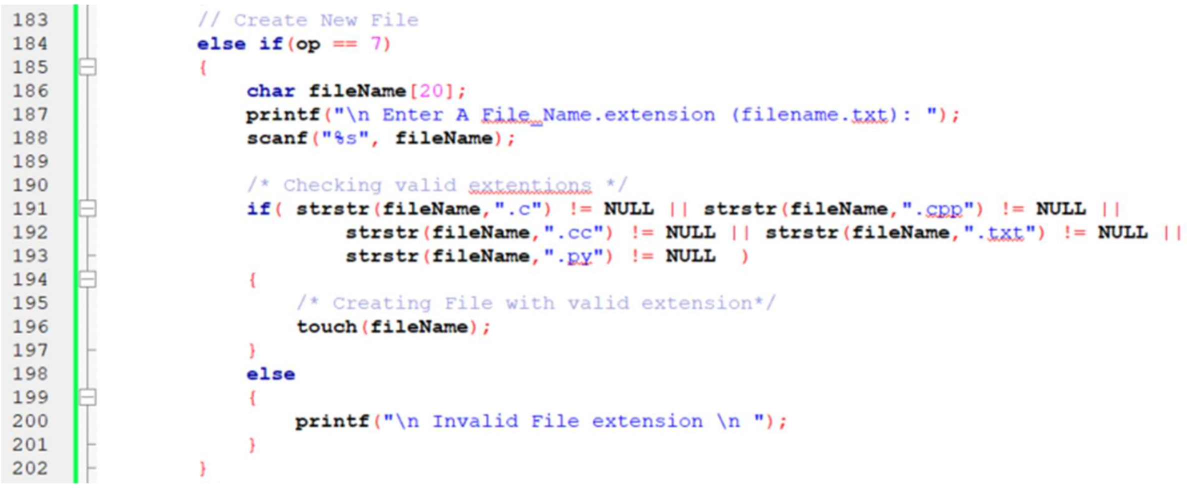
```
        // Delete Directory
        else if(op == 5)
        {
            char dirName[16];
            int ret = 0;

            printf("Enter directory name: ");
            scanf("%s", dirName);

            /* Deleting given directory */
            ret = rmdir(dirName);

            if (ret == 0)
            {
                printf("Given empty directory removed successfully\n");
            }
            else
            {
                printf("Unable to remove directory %s\n", dirName);
                perror("");
            }
        }
```

This code is for deleting an empty directory provided by user input. We use **rmdir()** function to remove the directory with the name specified by the user input. The return value of the **rmdir()** function is stored in the **“ret”** variable. If ret is equal to zero then directory will successfully remove, otherwise it will show an error message.



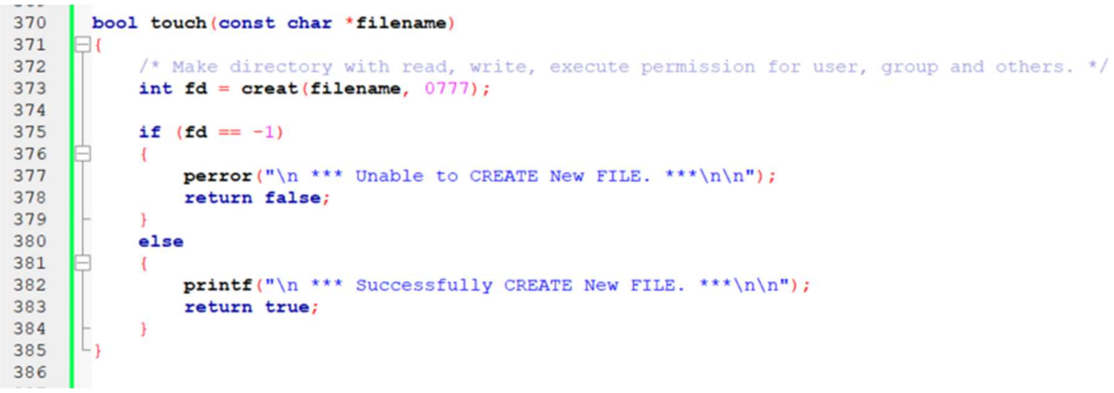
```c
// Change Path/ Directory
else if(op == 6)
{
    /* Dynamically allocation memory. */
    char *change_dir = (char *) malloc (1024);

    printf("\nCurrent Directory: ");
    pwd();

    printf("\nEnter New Path/Directory: ");
    scanf("%s", change_dir);

    /* Changing location */
    int cd = chdir(change_dir);

    if(cd == 0)
    {
        printf("\nNew Directory: ");
        pwd();
    }
    else
    {
        perror("");
    }

    /* Free the allocated memory. */
    free(change_dir);
}
```

Here we dynamically allocate the size of the variable then take a new path where we want to go. Then we call **chdir()** function to go to that location. The **chdir()** function will return zero if directory change is successful, otherwise give an error message.



```c
// Create New File
else if(op == 7)
{
    char fileName[20];
    printf("\n Enter A File Name.extension (filename.txt): ");
    scanf("%s", fileName);

    /* Checking valid extentions */
    if( strstr(fileName,".c") != NULL || strstr(fileName,".cpp") != NULL ||
            strstr(fileName,".cc") != NULL || strstr(fileName,".txt") != NULL ||
            strstr(fileName,".py") != NULL  )
    {
        /* Creating File with valid extension*/
        touch(fileName);
    }
    else
    {
        printf("\n Invalid File extension \n ");
    }
}
```

After check valid extension, we call the **touch()** function which create the file.

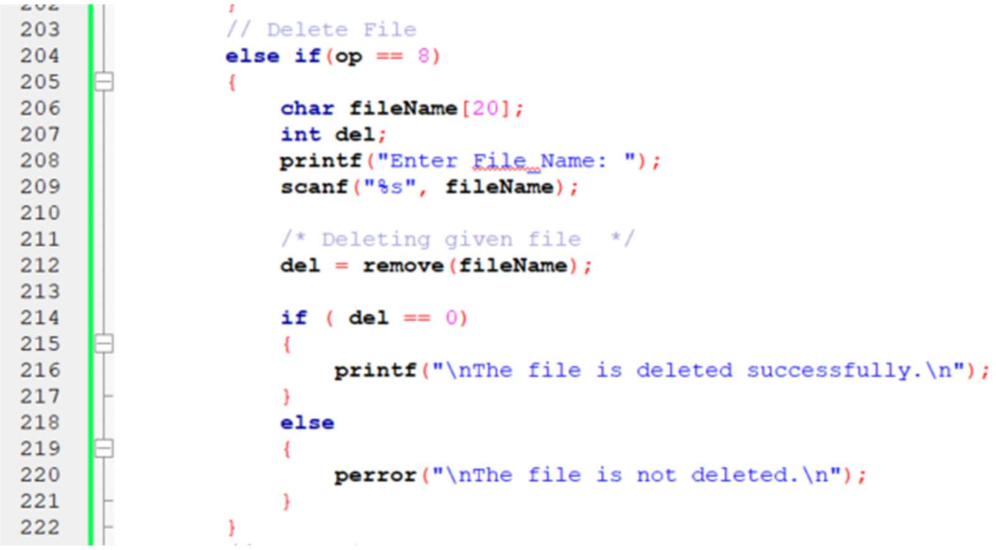


```
bool touch(const char *filename)
{
    /* Make directory with read, write, execute permission for user, group and others. */
    int fd = creat(filename, 0777);

    if (fd == -1)
    {
        perror("\n *** Unable to CREATE New FILE. ***\n\n");
        return false;
    }
    else
    {
        printf("\n *** Successfully CREATE New FILE. ***\n\n");
        return true;
    }
}
```

This code create file with the given name and valid extension using **creat()** function. And the **0777** is there to give permission for the user, group and others to have the read, write and execute access.

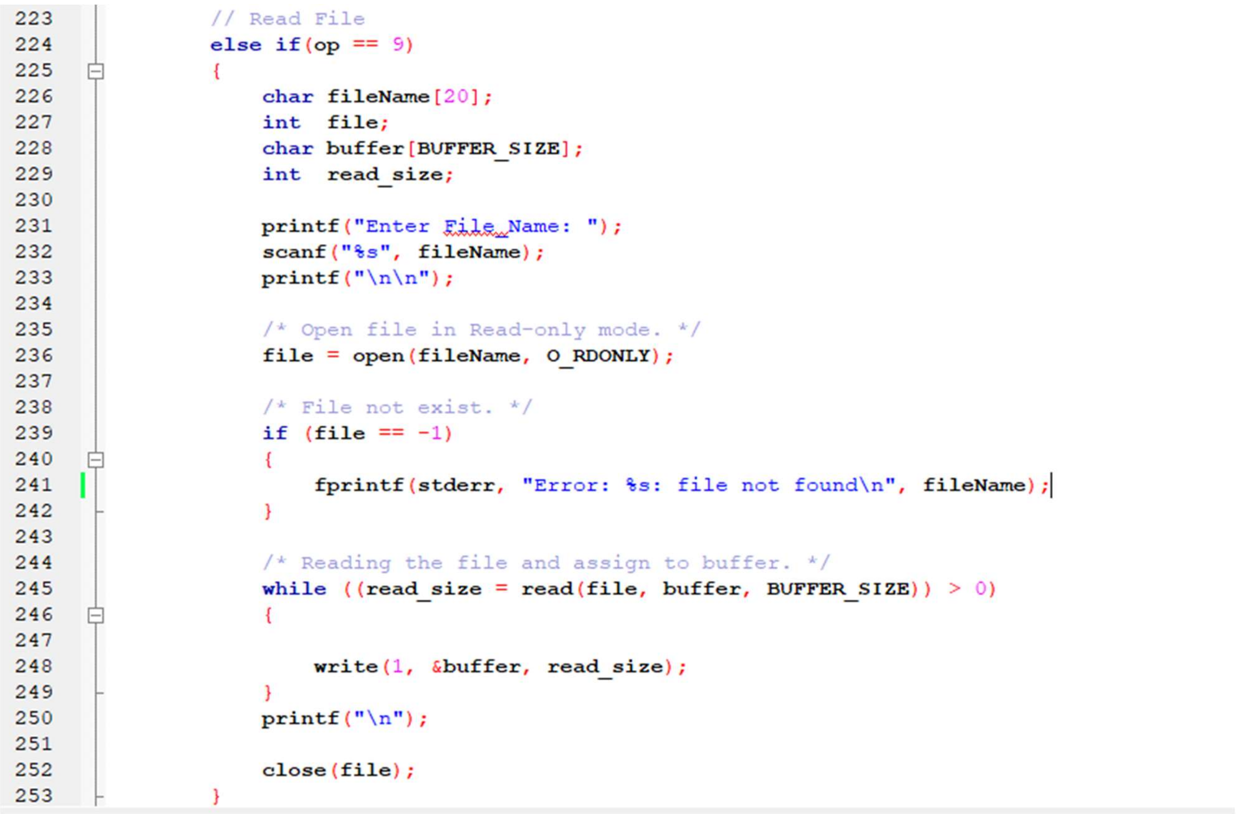
	User	**Group**	**Others**
Read = 4	x	x	x
Write = 2	x	x	x
Execute = 1	x	x	x
Total = 7	**7**	**7**	**7**

777 means users, groups and others all have their permission to read, write and execute the file. That means full access to the file.



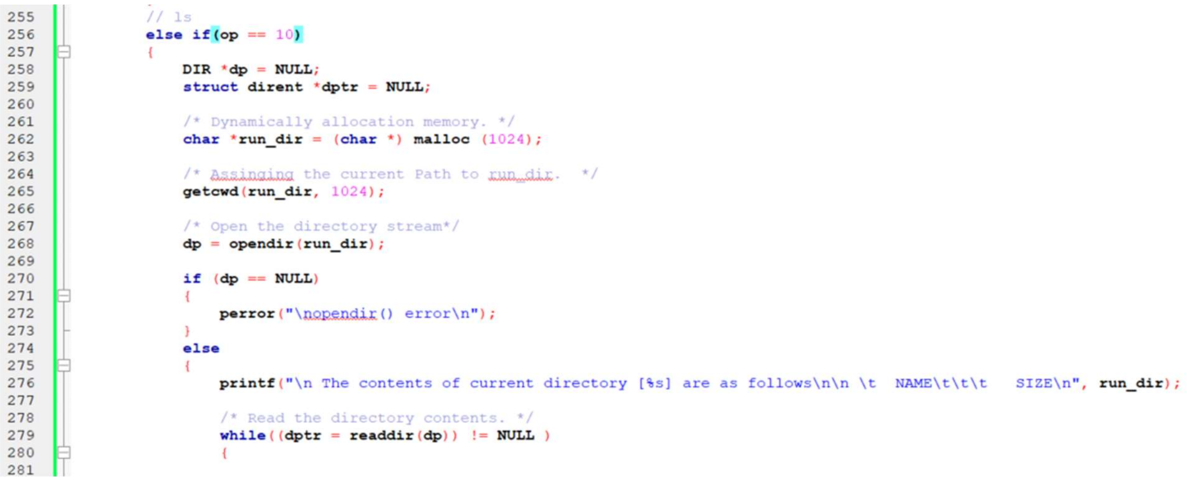
```
        // Delete File
        else if(op == 8)
        {
            char fileName[20];
            int del;
            printf("Enter File Name: ");
            scanf("%s", fileName);

            /* Deleting given file  */
            del = remove(fileName);

            if ( del == 0)
            {
                printf("\nThe file is deleted successfully.\n");
            }
            else
            {
                perror("\nThe file is not deleted.\n");
            }
        }
```

Here we call **remove()** function to delete a file specified by user input. It will return zero to **del** if the deletion of the file is successful, otherwise it will show an error message.



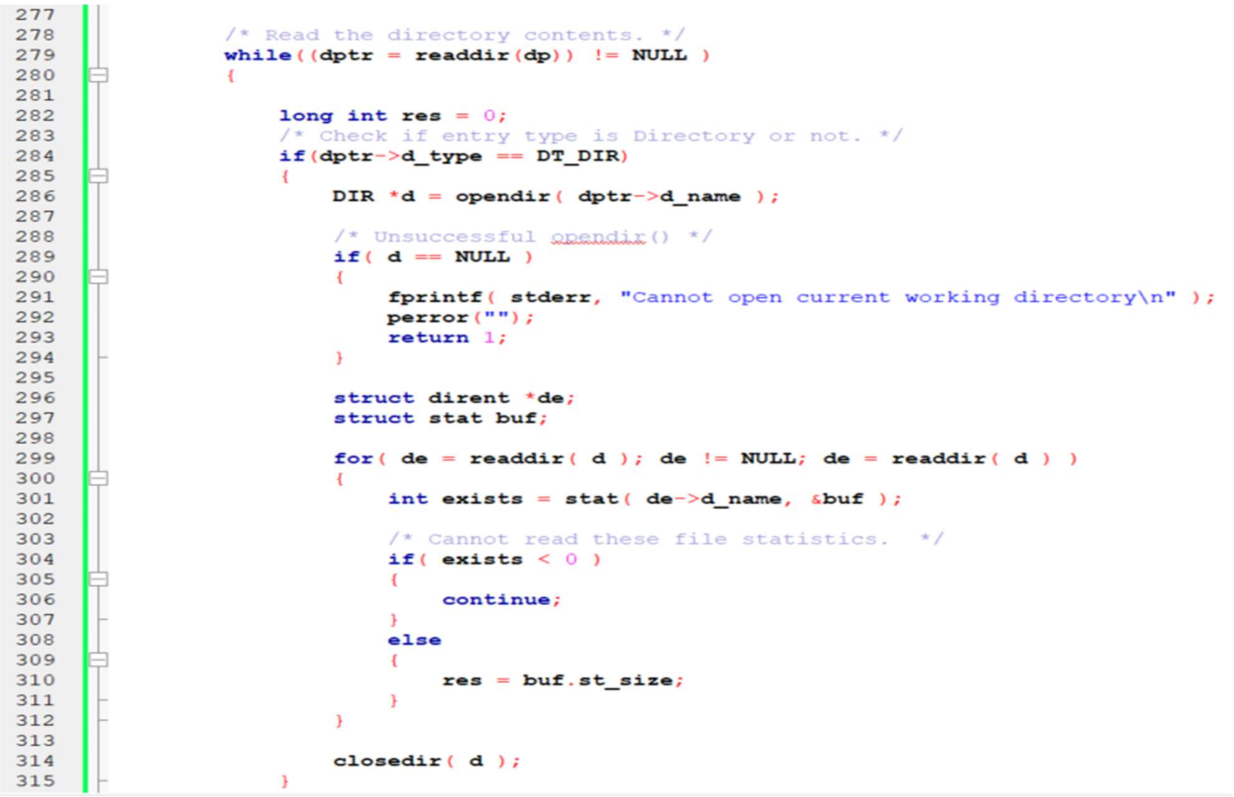
```
        // Read File
        else if(op == 9)
        {
            char fileName[20];
            int  file;
            char buffer[BUFFER_SIZE];
            int  read_size;

            printf("Enter File Name: ");
            scanf("%s", fileName);
            printf("\n\n");

            /* Open file in Read-only mode. */
            file = open(fileName, O_RDONLY);

            /* File not exist. */
            if (file == -1)
            {
                fprintf(stderr, "Error: %s: file not found\n", fileName);
            }

            /* Reading the file and assign to buffer. */
            while ((read_size = read(file, buffer, BUFFER_SIZE)) > 0)
            {

                write(1, &buffer, read_size);
            }
            printf("\n");

            close(file);
        }
```

Here we are taking the file name from the user than scanning to know if the file does exist. If not exists then file = = -1 which printing error. And the **open()** function opens the file in read only mode. Then in the while loop, we are reading the file and printing it in the command for the user. Taking constant buffer size to keep the data in flow. In while loop the **read()** function, read the file and if the file is true then it will give 1 to **read_size** and store the contents in buffer which is bigger than 0. So, the **write()** function will show the contents of that buffer.

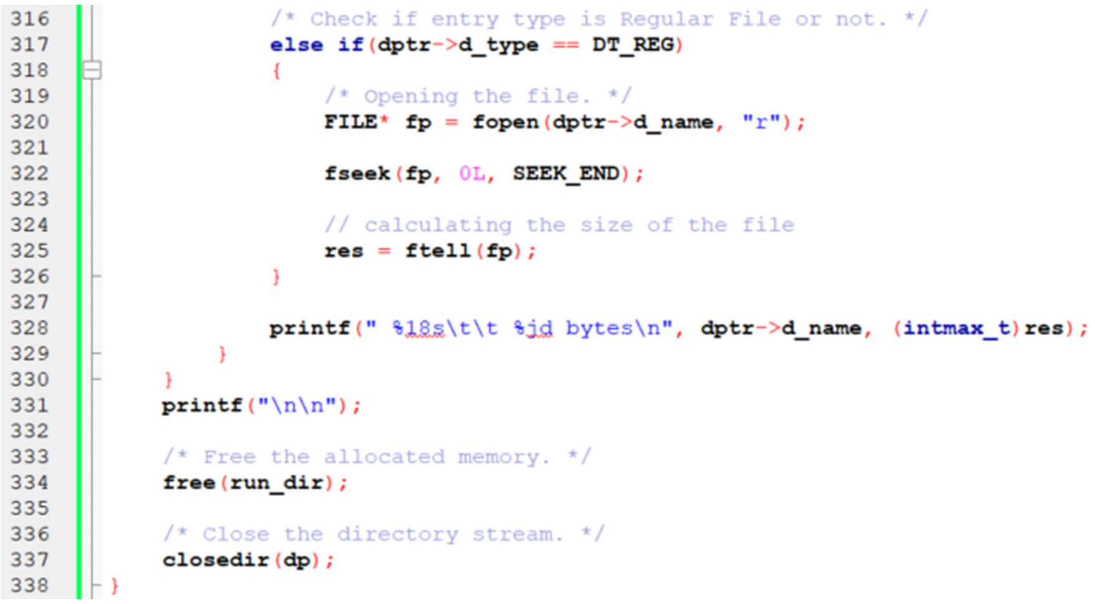


```
        // ls
        else if(op == 10)
        {
            DIR *dp = NULL;
            struct dirent *dptr = NULL;

            /* Dynamically allocation memory. */
            char *run_dir = (char *) malloc (1024);

            /* Assinging the current Path to run_dir.  */
            getcwd(run_dir, 1024);

            /* Open the directory stream*/
            dp = opendir(run_dir);

            if (dp == NULL)
            {
                perror("\nopendir() error\n");
            }
            else
            {
                printf("\n The contents of current directory [%s] are as follows\n\n \t  NAME\t\t\t   SIZE\n", run_dir);

                /* Read the directory contents. */
                while((dptr = readdir(dp)) != NULL )
                {
```

This code is for showing the files and directories name and size for current directory. Firstly, I get the directory path using **getcwd()** function. Then I call **opendir()** function which returns a pointer to a **DIR object**. If it returns NULL, then the function call is unsuccessful and show its corresponding error. After opening the directory, I call **readdir()** function. If successful, **readdir()** returns a pointer to a **dirent structure** describing the next directory entry in the directory stream. When **readdir()** reaches the end of the directory stream, it returns a NULL pointer. If unsuccessful, **readdir()** returns a NULL pointer and sets corresponding errno.

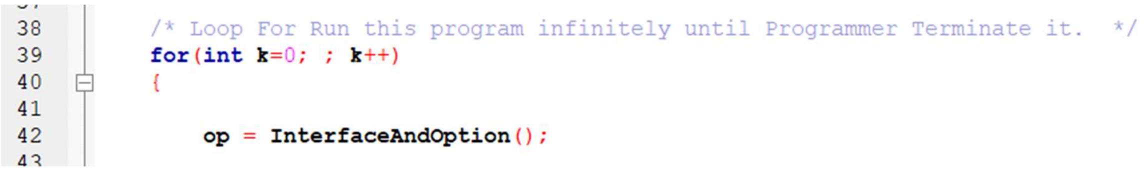


```
/* Read the directory contents. */
while((dptr = readdir(dp)) != NULL )
{

    long int res = 0;
    /* Check if entry type is Directory or not. */
    if(dptr->d_type == DT_DIR)
    {
        DIR *d = opendir( dptr->d_name );

        /* Unsuccessful opendir() */
        if( d == NULL )
        {
            fprintf( stderr, "Cannot open current working directory\n" );
            perror("");
            return 1;
        }

        struct dirent *de;
        struct stat buf;

        for( de = readdir( d ); de != NULL; de = readdir( d ) )
        {
            int exists = stat( de->d_name, &buf );

            /* Cannot read these file statistics.  */
            if( exists < 0 )
            {
                continue;
            }
            else
            {
                res = buf.st_size;
            }
        }

        closedir( d );
    }
```



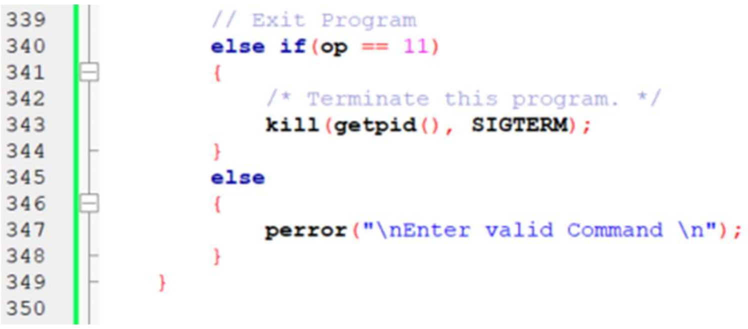
```
            /* Check if entry type is Regular File or not. */
            else if(dptr->d_type == DT_REG)
            {
                /* Opening the file. */
                FILE* fp = fopen(dptr->d_name, "r");

                fseek(fp, 0L, SEEK_END);

                // calculating the size of the file
                res = ftell(fp);
            }

            printf(" %18s\t\t %jd bytes\n", dptr->d_name, (intmax_t)res);
        }
    }
    printf("\n\n");

    /* Free the allocated memory. */
    free(run_dir);

    /* Close the directory stream. */
    closedir(dp);
}
```

Then, I check if the contents are Directory or file using **DT_DIR** a constant for directory and **DT_REG** a constant for regular files. After that I calculate the file / directory size and list the file/ directory names and their size. The size is converted to the longest integer type by using **intmax_t** – it's the same as the long or the long long type. The absolute value of a number (n) is the non-negative value of n. If the directory is empty then its minimum size will be 4096 bytes. Because in Linux minimum block size or smallest allocation unite for file system is 4096 bytes in the disk. Then I close the directory stream by calling **closedir()** function. It frees the buffer which I was using when I call **readdir()**. These functions are under **<dirent.h>** header file.



```
/* Loop For Run this program infinitely until Programmer Terminate it.  */
for(int k=0; ; k++)
{

    op = InterfaceAndOption();
```

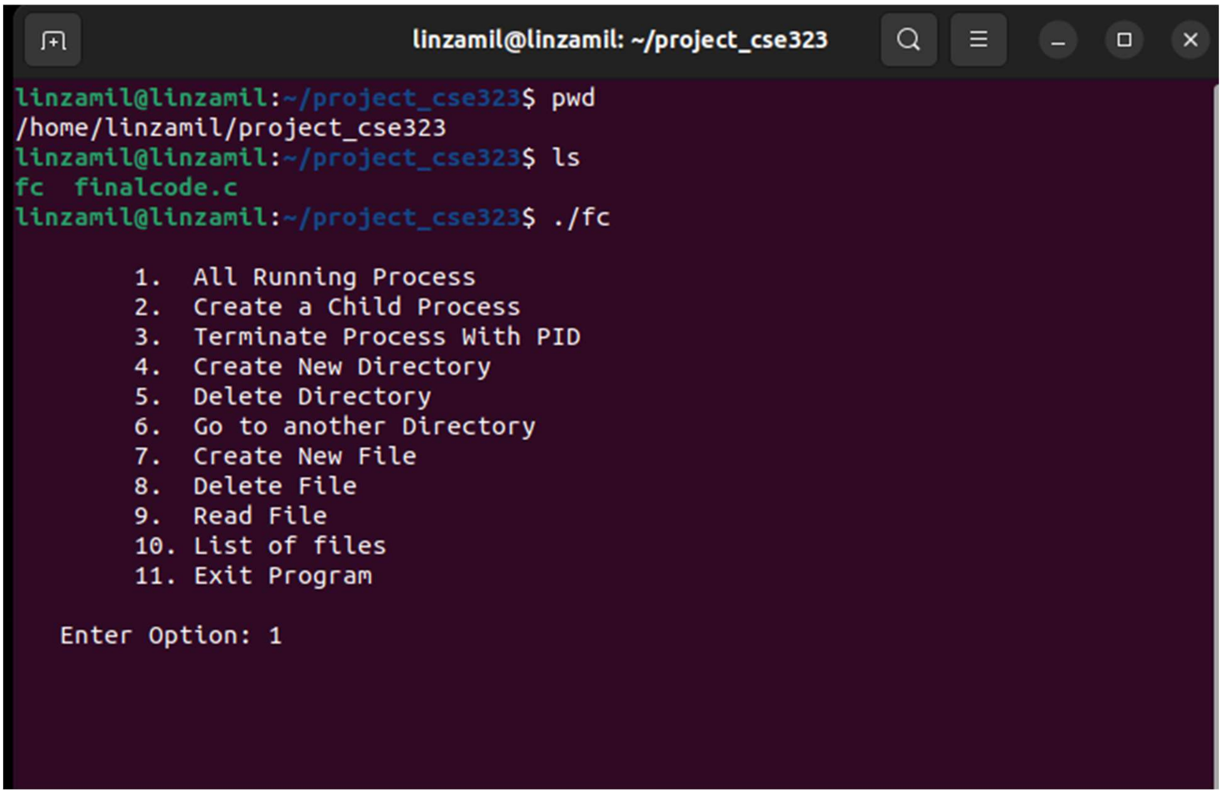
This is the infinite loop for our program to run until user press 11.



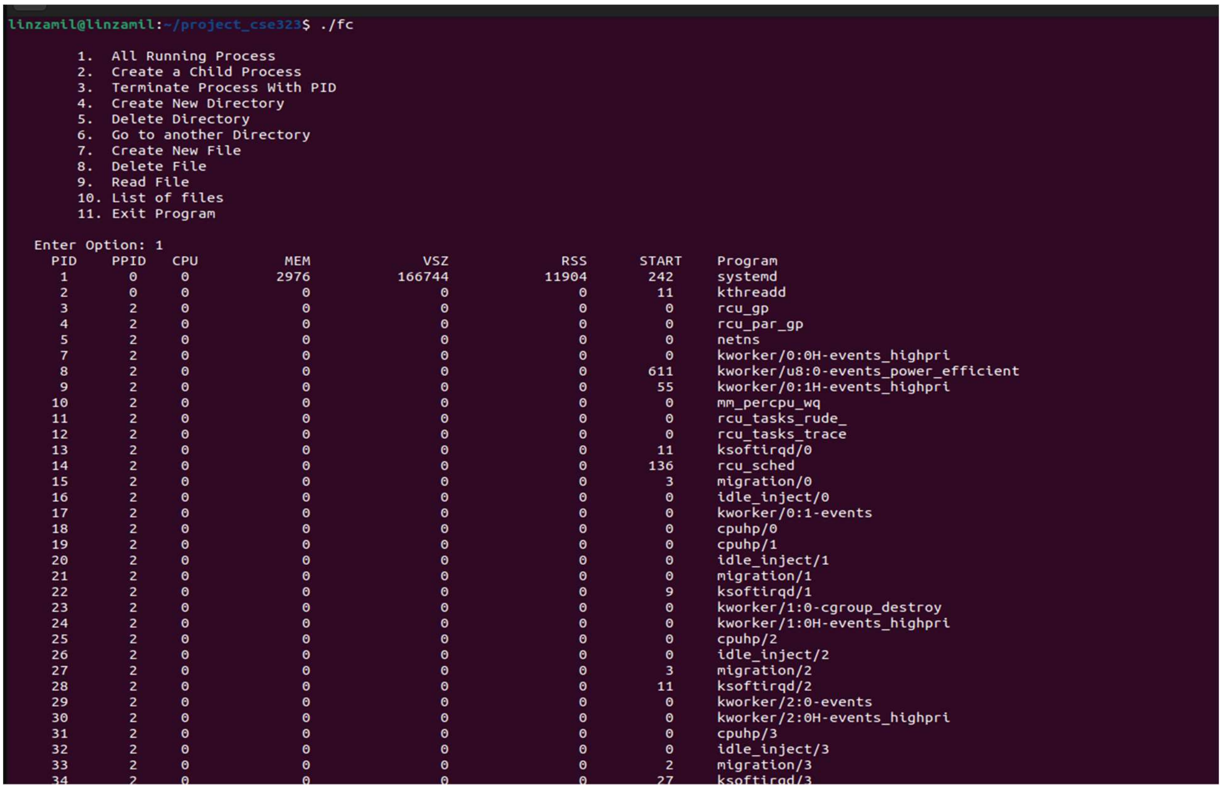
```
        // Exit Program
        else if(op == 11)
        {
            /* Terminate this program. */
            kill(getpid(), SIGTERM);
        }
        else
        {
            perror("\nEnter valid Command \n");
        }
    }
```

Lastly, I terminated this infinite loop, when user enter option 11. It passes the pid of this process and send **SIGTERM** signal to **kill()** function to terminate the program.

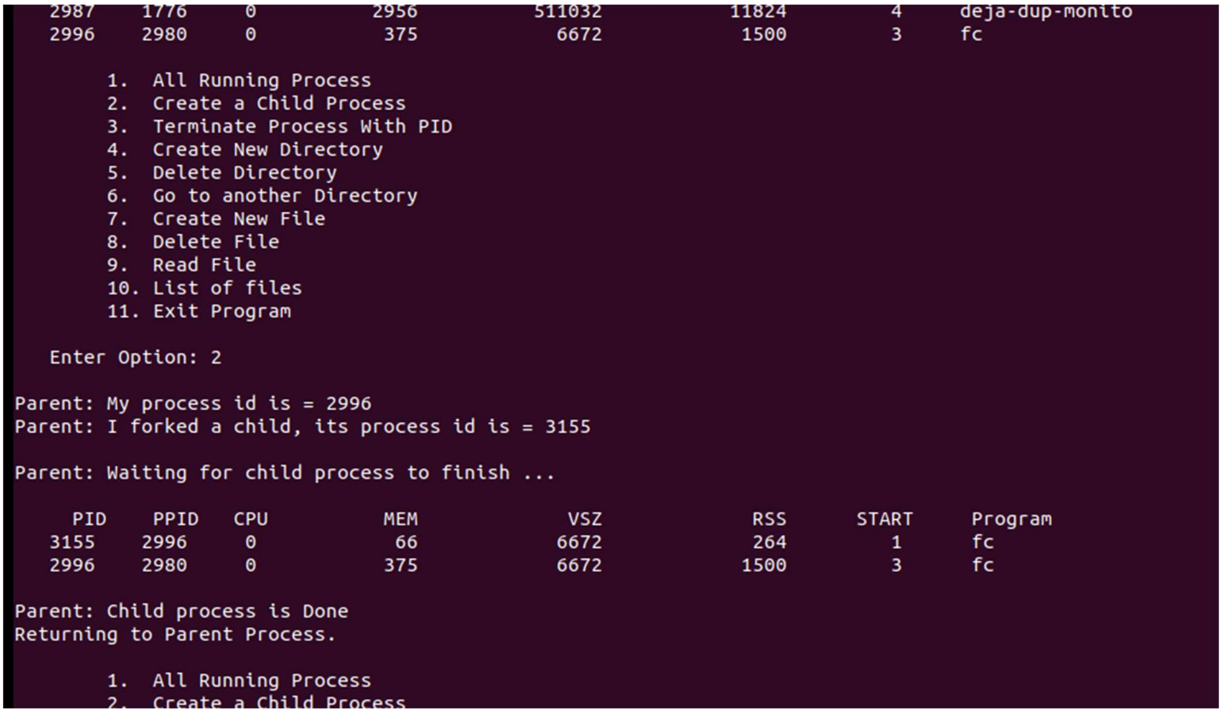
Results (Screenshots):



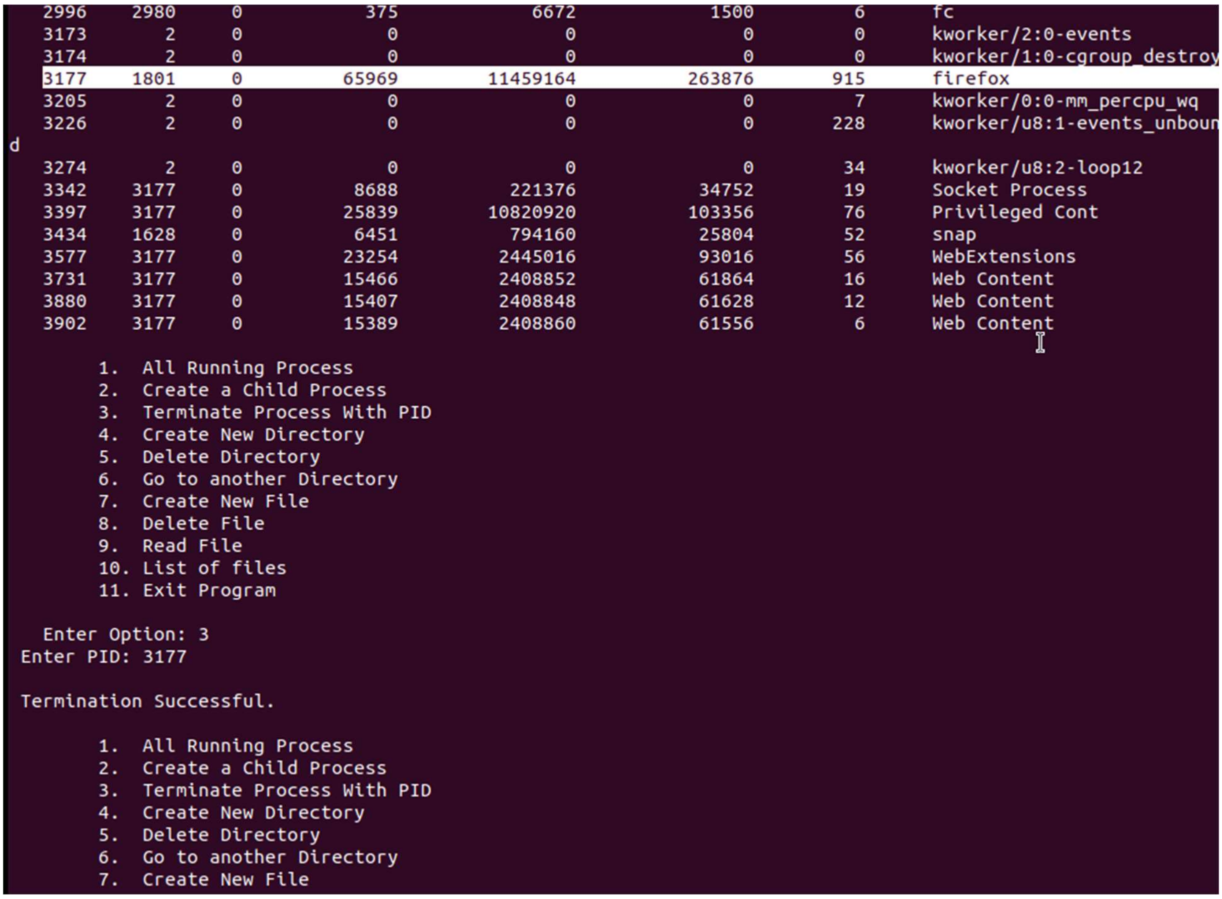
```
linzamil@linzamil:~/project_cse323$ pwd
/home/linzamil/project_cse323
linzamil@linzamil:~/project_cse323$ ls
fc  finalcode.c
linzamil@linzamil:~/project_cse323$ ./fc

        1.  All Running Process
        2.  Create a Child Process
        3.  Terminate Process With PID
        4.  Create New Directory
        5.  Delete Directory
        6.  Go to another Directory
        7.  Create New File
        8.  Delete File
        9.  Read File
        10. List of files
        11. Exit Program

   Enter Option: 1
```



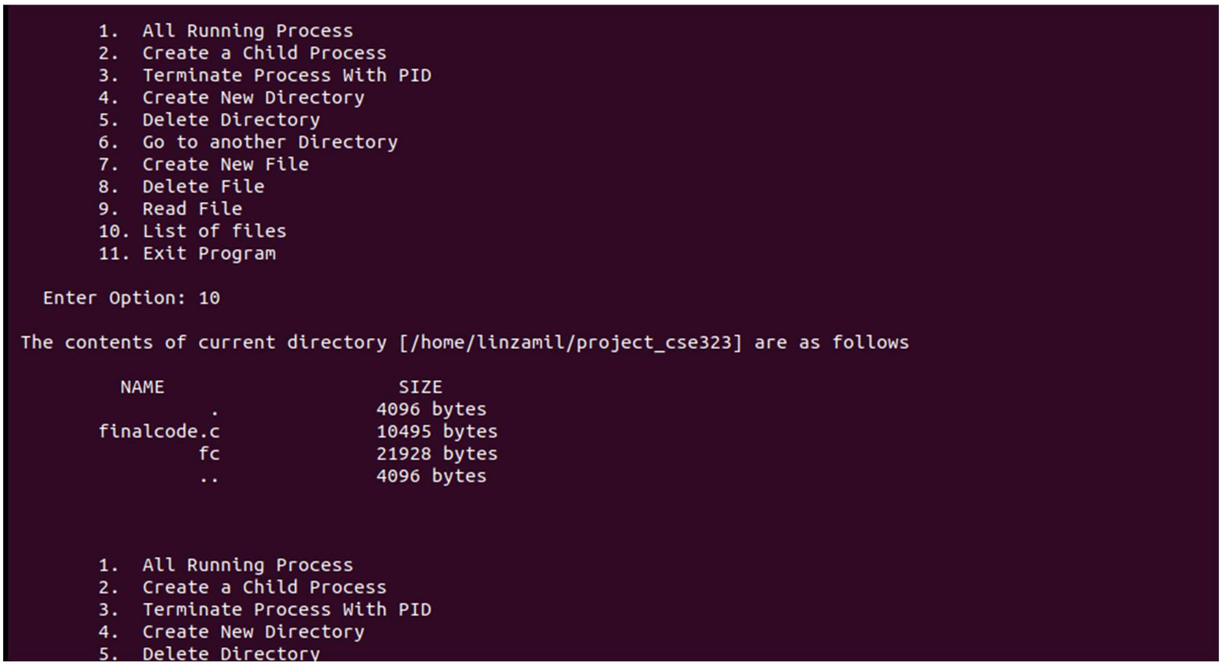
```
linzamil@linzamil:~/project_cse323$ ./fc

        1.  All Running Process
        2.  Create a Child Process
        3.  Terminate Process With PID
        4.  Create New Directory
        5.  Delete Directory
        6.  Go to another Directory
        7.  Create New File
        8.  Delete File
        9.  Read File
        10. List of files
        11. Exit Program

   Enter Option: 1
   PID   PPID  CPU         MEM            VSZ            RSS    START    Program
     1      0    0        2976         166744          11904      242    systemd
     2      0    0           0              0              0       11    kthreadd
     3      2    0           0              0              0        0    rcu_gp
     4      2    0           0              0              0        0    rcu_par_gp
     5      2    0           0              0              0        0    netns
     7      2    0           0              0              0        0    kworker/0:0H-events_highpri
     8      2    0           0              0              0      611    kworker/u8:0-events_power_efficient
     9      2    0           0              0              0       55    kworker/0:1H-events_highpri
    10      2    0           0              0              0        0    mm_percpu_wq
    11      2    0           0              0              0        0    rcu_tasks_rude_
    12      2    0           0              0              0        0    rcu_tasks_trace
    13      2    0           0              0              0       11    ksoftirqd/0
    14      2    0           0              0              0      136    rcu_sched
    15      2    0           0              0              0        3    migration/0
    16      2    0           0              0              0        0    idle_inject/0
    17      2    0           0              0              0        0    kworker/0:1-events
    18      2    0           0              0              0        0    cpuhp/0
    19      2    0           0              0              0        0    cpuhp/1
    20      2    0           0              0              0        0    idle_inject/1
    21      2    0           0              0              0        0    migration/1
    22      2    0           0              0              0        9    ksoftirqd/1
    23      2    0           0              0              0        0    kworker/1:0-cgroup_destroy
    24      2    0           0              0              0        0    kworker/1:0H-events_highpri
    25      2    0           0              0              0        0    cpuhp/2
    26      2    0           0              0              0        0    idle_inject/2
    27      2    0           0              0              0        3    migration/2
    28      2    0           0              0              0       11    ksoftirqd/2
    29      2    0           0              0              0        0    kworker/2:0-events
    30      2    0           0              0              0        0    kworker/2:0H-events_highpri
    31      2    0           0              0              0        0    cpuhp/3
    32      2    0           0              0              0        0    idle_inject/3
    33      2    0           0              0              0        2    migration/3
    34      2    0           0              0              0       27    ksoftirqd/3
```



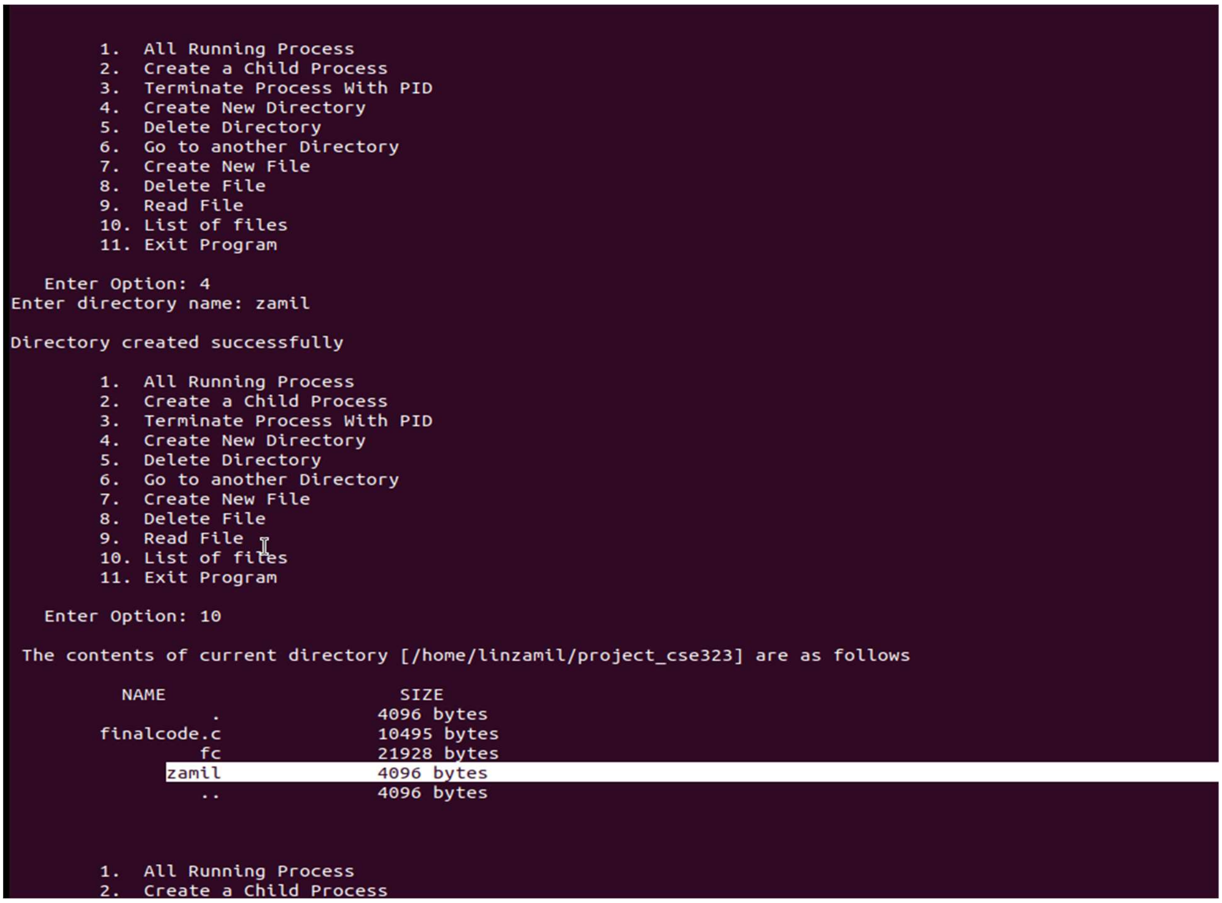
```
   2987    1776     0           2956          511032          11824         4     deja-dup-monito
   2996    2980     0            375            6672           1500         3     fc

        1.  All Running Process
        2.  Create a Child Process
        3.  Terminate Process With PID
        4.  Create New Directory
        5.  Delete Directory
        6.  Go to another Directory
        7.  Create New File
        8.  Delete File
        9.  Read File
        10. List of files
        11. Exit Program

   Enter Option: 2

Parent: My process id is = 2996
Parent: I forked a child, its process id is = 3155

Parent: Waiting for child process to finish ...

     PID     PPID   CPU           MEM              VSZ              RSS       START      Program
   3155     2996     0             66             6672              264         1      fc
   2996     2980     0            375             6672             1500         3      fc

Parent: Child process is Done
Returning to Parent Process.

        1.  All Running Process
        2.  Create a Child Process
```



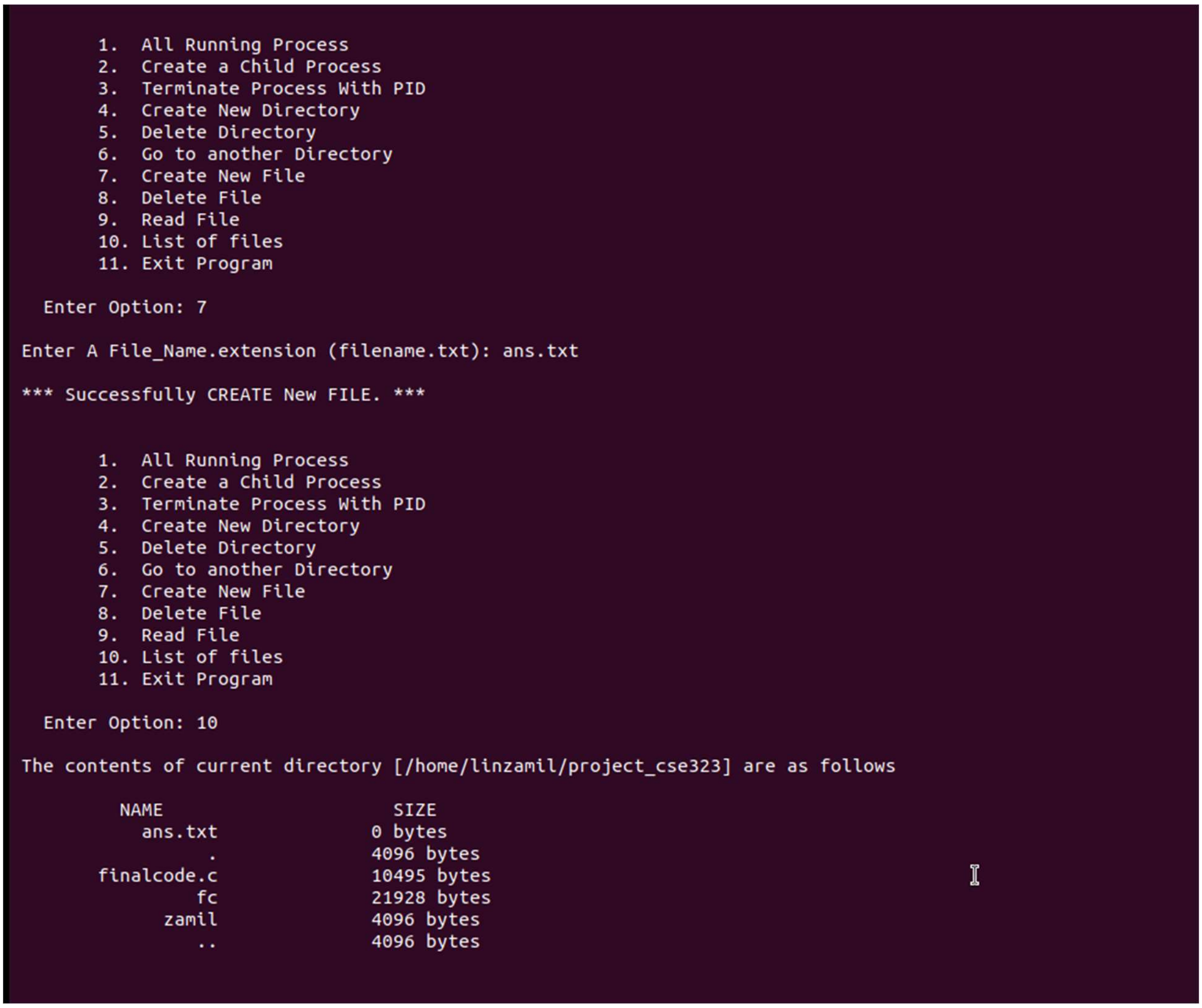
```
   2996    2980     0          375          6672          1500        6       fc
   3173       2     0            0             0             0        0       kworker/2:0-events
   3174       2     0            0             0             0        0       kworker/1:0-cgroup_destroy
   3177    1801     0        65969      11459164        263876      915       firefox
   3205       2     0            0             0             0        7       kworker/0:0-mm_percpu_wq
   3226       2     0            0             0             0      228       kworker/u8:1-events_unboun
d
   3274       2     0            0             0             0       34       kworker/u8:2-loop12
   3342    3177     0         8688        221376         34752       19       Socket Process
   3397    3177     0        25839      10820920        103356       76       Privileged Cont
   3434    1628     0         6451        794160         25804       52       snap
   3577    3177     0        23254       2445016         93016       56       WebExtensions
   3731    3177     0        15466       2408852         61864       16       Web Content
   3880    3177     0        15407       2408848         61628       12       Web Content
   3902    3177     0        15389       2408860         61556        6       Web Content

        1.  All Running Process
        2.  Create a Child Process
        3.  Terminate Process With PID
        4.  Create New Directory
        5.  Delete Directory
        6.  Go to another Directory
        7.  Create New File
        8.  Delete File
        9.  Read File
        10. List of files
        11. Exit Program

   Enter Option: 3
 Enter PID: 3177

 Termination Successful.

        1.  All Running Process
        2.  Create a Child Process
        3.  Terminate Process With PID
        4.  Create New Directory
        5.  Delete Directory
        6.  Go to another Directory
        7.  Create New File
```



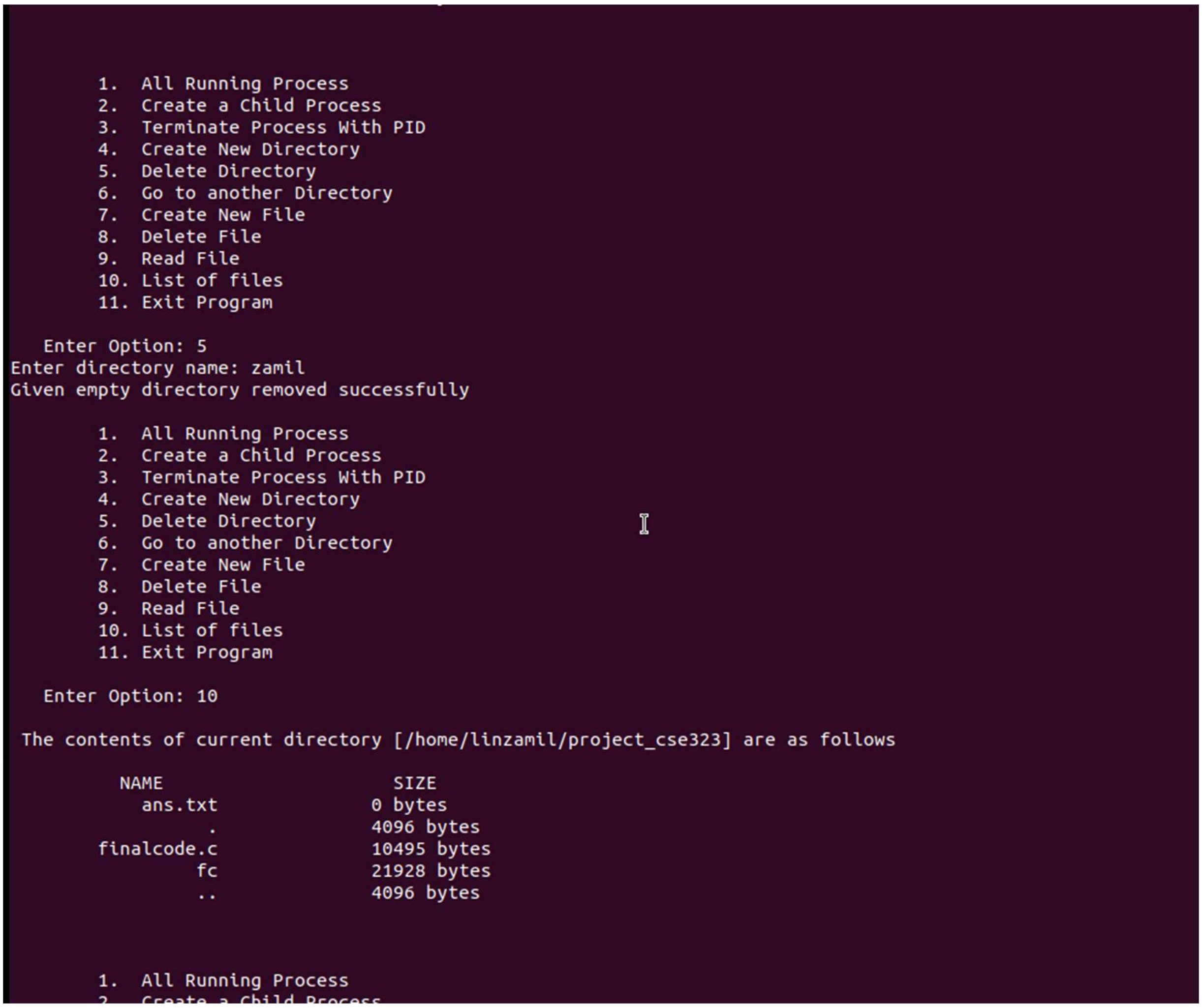
```
        1.  All Running Process
        2.  Create a Child Process
        3.  Terminate Process With PID
        4.  Create New Directory
        5.  Delete Directory
        6.  Go to another Directory
        7.  Create New File
        8.  Delete File
        9.  Read File
        10. List of files
        11. Exit Program

   Enter Option: 10

 The contents of current directory [/home/linzamil/project_cse323] are as follows

          NAME                        SIZE
                  .                 4096 bytes
        finalcode.c                 10495 bytes
                 fc                 21928 bytes
                 ..                 4096 bytes



        1.  All Running Process
        2.  Create a Child Process
        3.  Terminate Process With PID
        4.  Create New Directory
        5.  Delete Directory
```



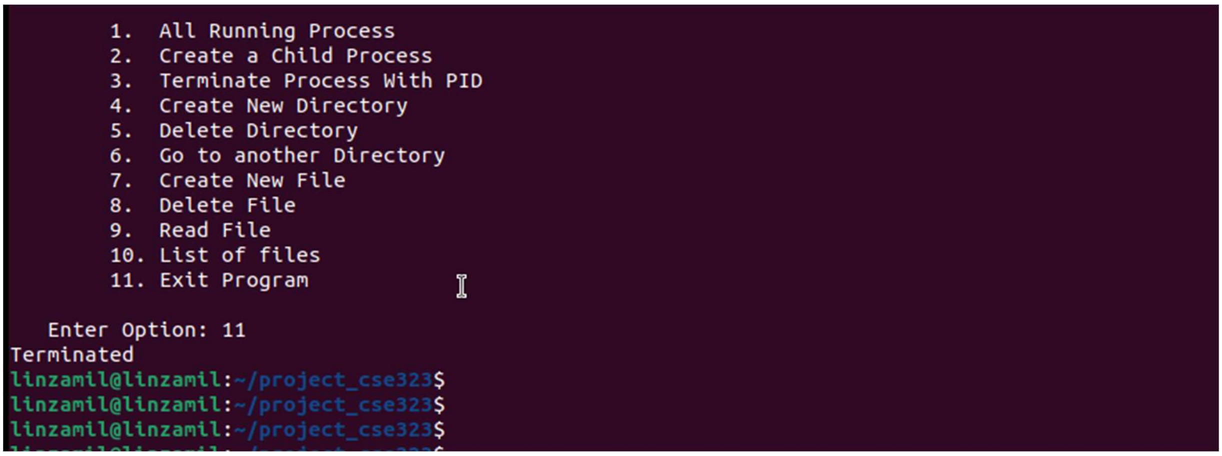
```
        1.  All Running Process
        2.  Create a Child Process
        3.  Terminate Process With PID
        4.  Create New Directory
        5.  Delete Directory
        6.  Go to another Directory
        7.  Create New File
        8.  Delete File
        9.  Read File
        10. List of files
        11. Exit Program

   Enter Option: 4
Enter directory name: zamil

Directory created successfully

        1.  All Running Process
        2.  Create a Child Process
        3.  Terminate Process With PID
        4.  Create New Directory
        5.  Delete Directory
        6.  Go to another Directory
        7.  Create New File
        8.  Delete File
        9.  Read File
        10. List of files
        11. Exit Program

   Enter Option: 10

 The contents of current directory [/home/linzamil/project_cse323] are as follows

          NAME                        SIZE
                  .                 4096 bytes
        finalcode.c                 10495 bytes
                 fc                 21928 bytes
             zamil                  4096 bytes
                 ..                 4096 bytes



        1.  All Running Process
        2.  Create a Child Process
```



```
       1.  All Running Process
       2.  Create a Child Process
       3.  Terminate Process With PID
       4.  Create New Directory
       5.  Delete Directory
       6.  Go to another Directory
       7.  Create New File
       8.  Delete File
       9.  Read File
       10. List of files
       11. Exit Program

  Enter Option: 7

Enter A File_Name.extension (filename.txt): ans.txt

*** Successfully CREATE New FILE. ***


       1.  All Running Process
       2.  Create a Child Process
       3.  Terminate Process With PID
       4.  Create New Directory
       5.  Delete Directory
       6.  Go to another Directory
       7.  Create New File
       8.  Delete File
       9.  Read File
       10. List of files
       11. Exit Program

  Enter Option: 10

The contents of current directory [/home/linzamil/project_cse323] are as follows

         NAME                     SIZE
           ans.txt              0 bytes
                 .              4096 bytes
       finalcode.c              10495 bytes
                fc              21928 bytes
             zamil              4096 bytes
                ..              4096 bytes
```



```
        1.  All Running Process
        2.  Create a Child Process
        3.  Terminate Process With PID
        4.  Create New Directory
        5.  Delete Directory
        6.  Go to another Directory
        7.  Create New File
        8.  Delete File
        9.  Read File
        10. List of files
        11. Exit Program

   Enter Option: 5
Enter directory name: zamil
Given empty directory removed successfully

        1.  All Running Process
        2.  Create a Child Process
        3.  Terminate Process With PID
        4.  Create New Directory
        5.  Delete Directory
        6.  Go to another Directory
        7.  Create New File
        8.  Delete File
        9.  Read File
        10. List of files
        11. Exit Program

   Enter Option: 10

 The contents of current directory [/home/linzamil/project_cse323] are as follows

          NAME                     SIZE
            ans.txt              0 bytes
                  .              4096 bytes
        finalcode.c              10495 bytes
                 fc              21928 bytes
                 ..              4096 bytes



        1.  All Running Process
        2.  Create a Child Process
```



```
        1.  All Running Process
        2.  Create a Child Process
        3.  Terminate Process With PID
        4.  Create New Directory
        5.  Delete Directory
        6.  Go to another Directory
        7.  Create New File
        8.  Delete File
        9.  Read File
        10. List of files
        11. Exit Program

   Enter Option: 11
Terminated
linzamil@linzamil:~/project_cse323$
linzamil@linzamil:~/project_cse323$
linzamil@linzamil:~/project_cse323$
linzamil@linzamil:~/project_cse323$
```

Further Enhancement:

We can reuse this code and add some extra features. The code has easy readability and comment out the programs as well. So, anyone can modify the code to implement certain problems. We make some functions which we can reuse to implement the following problems:

- We can use the information to make a schedular.
    - Program can check the usages and terminate or reallocate the process
- We can make something similar to Task Manager.
    - Program can to build a customized task manager
- We can modify this code to clear / free the cache memory.
    - When a process ends, it can automatically free the buffer and cache memory
- We can write in created files
- We can use dynamic allocations to fix the input lengths
- We can change some parts to delete a non-empty directory

Summary:

This process is about process management. Here we check all running process and its memory usage and PID. Then we create a new process known as child process, which only show the information of that process (child) and its parent process. Then we write code to terminate process with PID. This will exit that process. Then we make directory and delete directory with read, write, and execute permission for user, group and others. There is also a part where we can change the directory and perform the creation and deletion of files. After that, we give an option to read any file which has read permission. Then this program can also show the entries of a directory. Dynamically allocated memory was released before closing the code. And lastly, we add an exit option which will terminate this infinite loop and close the program. These is the tasks; we can perform from this program.

**** * * The End * * ****