Sudip Das

20210005

M.Tech (CSE)

Assignment 1

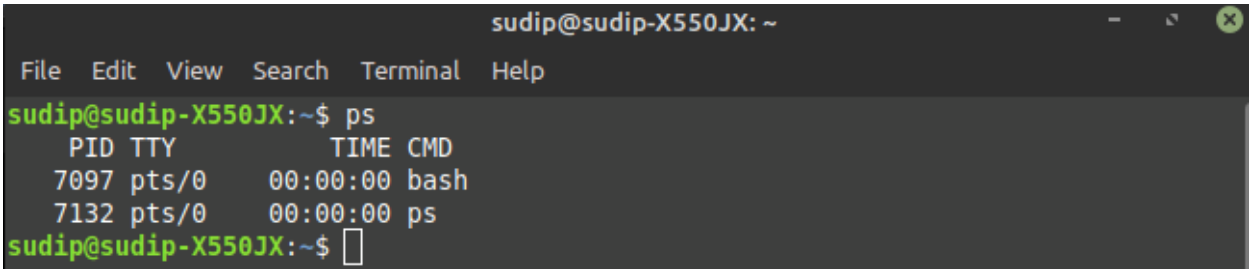
CS 612

https://github.com/David-jon/OS-Assignment-1-IITGN

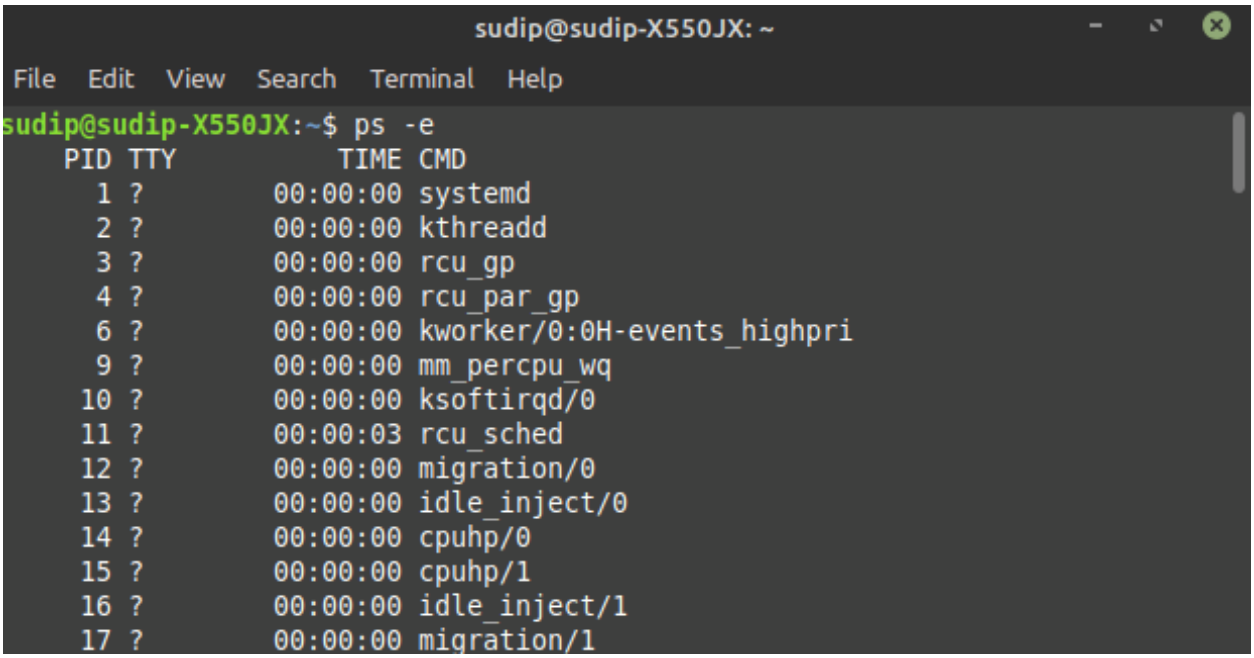
Part 1 - Running inbuilt binaries

a) **The ps command** – This function reports a snapshot of the current processes.

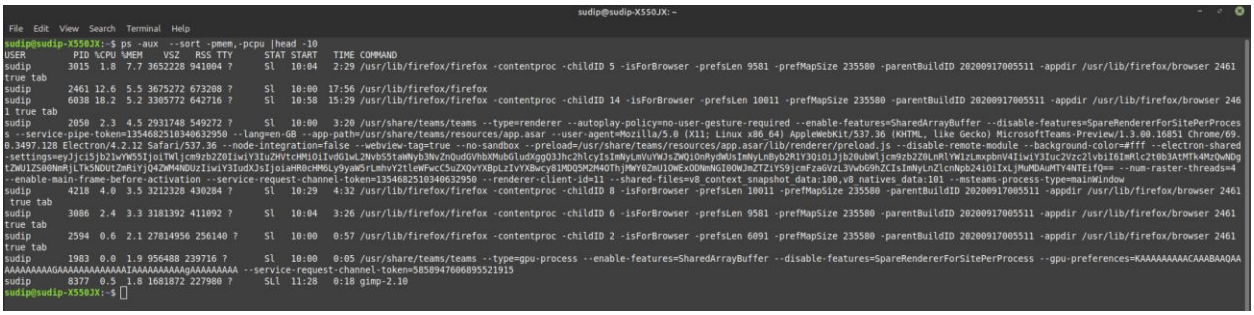
If it is not used without any arguments, it displays all the processes that is being run on the current shell



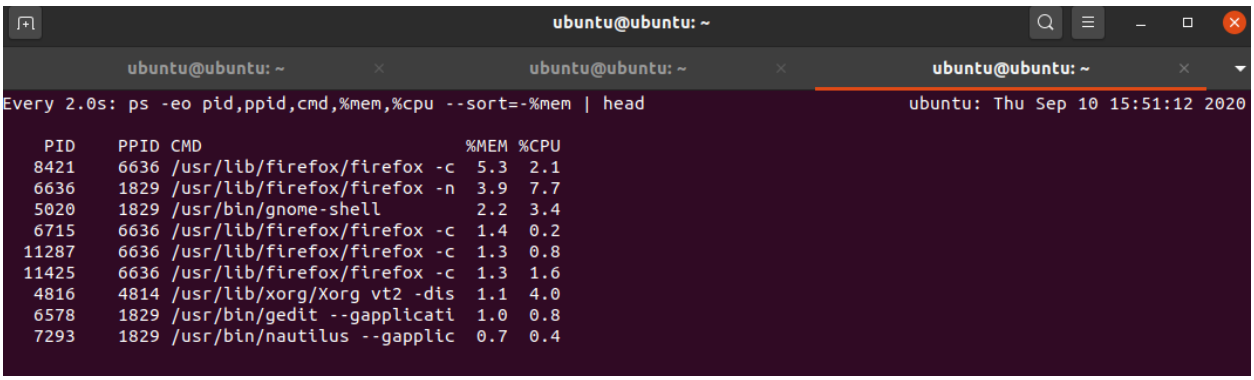
If we use the -e flag we can see all the processes and their details that are currently running in the system. By default the processes are sorted by their PID number

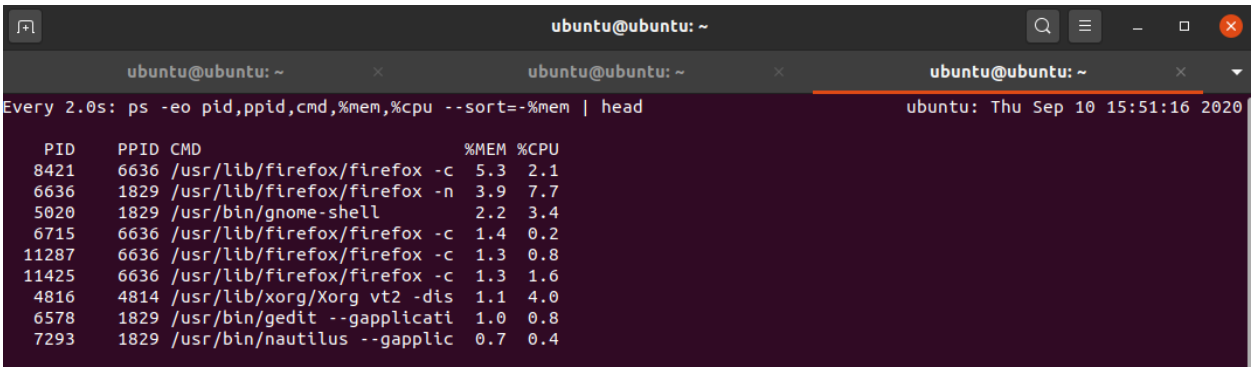


In this case I have sorted the output of the ps-aux by the memory and CPU consumption by decreasing order. Then printed the first 10 values

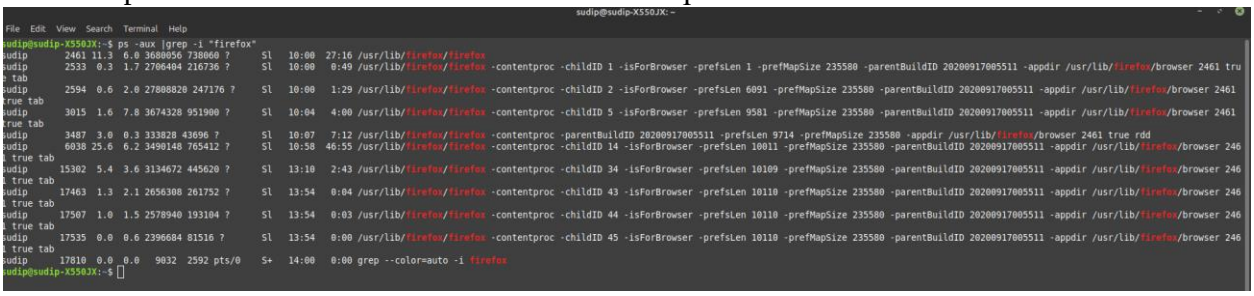


Used the watch utility with the time interval of 2 seconds



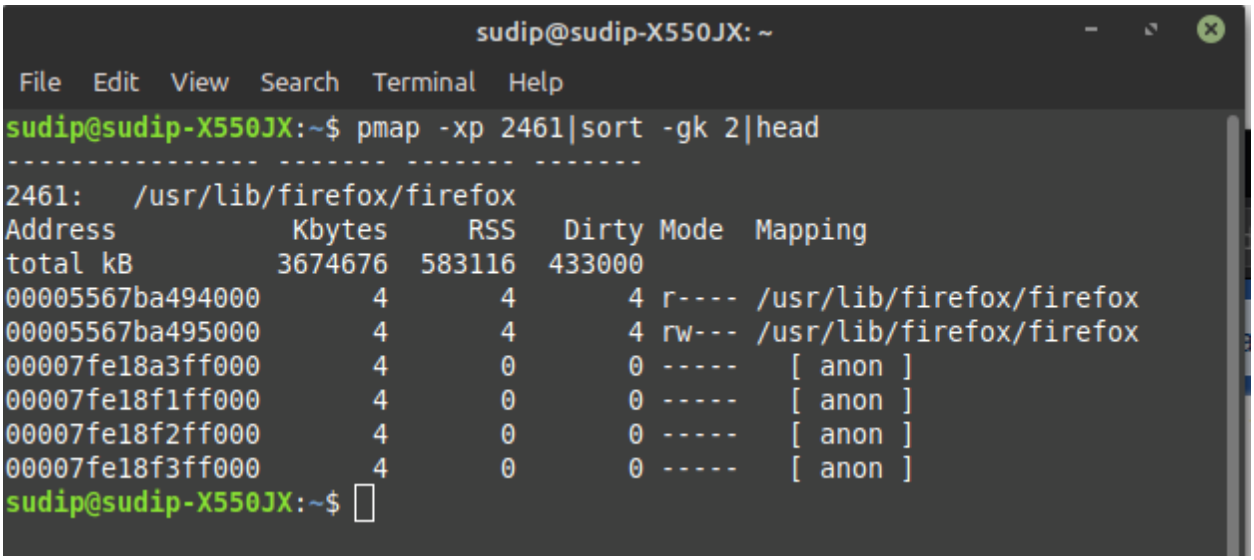


Used the ps -aux and found out the details about the process named “firefox”

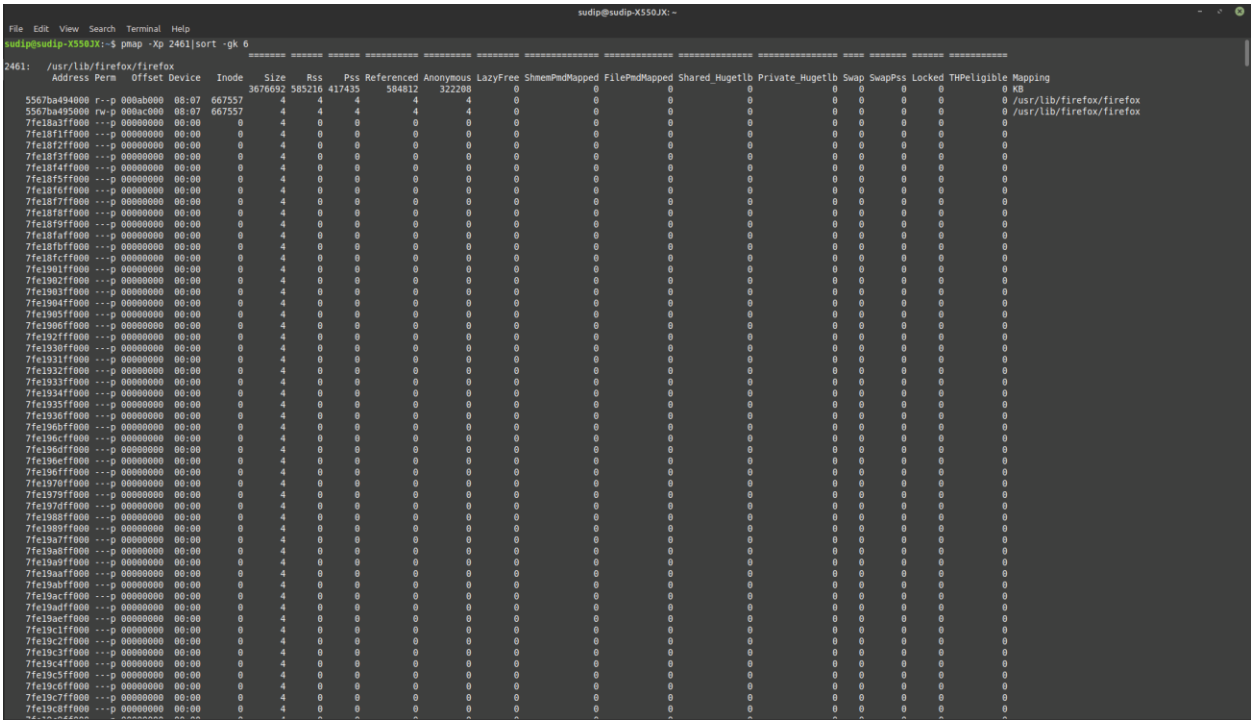


b) **The pmap command** - This command reports the memory map of the processes

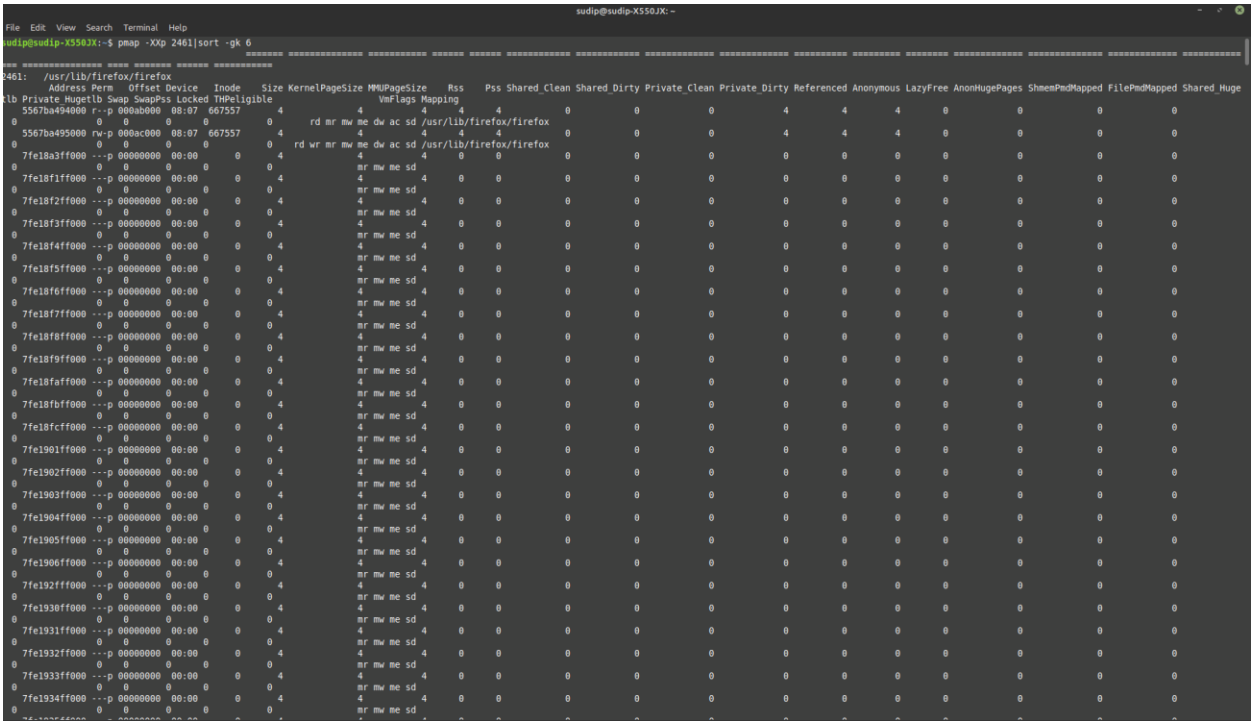
Used the pmap command to find out the memory map of the firefox process. Here the one of the PID of firefox is 2461



Used the extended flag along with the path flag for showing the memory map of the firefox process



Used the -XX flag which provide all the details that the kernel provides about firefox process

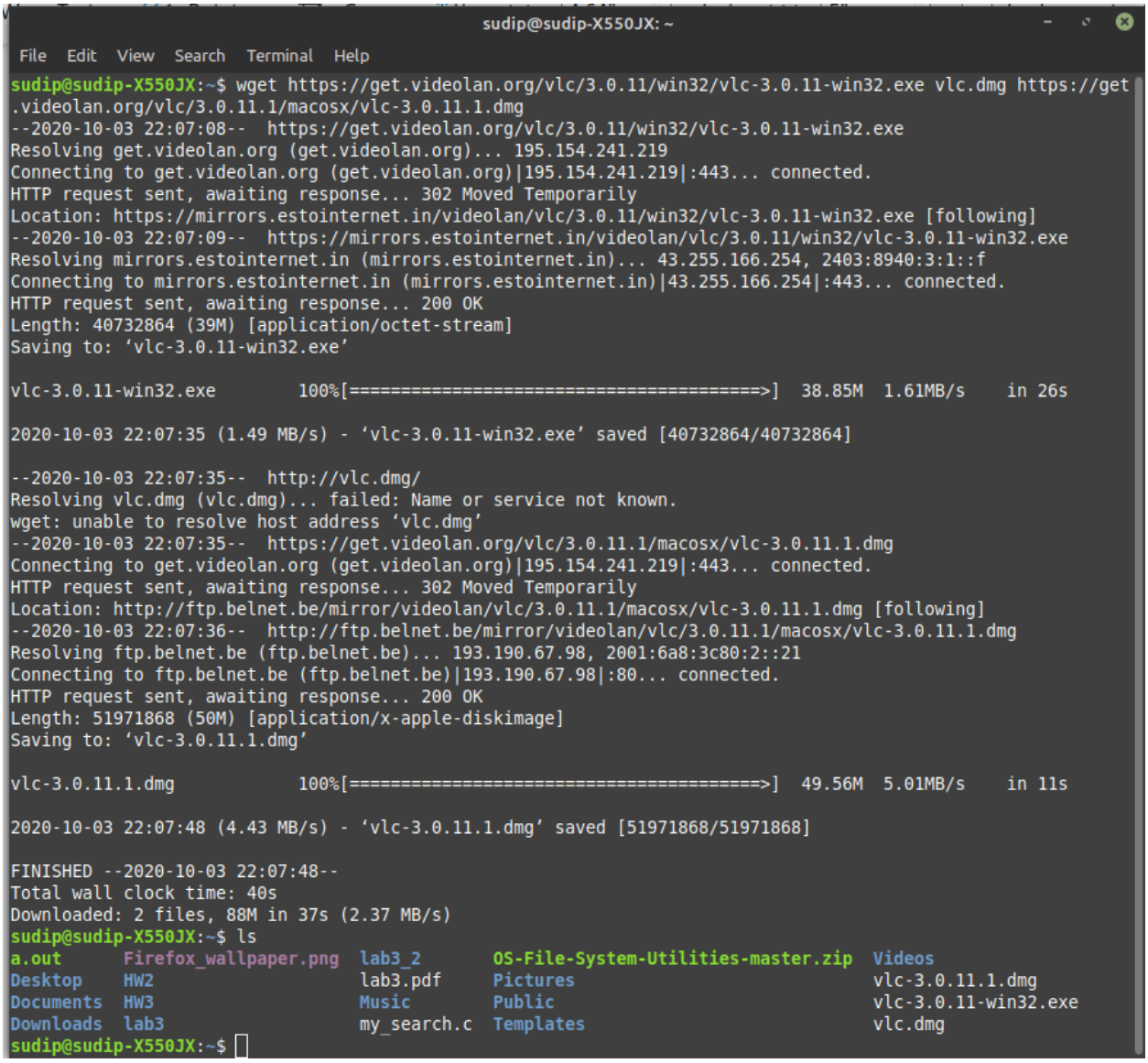


c) **The wget command** – It is a non-interactive network downloader.

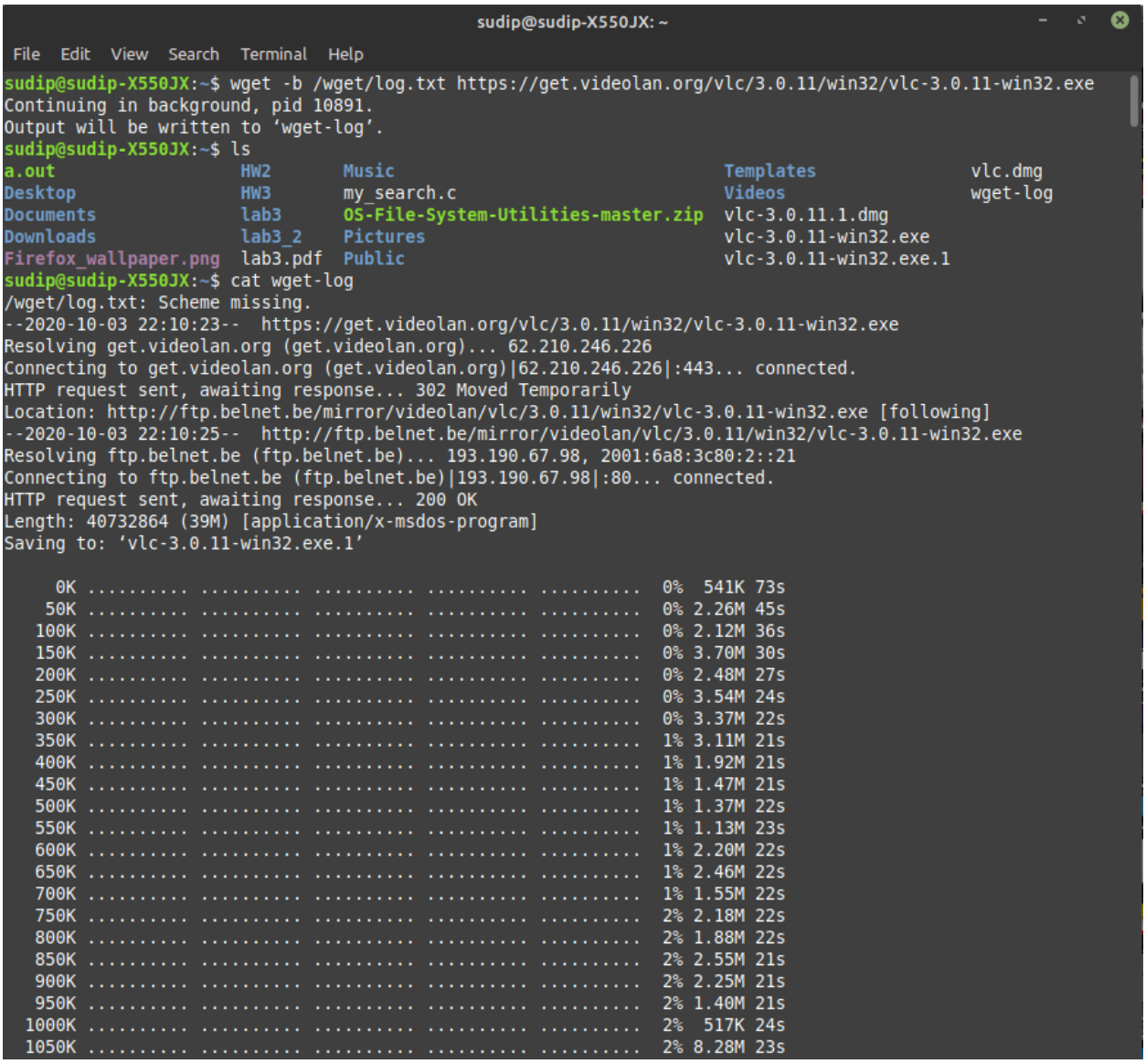
I used wget to download vlc media player from their site

Here we can see that that a executable file vlc.exe has been downloaded

I used wget to download 2 files, vlc media player of both mac and windows versions. We see that 2 files were created with the name vlc and having the extensions dmg and exe

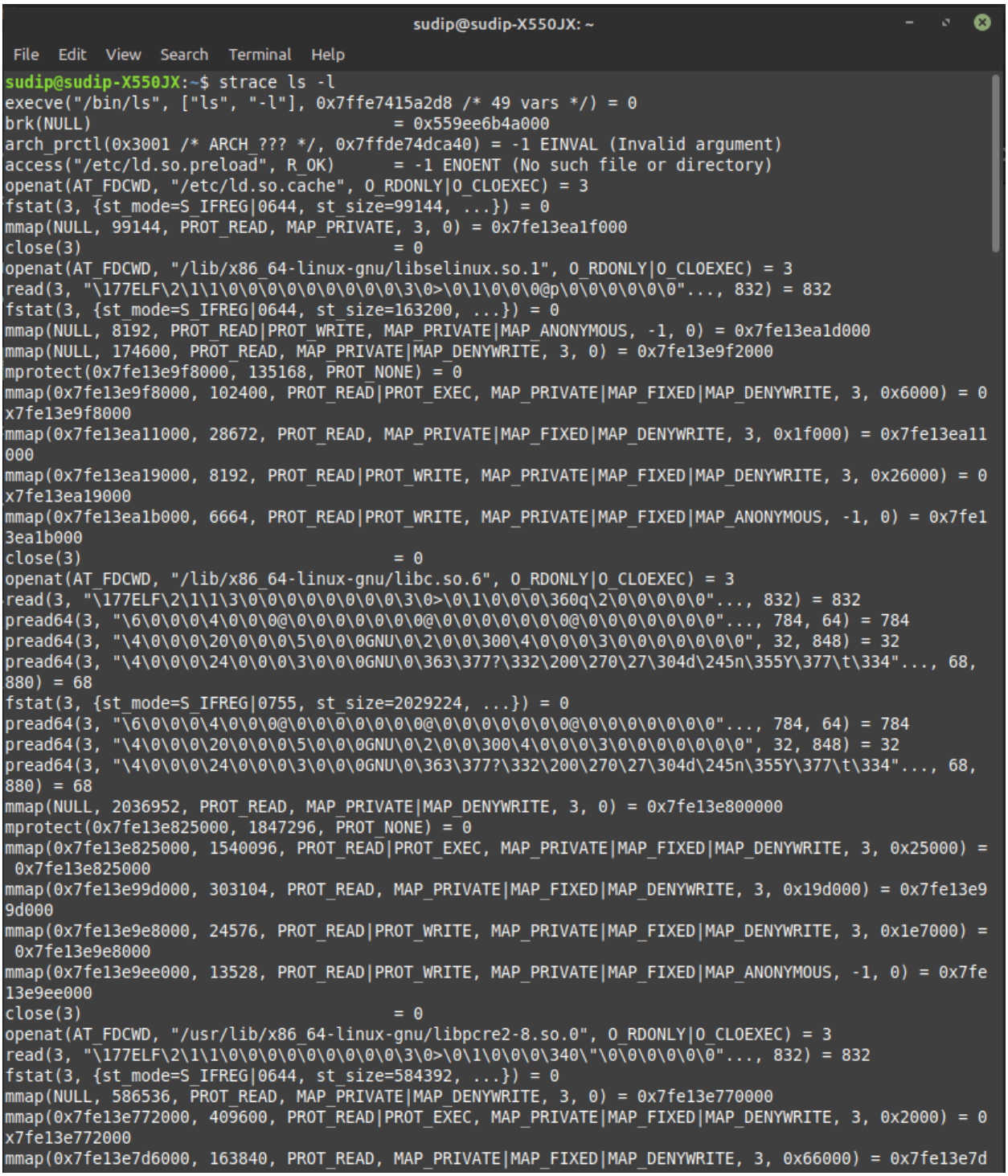


I used wget using the background flag. A log was created after the download has been completed.

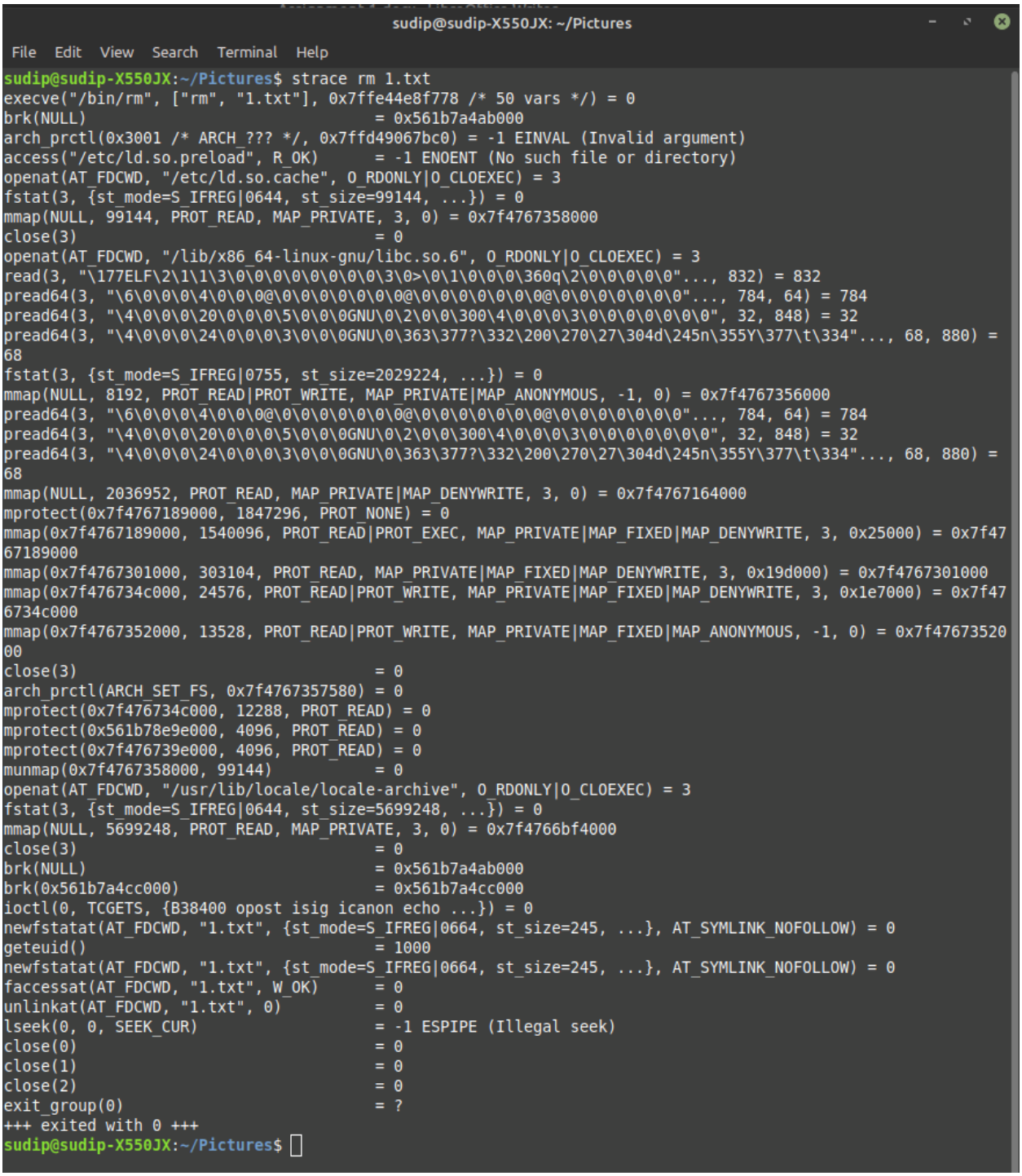


d) **The strace command** - This command traces system calls and signals

I called strace command on ls -l. We can see in the below figure some of the systems calls that ls -l executes. At first execve() is called, then some dependency files are opened then memory is allocated etc.

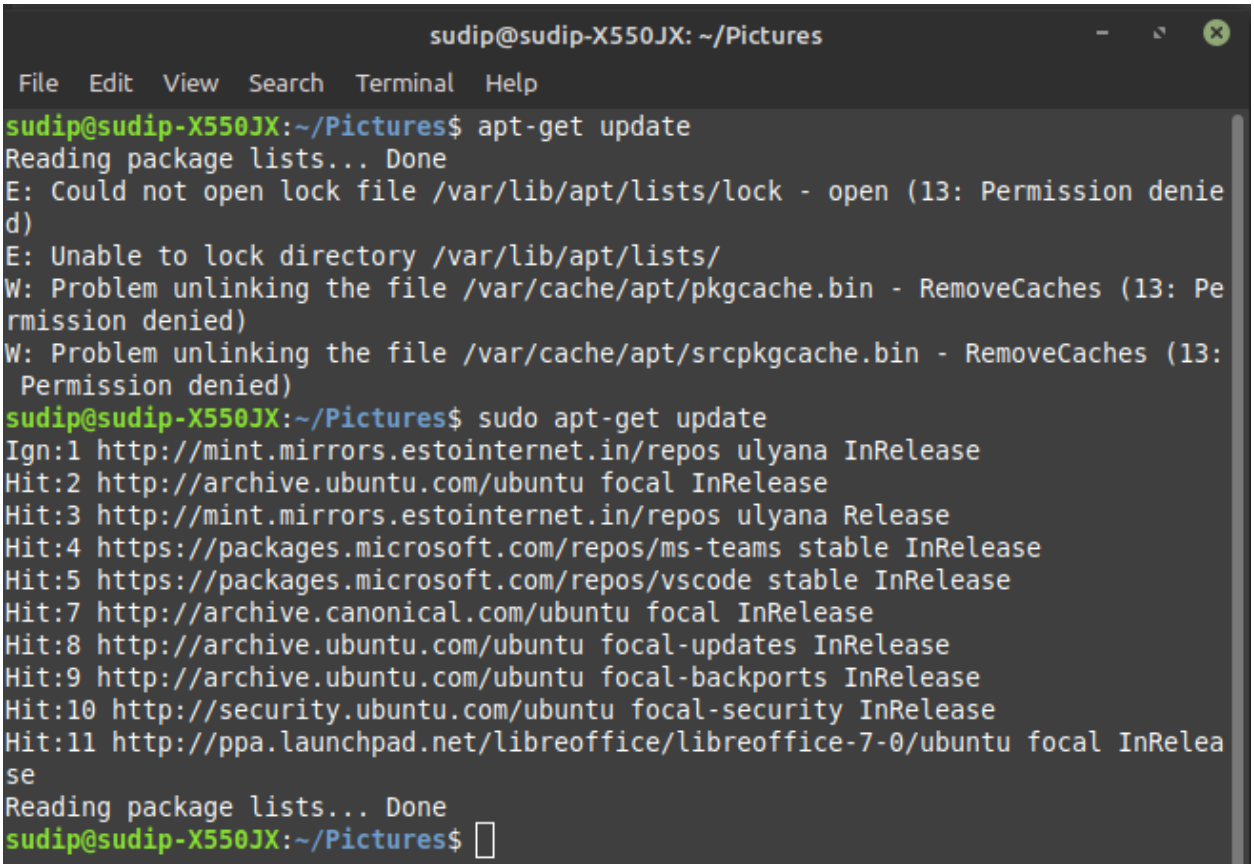


Similarly when we execute strace() on rm command, we can see at first execve() is executed at the beginning, then dependencies are opened, then memory is allocated, after deleting the link of the file is deleted. At last all the dependencies are closed.

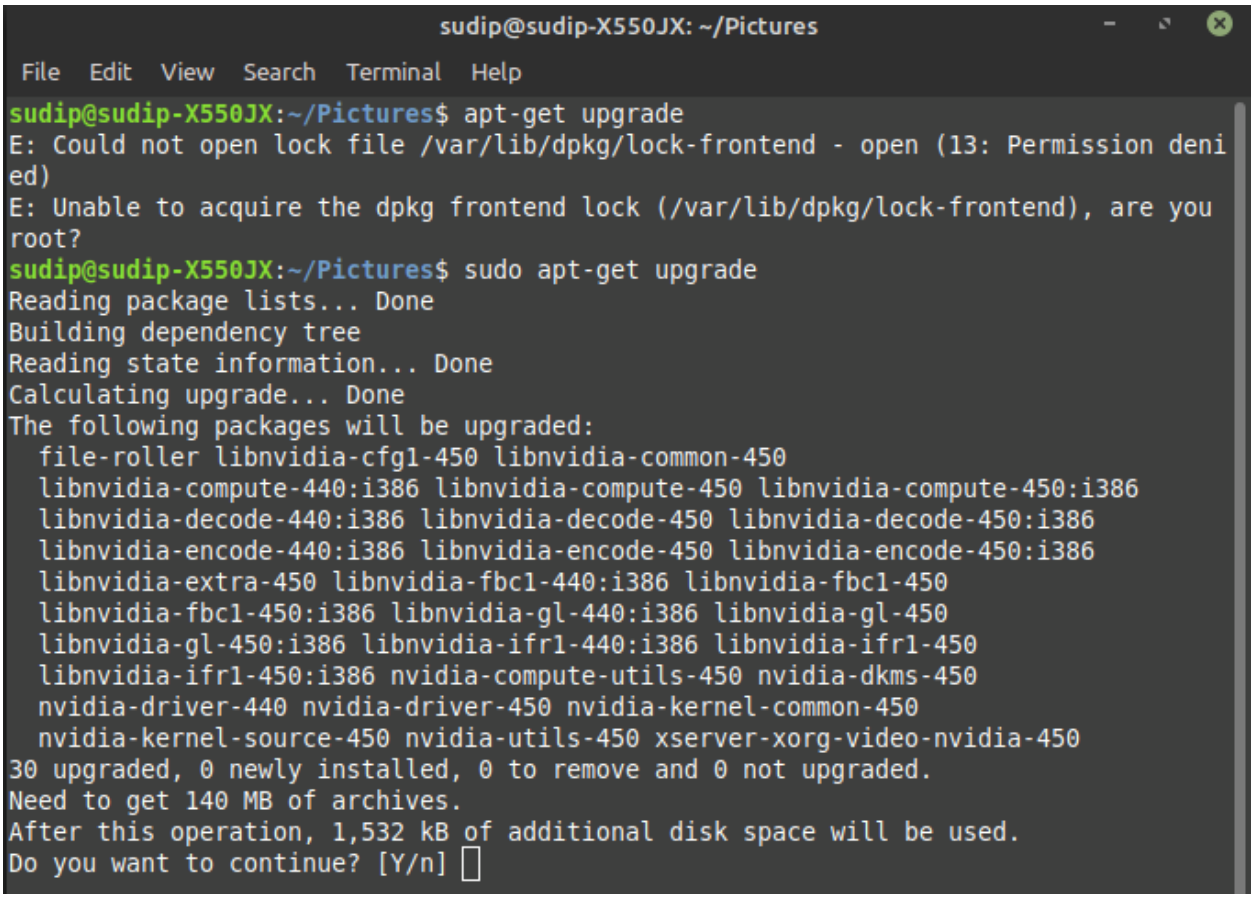


e) **The sudo command** – It executes a command as another user. It stands for super user do When a process can't execute due to lack of permissions, we can use sudo along with the system password to make the process execute seamlessly.

Here I have updated the list of latest versions of packages that can be used for updating system processes



After updating the latest versions, if we want we can update all the packages to the latest versions using apt-get upgrade command. Since updating packages is an important process, it can't be done without super user privileges.



Part 2 - Implementation of commands

Github repository link : https://github.com/David-jon/OS-Assignment-1-IITGN

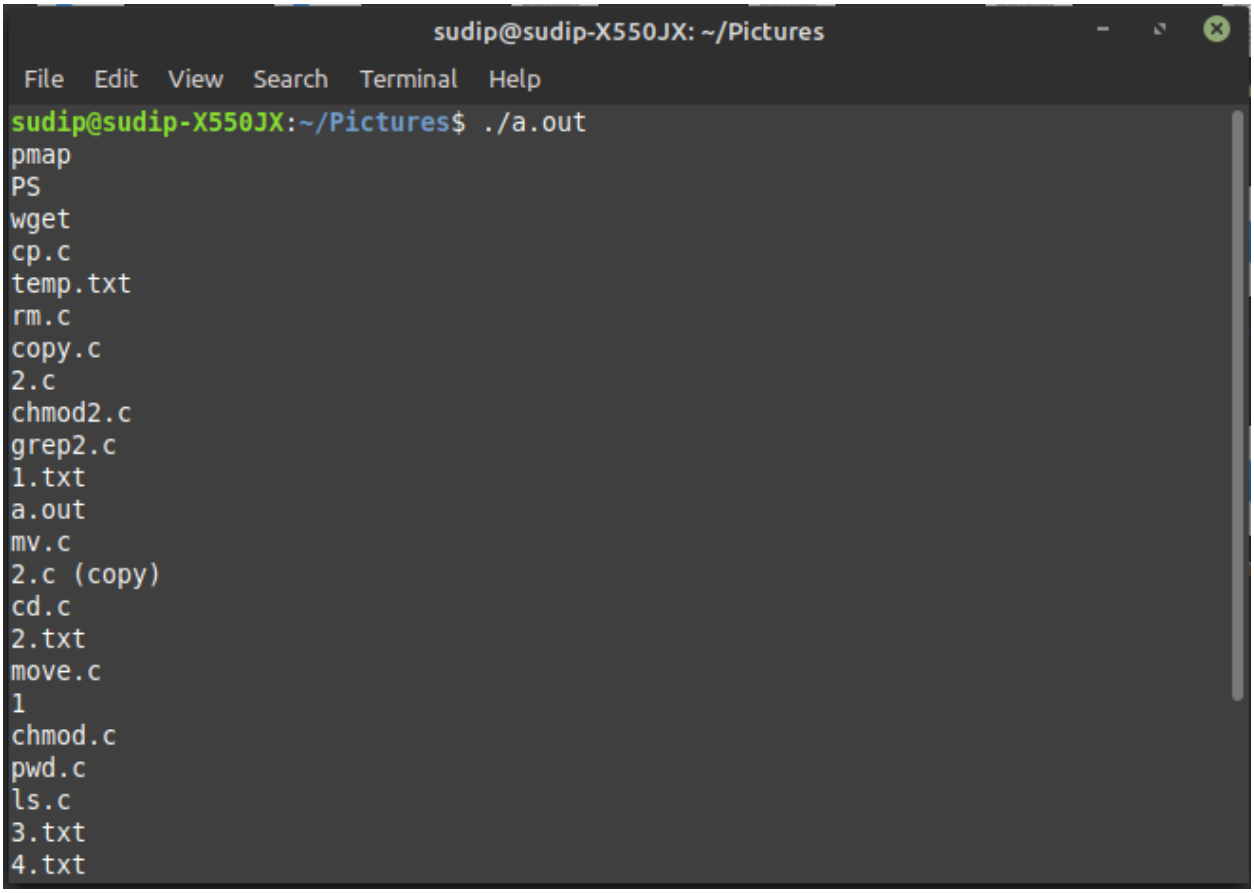
a. **ls command**

Source Code:

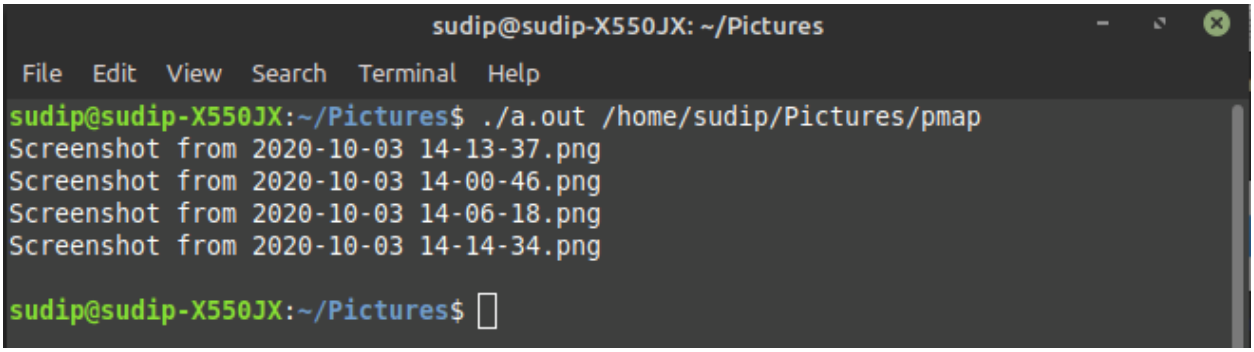
```
#include <fcntl.h>
#include <dirent.h>
#include <stdio.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
int main(int argc, char **argv)
{  //store the current directory value
   DIR *dir = opendir(".");
   if (argc == 2)
      dir = opendir(argv[1]);
   struct dirent *strdir = readdir(dir);
   while (strdir != NULL)
   {
      if (strdir->d_name[0] != '.' && strdir->d_name[0] != '\0')
      {
         //print the contents of the current working directory one at a time
         printf("%s\n", strdir->d_name);
      }
      strdir = readdir(dir);
   }
   printf("\n");
```

```
    closedir(dir);
}
```

Using ls command without any arguments lists the contents of a directory.



Using ls command with argument as the location of the directory lists all the contents of the given directory.



b)**grep command**

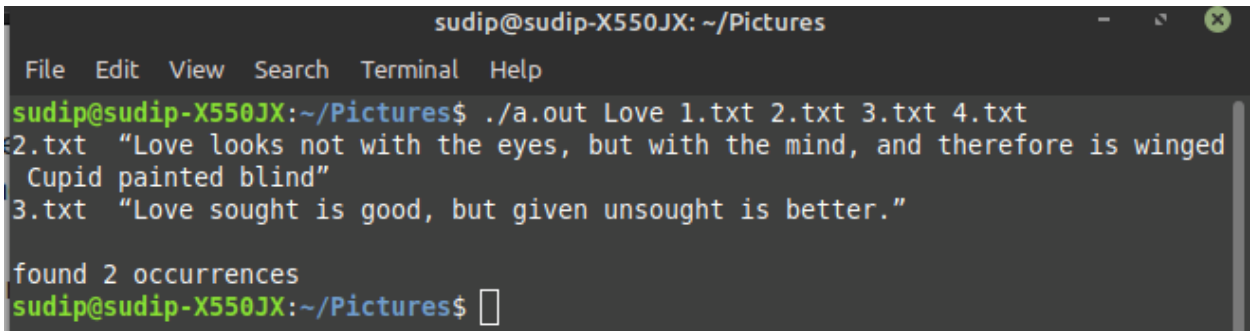
Source Code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
   char lineBuffer[512];
   FILE *fp = NULL;
   size_t n = 0;
   int flag = 0;
   char *buffer = NULL;
   int count = 0;
   //flag variable if 1 uses a temporary file temp.txt to already use the existing code for when argument count is greater than 2
   if (argc == 1)
   {
      printf("wgrep: searchterm [file ...]\n");
      exit(1);
   }
   if (argc == 2)
   {
      flag = 1;
      printf("Enter text\n");
      char str[2000];
      fgets(str, 2000, stdin);
      fp = fopen("temp.txt", "w");
      fputs(str, fp);
      fclose(fp);
```

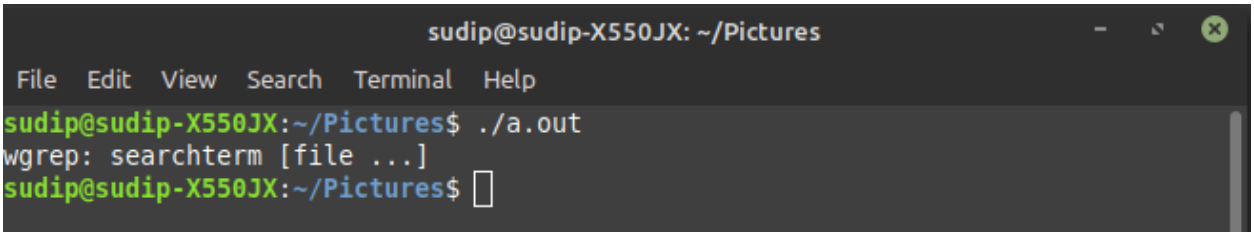
```
    //strcpy(argv[2],"temp.txt");
}
//for loop the grep for every argument vector value
for (int i = 2; i < argc || flag; i++)
{
    if (flag == 0)
    {
        fp = fopen(argv[i], "r");
        if (!fp)
        {
            printf("wgrep: cannot open file%s\n", argv[i]);
            exit(2);
        }
    }
    else
    {
        fp = fopen("temp.txt", "r");
    }
    // getline function for making the buffer size as dynamic as opposed to fgets()
    while (getline(&buffer, &n, fp) != -1)
    {
        //printf("What da 2\n");
        if (strstr(buffer, argv[1]))
        {
            if (flag == 0)
            {
                printf("\n%s  %s", argv[i], buffer);
            }
```

```
            else
                printf("\n%s ", buffer);
            ++count;
        }
    }
    if (flag == 1)
        flag = 0;
    fclose(fp);
}
printf("\nfound %d occurrences\n", count);
return 0;
}
```

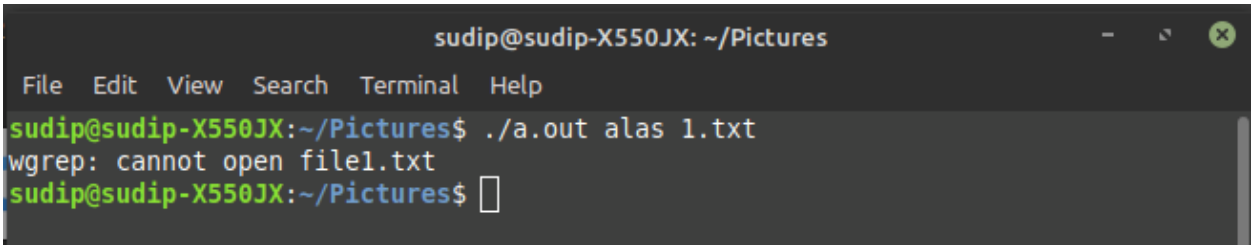
I have given four files text files each containing shakespeare texts and used the pattern “Love”.



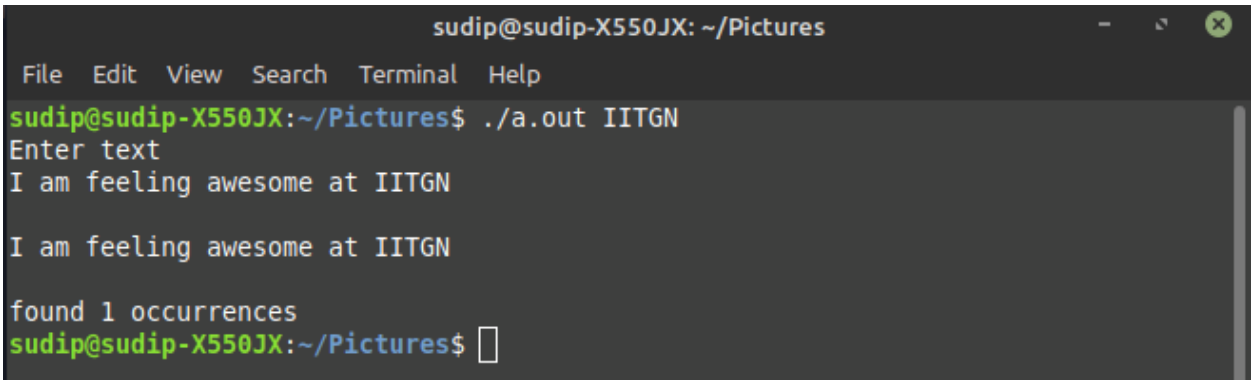
Running the program normally will give the function syntax



If the file can’t be opened, an appropriate error message will be displayed.



If we give only the search pattern, it will open a editor in which we can type anything and then we can search this text with the given pattern



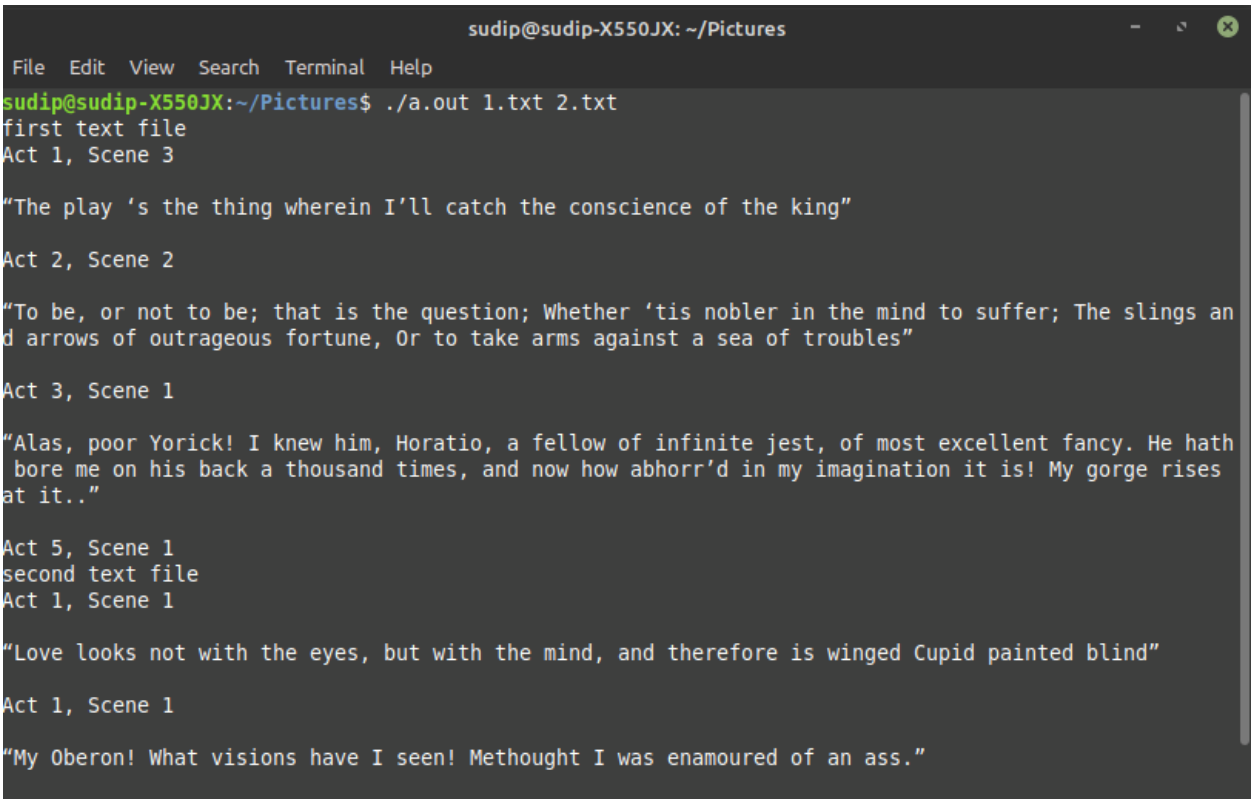
c)**cat command**

Source Code:

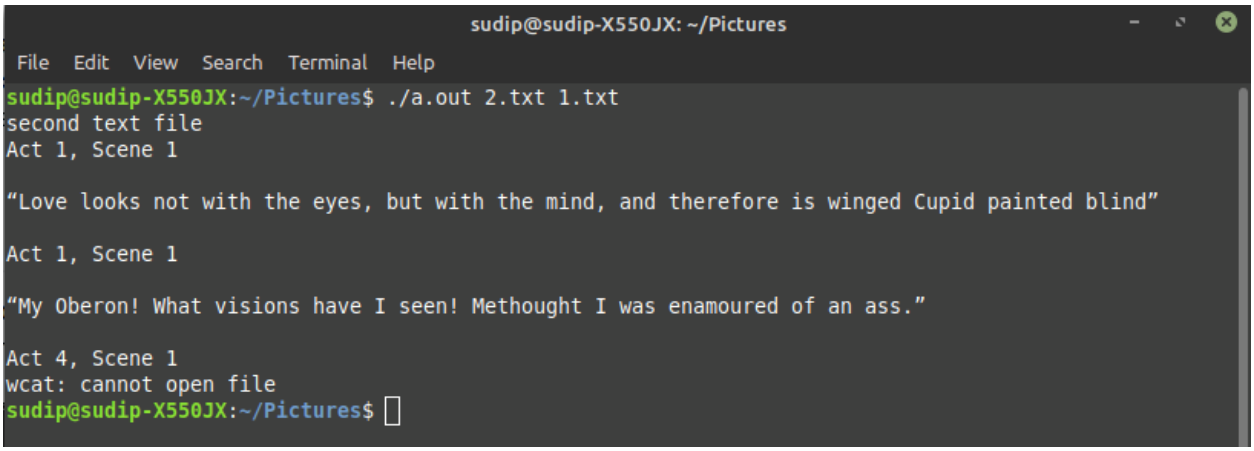
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char *argv[])
{
	int fp, op;
	char ch[1000];
	if (argc == 1)
		return 0;
	for (int i = 1; i < argc; i++)
	{
		fp = open(argv[i], O_RDONLY);
		//File can't be opened if returned value is -1
		if (fp == -1)
		{
			printf("wcat: cannot open file\n");
			exit(1);
		}
		//read every charecter and display on the screen
		while (op = read(fp, ch, 1000))
		{
			write(1, ch, op);
```

```
            }
            close(fp);
        }
        return 0;
}
```

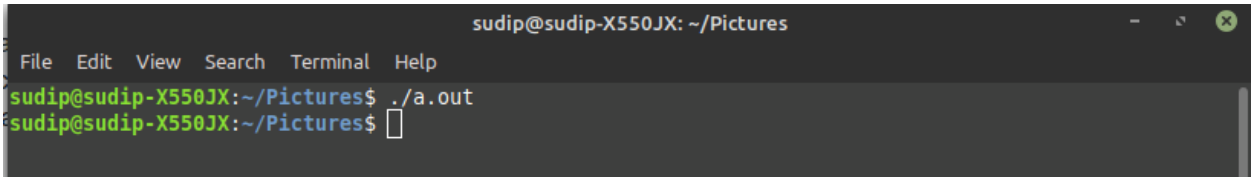
I have used my cat program to display two text files, in order.



I then changed permission of the 1.txt to have no permissions and then used the same command. The first file was displayed, where as for the second argument, the program displayed an appropriate error message and then exited



Running the program without giving any arguments produces no error as described in the requirements section.



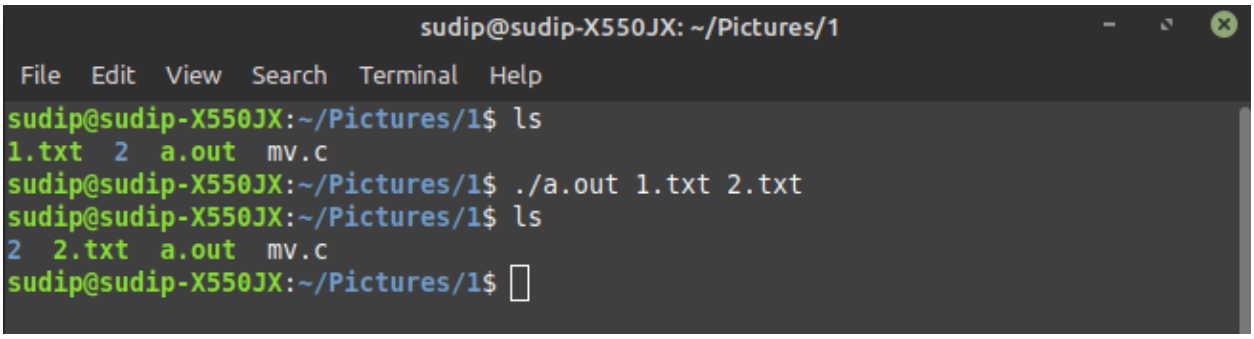
d**) mv command**

Source Code:

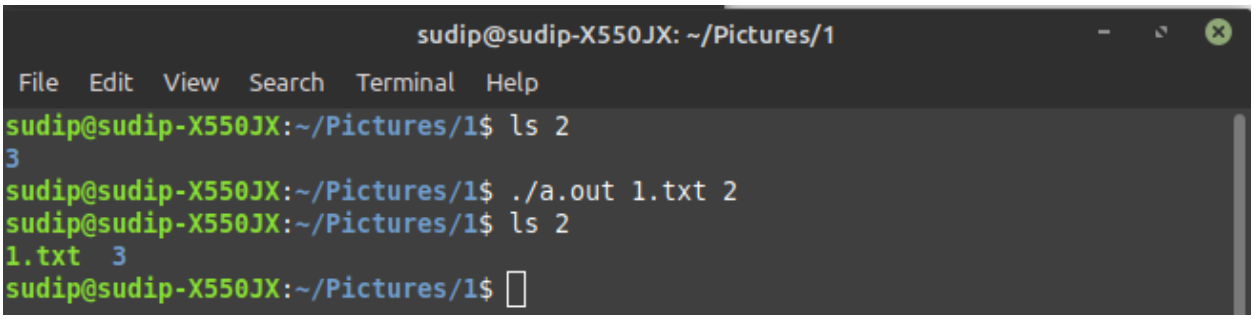
```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <unistd.h>
#include <string.h>
int main(int argc, char *argv[])
{
   char *file = argv[1];
   char *location = argv[2];
   char move_location[100];

   if (argc != 3)
   { //display the syntax if and when there is an error in the number og arguments
      printf("\nUsage: mv [source] [destination]\n");
      exit(EXIT_FAILURE);
   }
   else
   {
      DIR *dir;
      dir = opendir(location);
      if (dir == NULL)
      {
         //error if directory doesn't exist
         if (rename(file, location) != 0)
            printf("mv: cannot stat '%s': No such file or directory\n", argv[1]);
      }
```

```
        else
        {
            char *ptr;
            ptr = getcwd(move_location, 100);
            strcat(move_location, "/");
            strcat(move_location, location);
            strcat(move_location, "/");
            strcat(move_location, file);
            if (rename(file, ptr) == -1)
                printf("mv: cannot stat '%s': No such file or directory\n", argv[1]);
            closedir(dir);
        }
    }
    return 0;
}
```

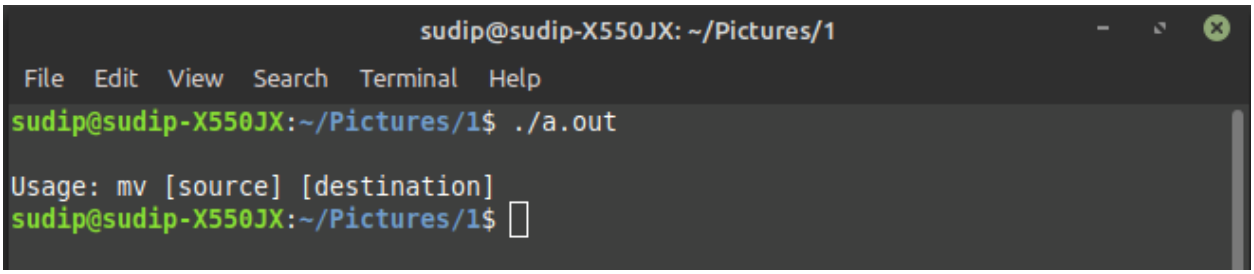
I used mv program to rename a file 1.txt to 2.txt



I then moved the file given as argument to a new directory also given as a argument. I displayed all the contents of the file using built in ls command to display before and after the program execution



When the program is run by giving an incomplete number of arguments, a syntax message is printed.



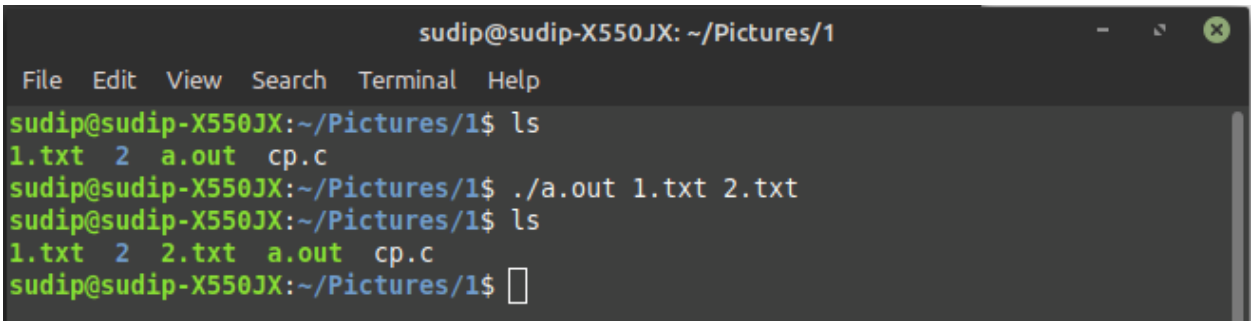
e) **cp command**

Source Code:

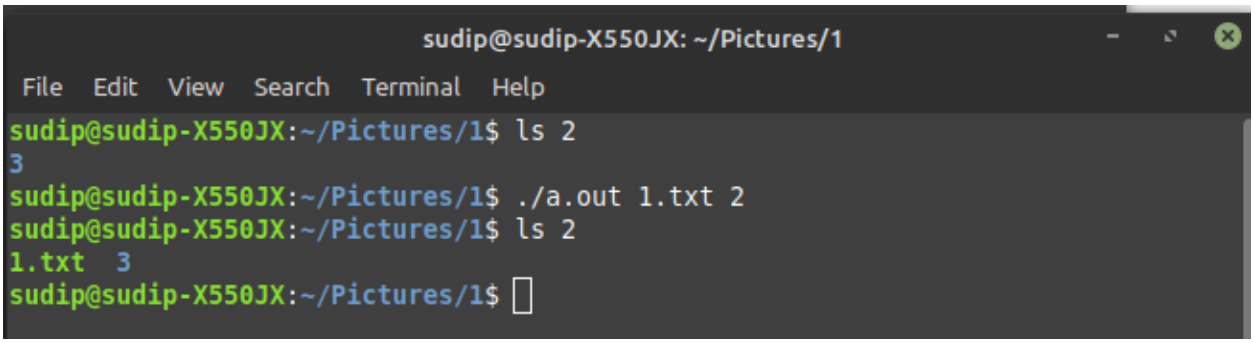
```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#define SIZE 1024

int main(int argc, char *argv[])
{
	int Source, Destination, ReadBuffer, WriteBuffer;
	char *buff[SIZE];
	//help syntax during errir
	if (argc != 3 || argv[1] == "--help")
	{
		printf("\nUsage: cp source destination\n");
		exit(EXIT_FAILURE);
	}
	Source = open(argv[1], O_RDONLY);

	if (Source == -1)
	{
		printf("\nError opening file %s errno = %d\n", argv[1], errno);
		exit(EXIT_FAILURE);
	}
	//making sure we have the permission to create a file there
	Destination = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
```

```
        if (Destination == -1)
        {
                printf("\nError opening file %s errno = %d\n", argv[2], errno);
                exit(EXIT_FAILURE);
        }

        while ((ReadBuffer = read(Source, buff, SIZE)) > 0)
        {
                if (write(Destination, buff, ReadBuffer) != ReadBuffer)
                        printf("\nError in writing data to \n");
        }

        if (close(Source) == -1)
                printf("\nError in closing file\n");

        if (close(Destination) == -1)
                printf("\nError in closing file\n");
}
```

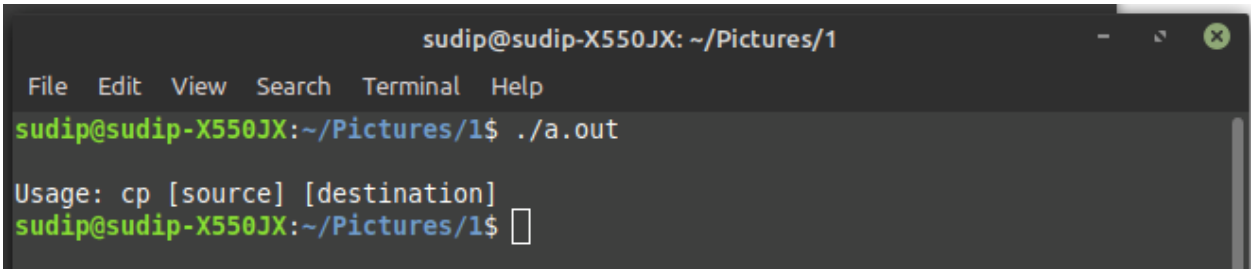
I have used the copy program to copy 1.txt and renamed the copied file to 2.txt. I then displayed all the contents of the file using built in ls command to display before and after the program execution



I then copied 1.txt to the directory 2. Then I displayed all the contents of the file using built in ls command to display before and after the program execution



When the program is run without any argument, it prints out the syntax of the input command



f) **cd command**

Source Code:

```
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  char *p = argv[1], dir[10000];
  if (argc != 2)
  {
    printf("Usage cd [Directory]\n");
    exit(EXIT_FAILURE);
  }
  //We are using chdir to change the directory
  else if (chdir(p) == -1)
  {
    perror("chdir");
    exit(EXIT_FAILURE);
  }
  else if (getcwd(dir, sizeof(dir)) != NULL)
    printf("Working dir: %s\n", dir);
  else
    perror("Error");

  /*
  The program changes the directory but the shell reverts back to theoriginal working directory.
```

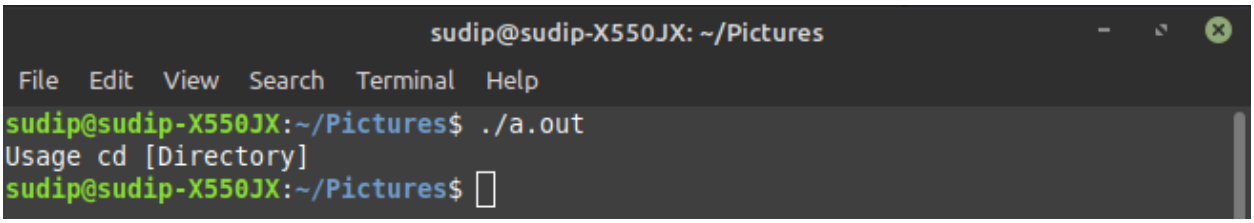
It is because when the program is executed in the shell, the shell follows fork on exec mechanism. So, it doesn't affect the current shell.

*/

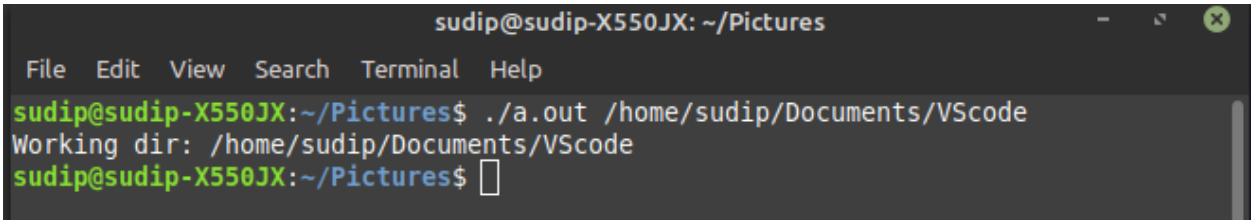
return 0;

}

If we are not giving any argument in the function a syntax message is displayed



I changed the directory from current directory to home/sudip/Documents/Vscode. The directory changed and it reverted back to the original directory. It is because when the program is executed in the shell, the shell follows fork on exec mechanism. So, it doesn't affect the current shell.

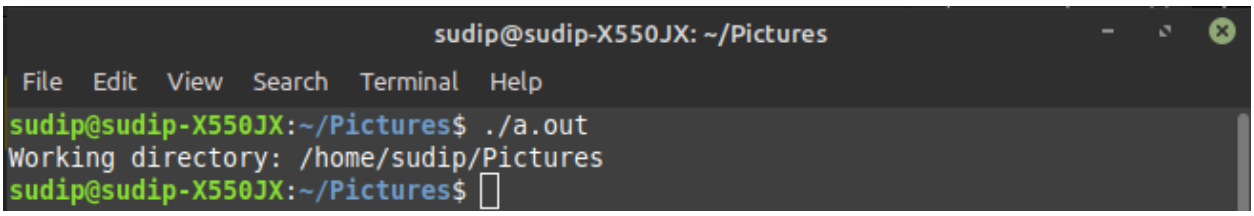


g) **pwd command**

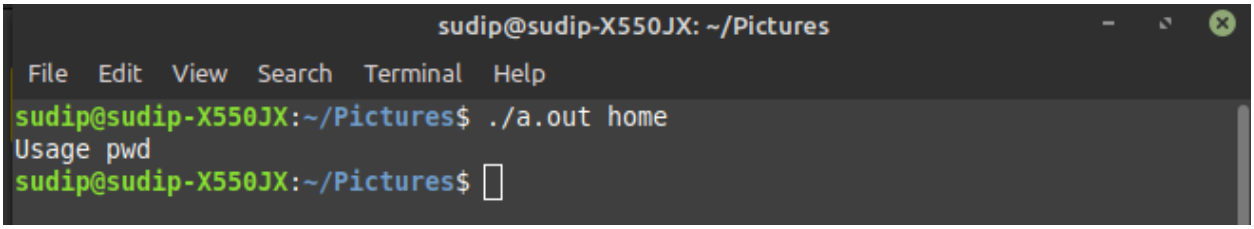
Source code:

```
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
void main(int argc)
{
  char *p = ".", dir[10000];
  if (argc != 1)
  {
    printf("Usage pwd\n");
    exit(EXIT_FAILURE);
  } //just print the current working directory
  else if (getcwd(dir, sizeof(dir)) != NULL)
    printf("Working directory: %s\n", dir);
}
```

Using pwd command prints the current working directory



if we use pwd command with arguments, it gives the usage of the command.

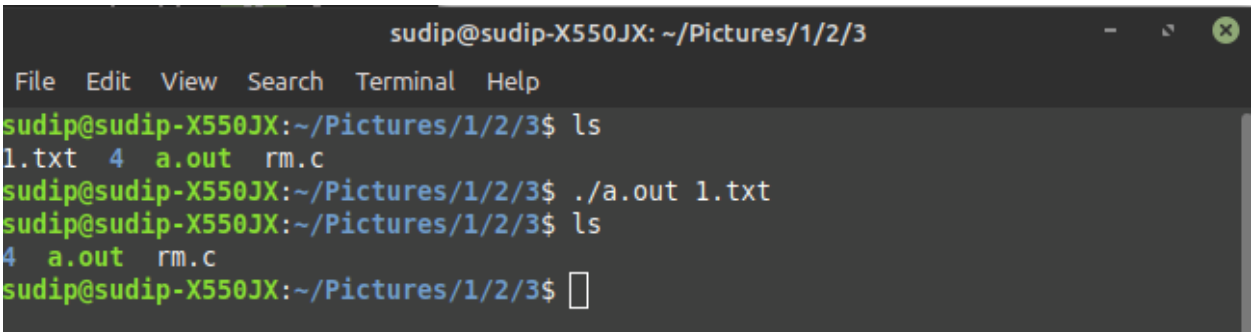


h) **rm command**

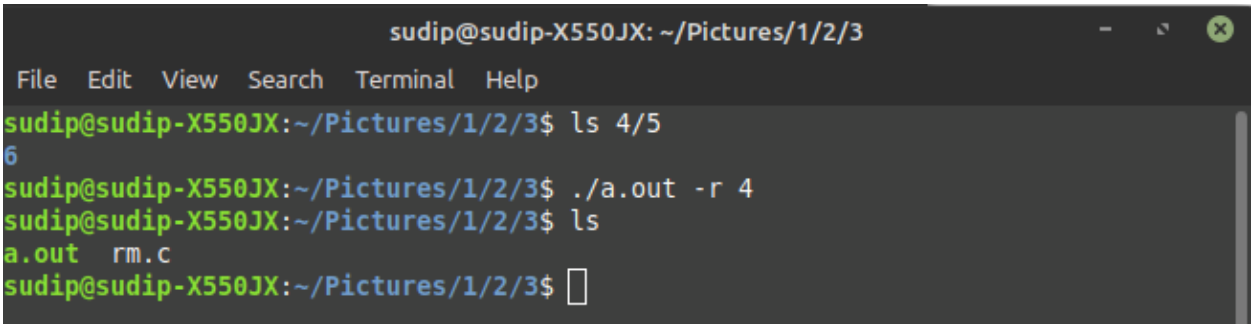
Source code:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main(int argc, char *argv[])
{
   //use exec function to recursively delete the files inside a non-empty directory
   if (argc == 3 && !(strcmp(argv[1], "-r")))
   {
      execlp("rm", "rm", "-r", argv[2], NULL);
      return 0;
   }
   if (argc != 2 || argv[1] == "--help")
   {
      printf("\nusage: rm [File Name] or rm -r [File Name]\n");
      return 0;
   }
   //use remove function if the directory is empty, ot when we are removing a file
   int status;
   status = remove(argv[1]);
   if (status != 0)
      printf("rm: cannot remove '%s': No such file or directory\n", argv[1]);
   return 0;
}
```

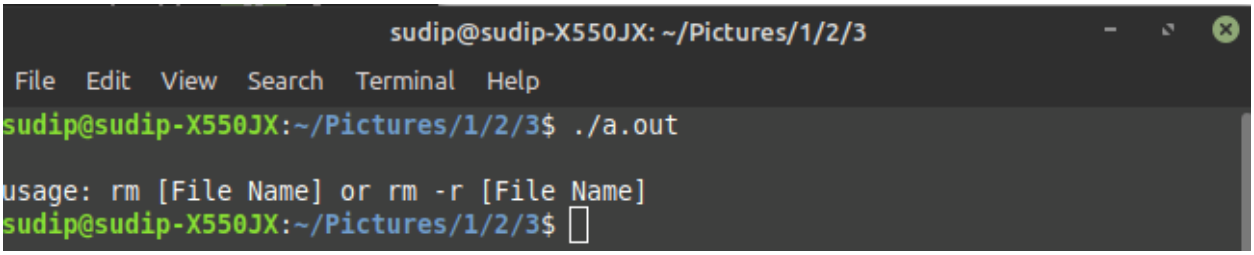
Using the remove program with argument as the file name removes the file name. It can also remove empty directory



Using the remove program with the -r flag will also remove directories which are not empty



If the user types incorrect number of arguments, the syntax of the input is displayed



i) **chmod command**

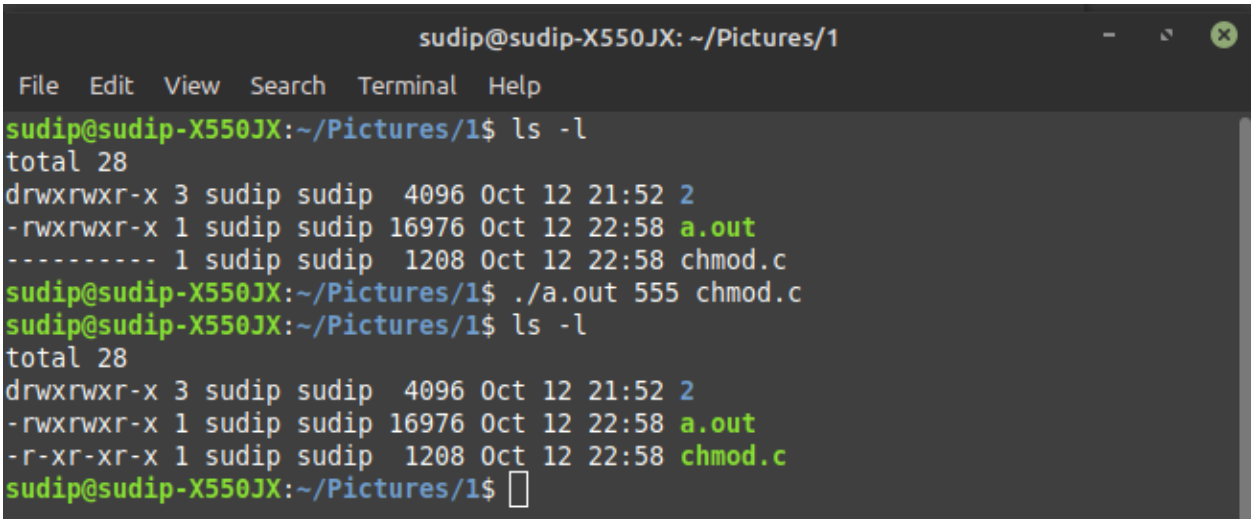
Source code:

```
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
	if (argc != 3)
	{
		printf("chmod [0-7][0-7][0-7] [Filename]\n");
		exit(1);
	}
	struct stat st;
	mode_t mode;
	const char *path = argv[2];
	int num = atoi(argv[1]);
	stat(path, &st);

	mode = st.st_mode & 07777;
	//first removing all the permissions of the current file
	mode &= ~(S_IRUSR);
	mode &= ~(S_IWUSR);
	mode &= ~(S_IXUSR);
	mode &= ~(S_IRGRP);
	mode &= ~(S_IWGRP);
	mode &= ~(S_IXGRP);
	mode &= ~(S_IROTH);
	mode &= ~(S_IWOTH);
```

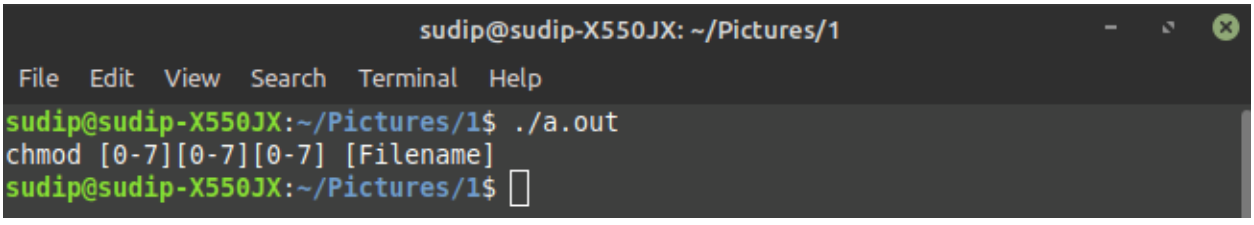
```
        mode &= ~(S_IXOTH);
        chmod(path, mode);
        //dividing the input number into three categories
        int others = num % 10;
        num /= 10;
        int group = num % 10;
        num /= 10;
        int user = num % 10;
        //printf("user %d,group %d,others %d argc %d num %d\n", user, group, others, argc,
num);
        //setting the permissions using bitwise operators
        if (user & 4)
                mode |= S_IRUSR;

        if (user & 2)
                mode |= S_IWUSR;

        if (user & 1)
                mode |= S_IXUSR;

        if (user & 4)
                mode |= S_IRGRP;

        if (user & 2)
                mode |= S_IWGRP;

        if (user & 1)
                mode |= S_IXGRP;
```

```
        if (user & 4)
                mode |= S_IROTH;

        if (user & 2)
                mode |= S_IWOTH;

        if (user & 1)
                mode |= S_IXOTH;
        chmod(path, mode);
        return 0;
}
```

I changed the permissions from 000 of chmod.c to 777. I displayed the permissions of the files using ls -l command.



If the number of arguments is incorrect, the proper syntax is displayed. This program only takes input as three digit octal integer for setting the file permissions

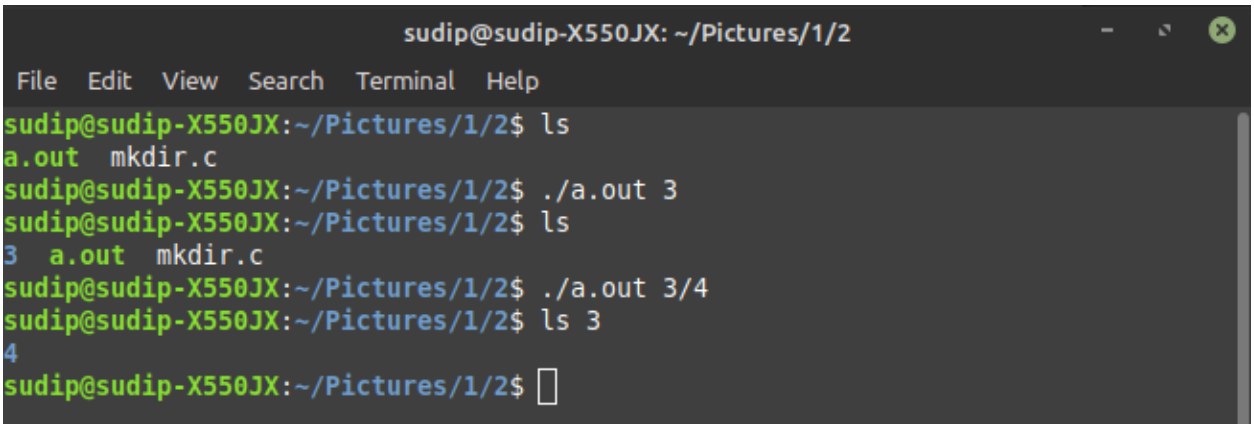


j) **mkdir command**

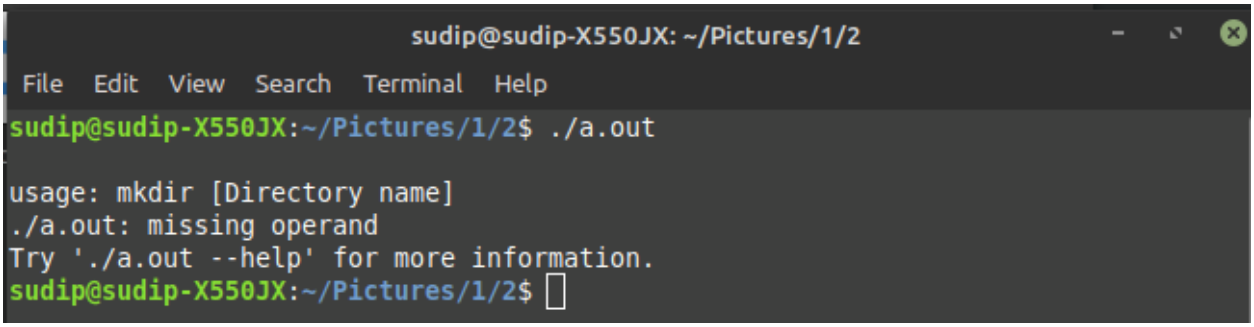
Source code:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
void main(int argc, char *argv[])
{

  if (argc != 2 || argv[1] == "--help")
  {
    printf("\nusage: mkdir [Directory name]\n");
    // break;
  }
  char *cmd = "mkdir";
  char *args[3];
  args[0] = "mkdir";
  args[1] = argv[1];
  args[2] = NULL;
  //using execvp to execute the mkdir command. We are passin the arguments that are taken from the command line
  execvp(cmd, argv);
}
```

I created a directory using mkdir program. Then I created a sub-directory inside the directory which I just created. I displayed the changes that were made by the program using ls command

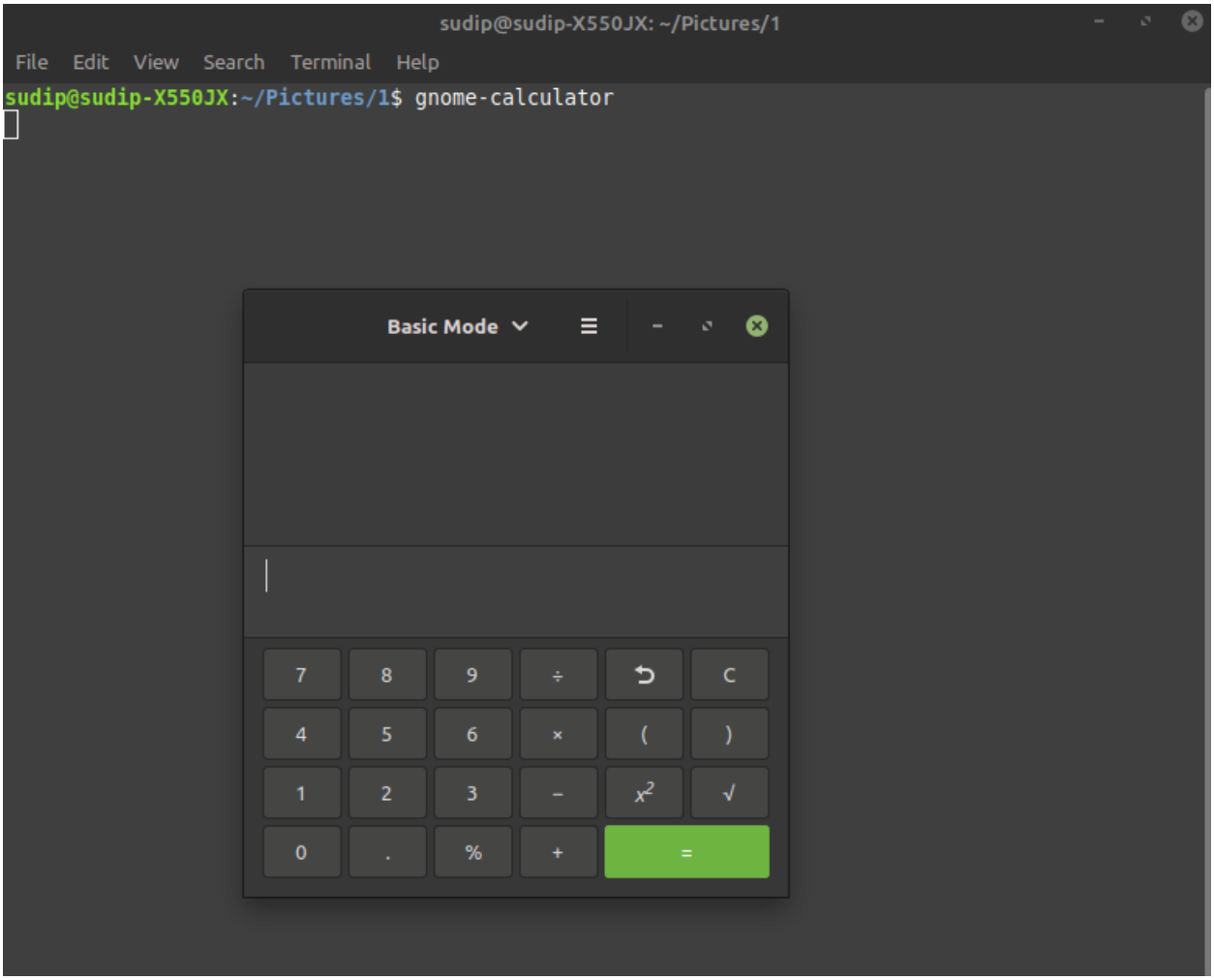


Running the program with incorrect number of arguments gives the syntax for correctly running the program

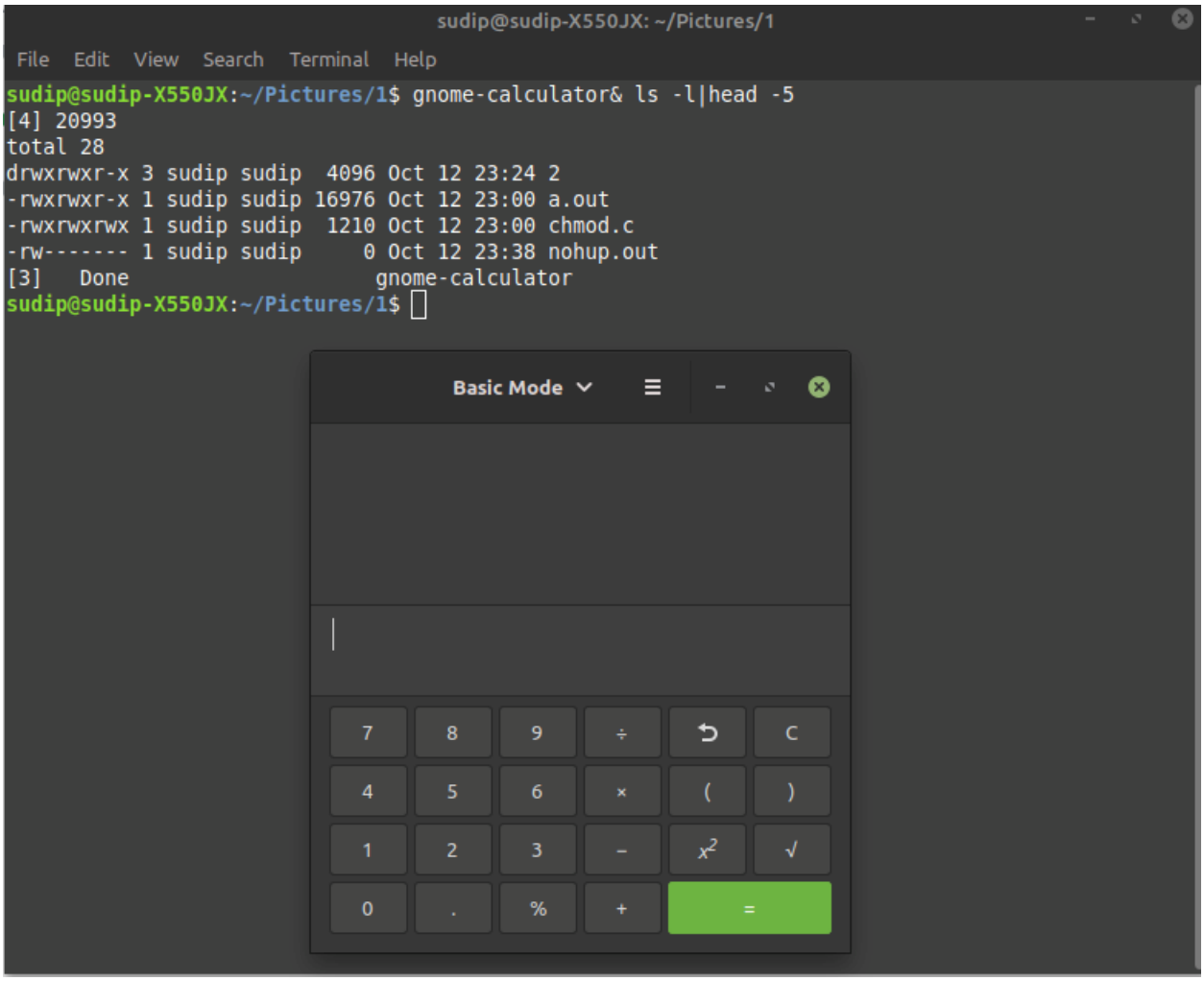


Part 3 - Run programs in background using & at the end

We can open the Gnome calculator using a simple command.



Using ampersand we can make the calculator as a background process, and we can still use the terminal. One disadvantage to using this method is, when we close the terminal, the calculator app also closes. To fix the above problem we use nohup command in the beginning of the sentence.



When we use nohup command, a nohup.out file is created which runs in the background. So even when we close the terminal, the calculator app still runs.