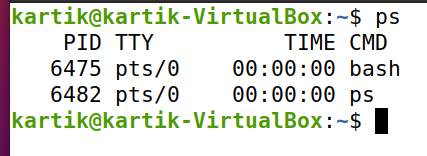
**INDEX**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Problem Description** | **Date of execution** |
| AP 1 | Run ps and note the PID of your shell. Log out and log in again and run ps again. What do you observe? |  |
| AP 2 | Enter the following commands, and note your observations: (i) who and tty, (ii) tput clear, (iii) id, (iv) ps and echo $$ |  |
| AP 3 | Run the following commands, and then invoke ls. What do you conclude? echo > README [Enter] echo > readme [Enter] |  |
| AP 4 | Create a directory, and change to that directory. Next, create another directory in the new directory, and then change to that directory too. Now, run $ cd without any arguments followed by pwd. What do you conclude? |  |
| AP 5 | Create a file mca containing the words “Hello MCA Class!”. Now create a directory bvicam, and then run mv mca bvicam. What do you observe when you run both ls and ls bar? |  |
| AP 6 | Run $ who am i and then interpret the output. |  |
| AP 7 | Find out whether the following commands are internal or external: echo, date, pwd, and ls. |  |
| AP 8 | Display the current date in the form dd/mm/yyyy. |  |
| AP 9 | Both of the following commands try to open the file mca, but the error messages are a little different. What could be the reason?  $ cat mca  cat: mca: No such file or directory  $ cat < mca  bash: mca: No such file or directory |  |
| AP 10 | Run the following commands, and discuss their output?  (a) $ uname  (b) $ passwd  (c) $ echo $SHELL  (d) $ man man  (e) $ which echo  (f) $ type echo  (g) $ whereis ls  (h) $ cd  (i) $ cd $HOME  (j) $ cd ~ |  |
| AP 11 | Frame ls command to (i) mark directories and executables separately, and (ii) also display hidden files. |  |
| AP 12 | Find out the result of following: $ cat mca mca mca |  |
| AP 13 | Run the following and determine which commands will work? Explain with reasons  . (a) $ mkdir a/b/  (b) $ mkdir a a/b  (c) $ rmdir a/b/c  (d) $ rmdir a a/b  (e) $ mkdir /bin/mca |  |
| AP 14 | How does the command mv mca1 mca2 behave, where both mca1 and mca2 are directories, when (i) mca2 exists, (ii) mca2 doesn‟t exist? |  |
| AP 15 | 5 Assuming that you are positioned in the directory /home/bvicam, what are these commands presumed to do, and explain whether they will work at all:  (a) $ cd ../..  (b) $ mkdir ../bin  (c) $ rmdir ..  (d) $ ls .. |  |
| BP 1 | Apply Peterson algorithm for solving the critical section problem with C/JAVA multi-threaded programming. Assume appropriate code snippets for the critical section. |  |
| BP 2 | Apply Bakery algorithm for synchronization of processes/threads in a C/Java program. Assume appropriate code snippet for critical section. |  |
| BP 3 | Write a C/Java program to simulate and solve the Producer-Consumer problem. |  |
| BP 4 | Implement Semaphore(s) in a C/Java-multithreaded program to simulate the working and solution of Reader-Writer problem. Assume multiple readers and writers. |  |
| BP 5 | Create a zombie process and an orphan process in a “C” program with appropriate system calls. |  |
| BP 6 | Write a “C” program which creates a new process and allows both, child and parent, to report their identification numbers (ids). The parent process should wait for the termination of the child process. |  |
| BP 7 | Write two “C” programs (A.c and B.c) where one program (A.c) creates a child process and then that child process executes the code of other program (B.c). The logic of program “B.c” is to generate all the prime numbers within the specified limit. |  |
| BP 8 | Write an appropriate “C” program which implements the concept of dynamic memory allocation (use of malloc(), calloc(), realloc(), and free() system call). |  |
| BP 9 | Create a text file, named as “courses.txt” that contains the following four lines:  Java Programming  Operating System  Discrete Structure  Write a “C” program that forks three other processes. After forking, the parent process goes into wait state and waits for the children to finish their execution. Each child process reads a line from the “course.txt” file (Child 1 Reads Line 1, Child 2 Reads Line 2, and Child 3 Reads Line 3) and each prints the respective line. The lines can be printed in any order. |  |
| CP 1 | Write an appropriate “C” program which implements the concept of dynamic memory allocation (use of malloc(), calloc(), realloc(), and free() system call). |  |
| CP 2 | Write a “C” program (using appropriate system calls of Linux) that generates “n” integers and stores them in a text file, named as “All.txt”. Then, retrieve the stored integers from this file and copy to “Odd.txt” and “Even.txt” based upon the type of number, i.e. if the retrieved integer if odd number then store in “Odd.txt” file or if the retrieved integer is even then store in Even.txt” file. Finally, display the contents of all three files on the screen. |  |
| CP 3 | Write a program in “C” which accepts the file or directory name and permission (access rights) from the user and then changes the access rights accordingly. Use appropriate system call(s) of Linux. |  |
| CP 4 | Write a “C” program (using appropriate system calls of Linux) which generates and stores the characters from „a‟ to „z‟. Then, display the stored characters in an alternative manner, like: a, c, e, g, …, etc. |  |
| CP 5 | Write a “C” program (using appropriate system calls of Linux) which receives roll number and names of “n” students, from the user one-by-one and then stores them in a text file, named as “Student.txt”. After inserting all “n” roll numbers and names, display the contents of the file. Also, display the access rights of the file “Student.txt”. |  |
| CP 6 | Demonstrate the use of following system calls by writing an appropriate “C” program.  (a) lseek()  (b) chmod()  (c) umask()  (d) access()  (e) utime() |  |

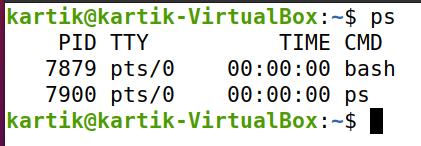
AP1:- Run ps and note the PID of your shell. Log out and log in again and run ps again. What do you observe?

OUTPUT:-

Ps = process status



Output after logging out and logging in the session.



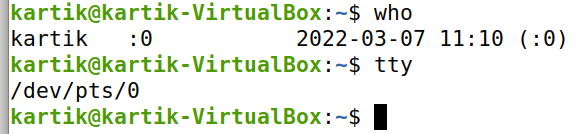
*PID* **–** The unique process ID.  
*TTY* **–** Terminal type that the user is logged into.  
*TIME* **–** Amount of CPU in minutes and seconds that the process has been running.  
*CMD* **–** Name of the command that launched the process.

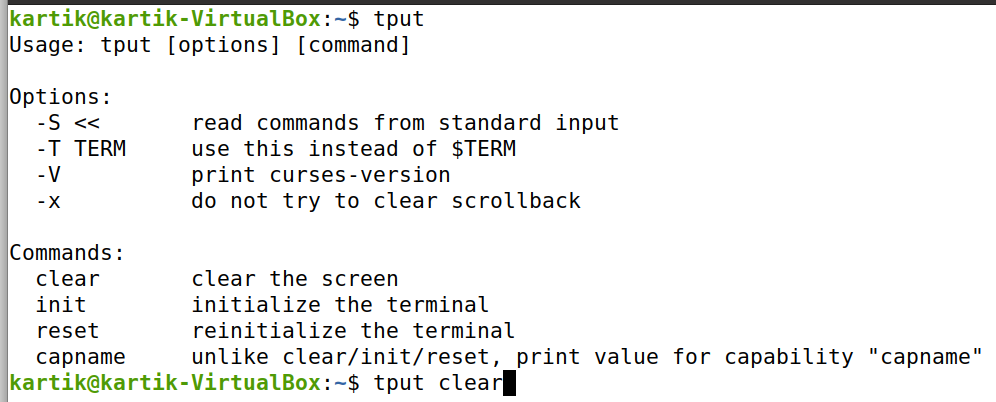
PID changes every time I open a new terminal session.

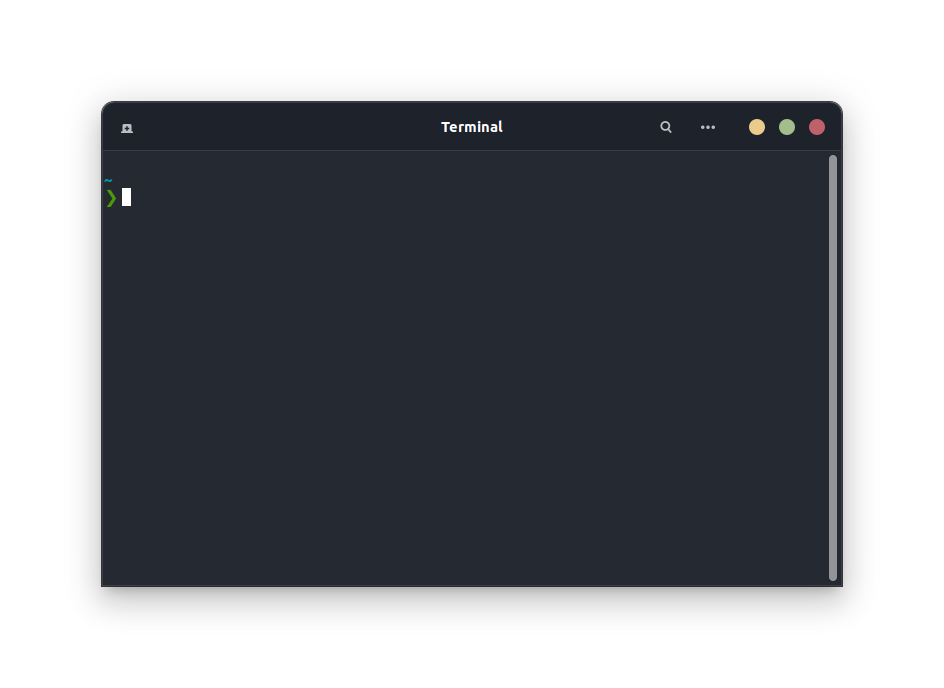
AP2:- Enter the following commands, and note your observations:

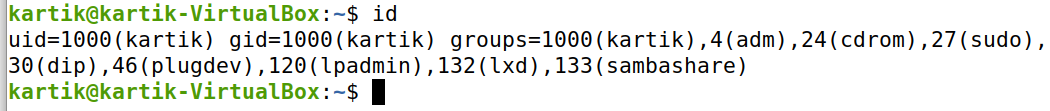
1. who and tty,
2. tput clear,
3. id,
4. ps and echo $$.

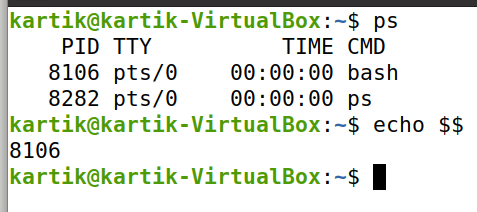
OUTPUT:-

1. who and tty
2. tput

****



1. id
2. ps and echo $$.

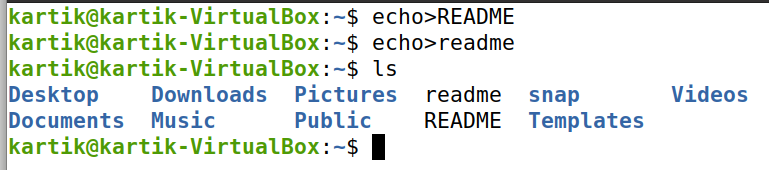
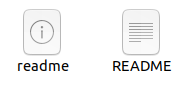
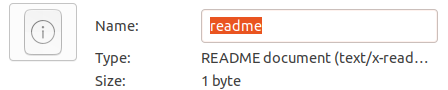
****

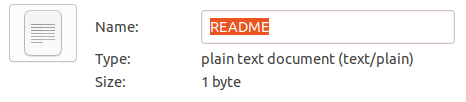
AP3:- Run the following commands, and then invoke ls. What do you conclude?

echo > README [Enter]

echo > readme [Enter]

Output:- Files in white and directories in blue.

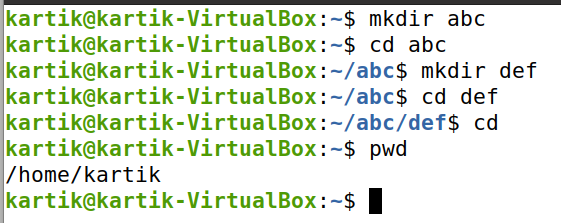
****

****

Echo command can also be used to create files. It creates two files one as readme document and one as normal text file, As their filename is same but change in type.

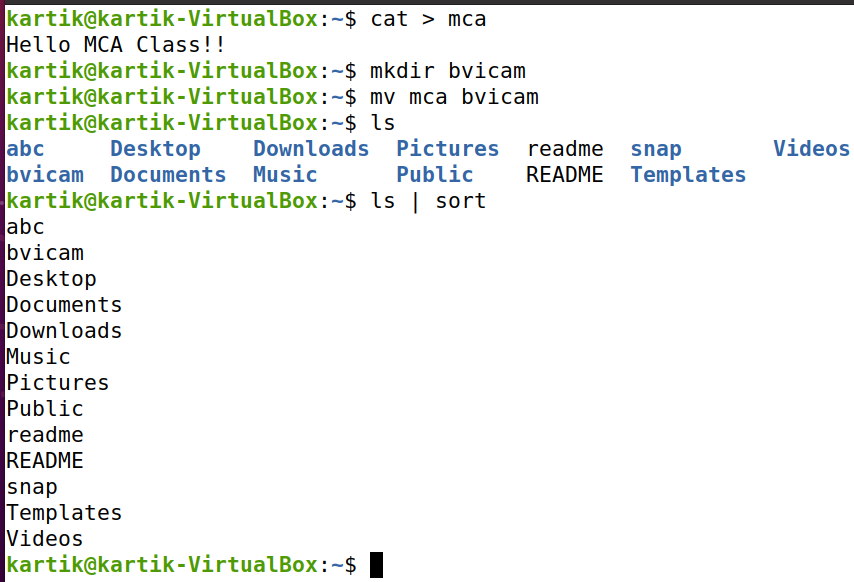
AP4:- Create a directory, and change to that directory. Next, create another directory in the new directory, and then change to that directory too. Now, run $ cd without any arguments followed by pwd. What do you conclude?

Output:-

****

cd command will take us to the initial position i.e. our home directory. When we use cd command it will take to your home directory and pwd shows your current working directory.

AP5:- Create a file mca containing the words “Hello MCA Class!”. Now create a directory bvicam, and then run mv mca bvicam. What do you observe when you run both ls and ls bar?

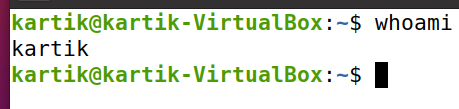


Output:-

mv command moved the mca file to bvicam directory**.**ls lists all the files in directory. If we use ‘ls bar’ which is used for piping but we only sorting the alphabetically.

AP6:- Run $ who am i and then interpret the output.

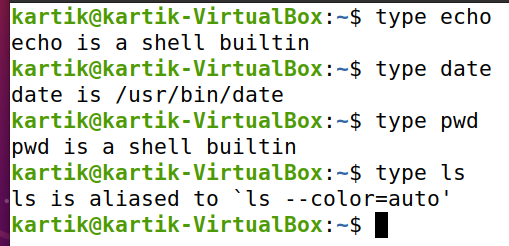
Output:-

****

It displays the username of current user.

AP7:- Find out whether the following commands are internal or external: echo, date, pwd, and ls.

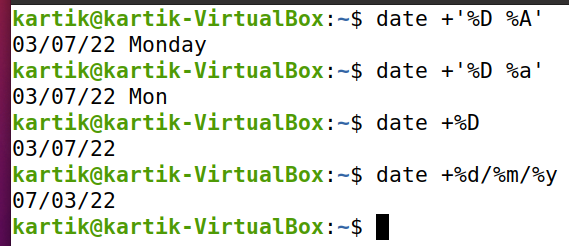
Output:-

****

* **Internal Commands :** Commands which are built into the shell. For all the shell built-in commands, execution of the same is fast in the sense that the shell doesn’t have to search the given path for them in the PATH variable, and also no process needs to be spawned for executing it.  
  Examples: source, cd, fg, etc.
* **External Commands :** Commands which aren’t built into the shell. When an external command has to be executed, the shell looks for its path given in the PATH variable, and also a new process has to be spawned and the command gets executed. They are usually located in /bin or /usr/bin. For example, when you execute the “cat” command, which usually is at /usr/bin, the executable /usr/bin/cat gets executed.  
  Examples: **ls**, cat etc.

AP8:- Display the current date in the form dd/mm/yyyy.

Output:-

****

AP9:- Both of the following commands try to open the file mca, but the error messages are a little different. What could be the reason?

$ cat mca

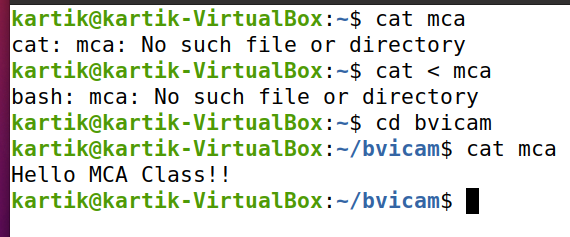
cat: mca: No such file or directory

$ cat < mca

bash: mca: No such file or directory.

Output:-“mca” file was moved to the directory bvicam so it can not be located in current directory(/home/kartik).

* In the first example we can see that cat opened the file and read from it, mca. In the second command we can see that cat reads the contents of the file mca via the STDIN file descriptor, identified as descriptor number 0.



AP10:- Run the following commands, and discuss their output?

(a) $ uname

(b) $ passwd

(c) $ echo $SHELL

(d) $ man man

(e) $ which echo

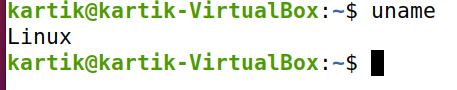
(f) $ type echo

(g) $ whereis ls

(h) $ cd

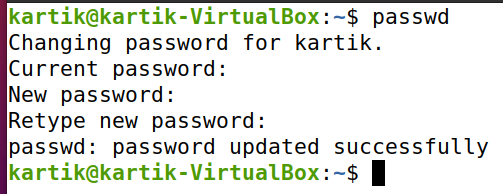
(i) $ cd $HOME

(j) $ cd ~

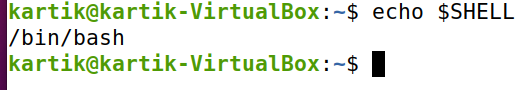
****

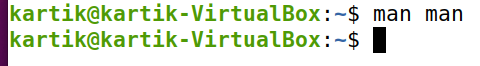
SOLN:-

1. $uname
2. $ passwd

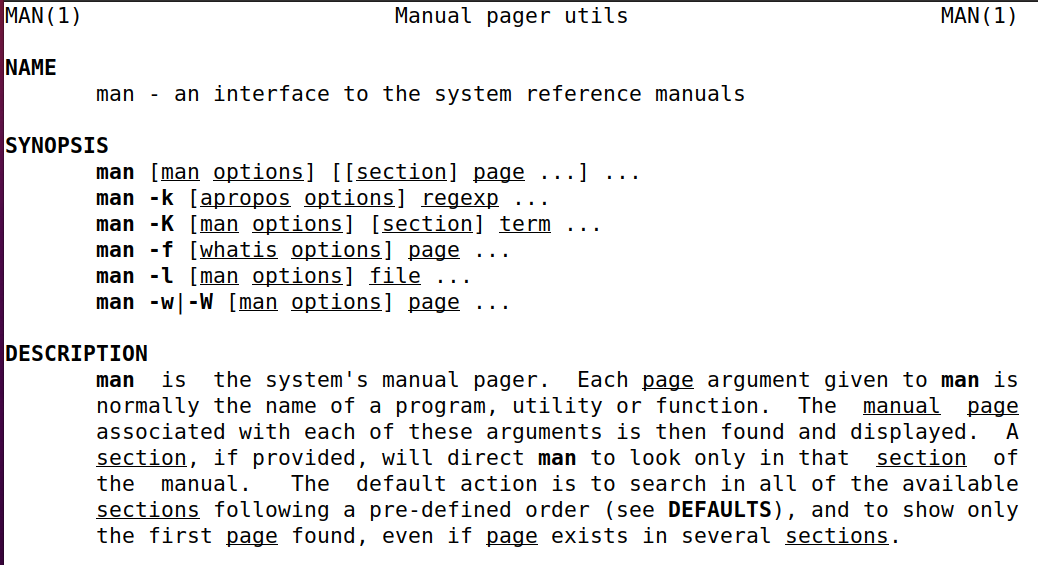


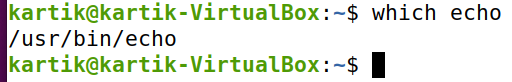
c) $ echo $SHELL : Represents current shell of the system.



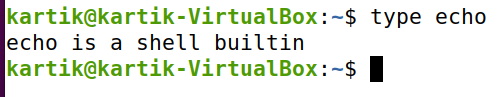


d) $ man man

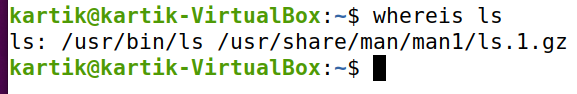




e) $ which echo

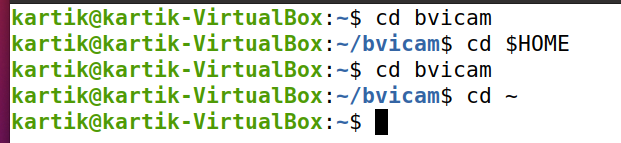


f) $ type echo



g) $ whereis ls

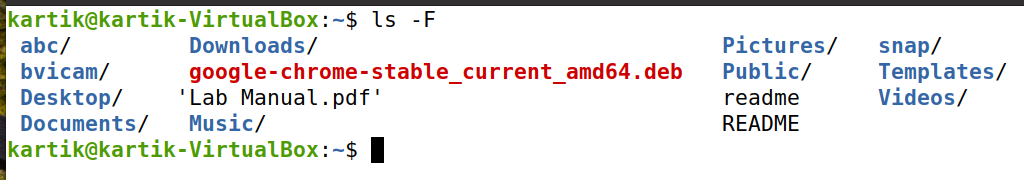
(h) $ cd (i) $ cd $HOME (j) $ cd ~



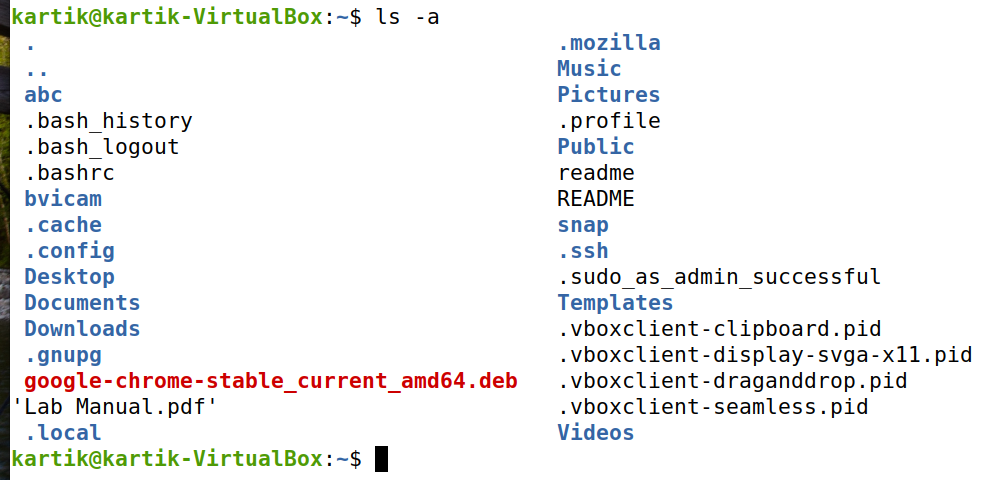
AP11:- Frame ls command to (i) mark directories and executables separately, and (ii) also display hidden files.

Output:-

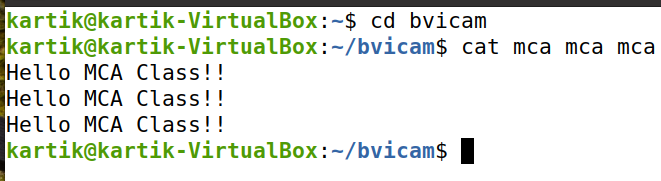
(i) mark directories and executables separately.

****

(ii) also display hidden files.

****

AP12:- Find out the result of following: $ cat mca mca mca



Output:-

AP13:- Run the following and determine which commands will work? Explain with reasons.

1. $ mkdir a/b/

* -> not working because directory “a” is not created before.

1. $ mkdir a a/b

->It will create directory “a” and subdirectory b.

(c) $ rmdir a/b/c

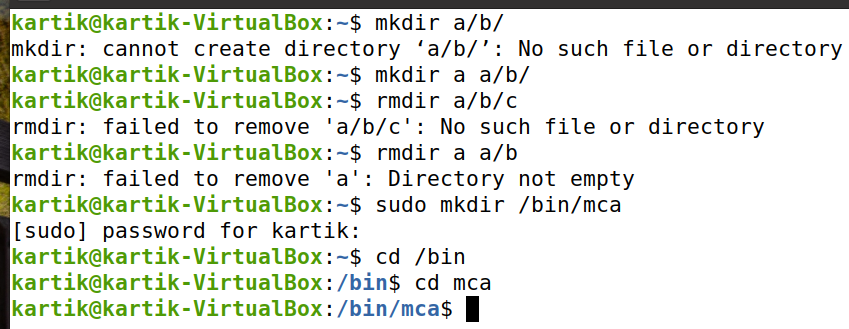
-> It will not remove c because there’s no such directory or file.

(d) $ rmdir a a/b

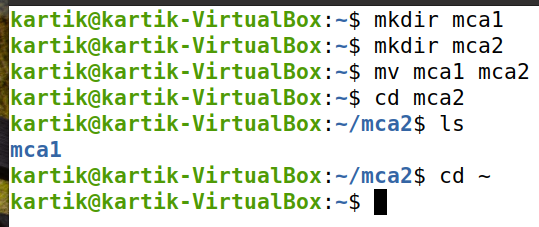
-> Failed to remove ‘a’ because the directory isn’t empty.

(e) $ mkdir /bin/mca

-> It will make a directory mca in /bin folder as we have to execute as superuser do .i.e. kartik.

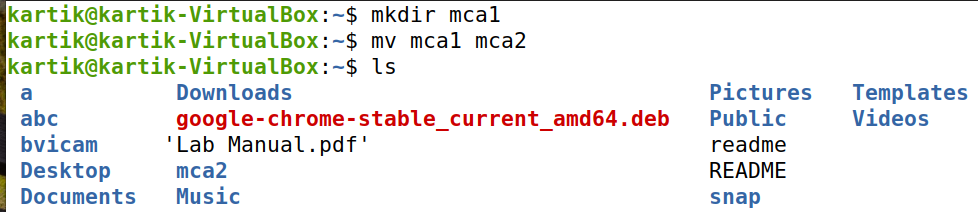


AP14:- How does the command mv mca1 mca2 behave, where both mca1 and mca2 are directories, when

1. mca2 exists

* In this case, “mca1” is moved to “mca2”.

(ii) mca2 doesn‟t exist?

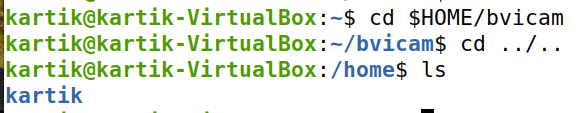


* In this case, the file name of “mca1” is replaced by “mca2”.

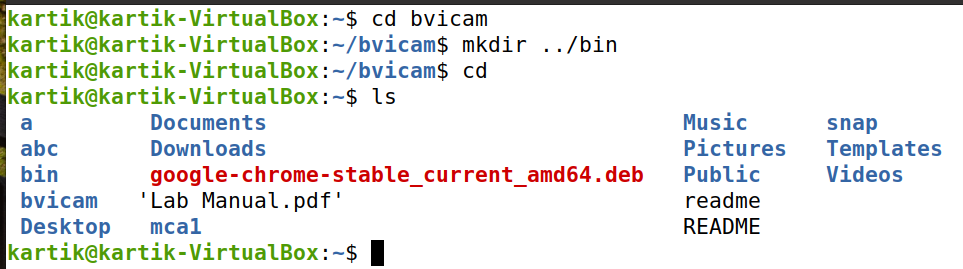
AP15:- Assuming that you are positioned in the directory /home/bvicam, what are these commands presumed to do, and explain whether they will work at all:

1. $ cd ../..

* It hops to two directories back.

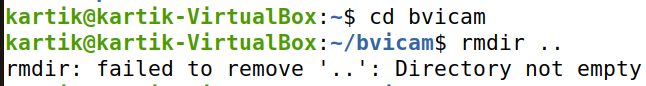


1. $ mkdir ../bin



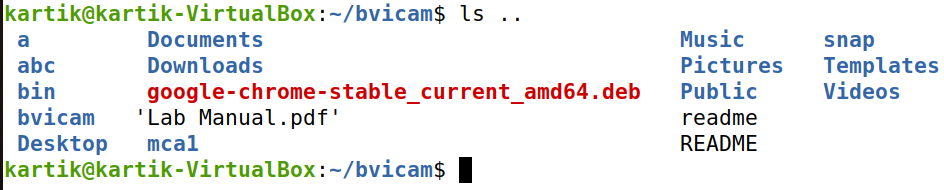
* It will create a directory named bin along with bvicam directory in the home folder.

1. $ rmdir ..



-> It will not work as bvicam directory has mca file that is why it can’t be deleted.

1. $ ls ..



* It will show the files of the directory prior to the bvicam directory.

Part B

BP1:- Apply Peterson algorithm for solving the critical section problem with C programming or Java multi-threaded programming. Assume appropriate code snippet for critical section.

Output:-

class peterson {

static boolean[] flag = {false, false};

static int turn = 0;

static int N = 4;

static Thread process(int i) {

return new Thread(() -> {

int j = 1 - i;

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

log(i+": want CS"); // LOCK

flag[i] = true; // 1

turn = j; // 2

while (flag[j] && turn == j) Thread.yield(); // 3

log(i+": in CS"+n);

sleep(1000 \* Math.random()); // 4

log(i+": done CS"); // UNLOCK

flag[i] = false; // 5

}

});

}

public static void main(String[] args) {

try {

log("Starting 2 processes (threads) ...");

Thread p0 = process(0);

Thread p1 = process(1);

p0.start();

p1.start();

p0.join();

p1.join();

}

catch (InterruptedException e) {}

}

static void sleep(double t) {

try { Thread.sleep((long)t); }

catch (InterruptedException e) {}

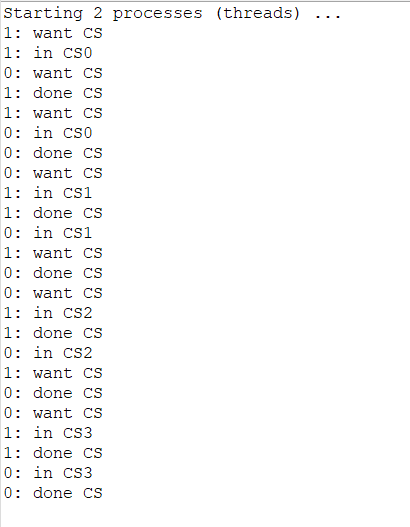
}

static void log(String x) {

System.out.println(x);

}

}

****

BP2:- Apply Bakery algorithm for synchronization of processes/threads in a C/Java program. Assume appropriate code snippet for critical section.

Output:-

import java.util.Arrays;

public class BP2

{

    static BakeryAlgo[] threads = new BakeryAlgo[10];

    static boolean[] flag = new boolean[10];

    static int[] processID = new int[10];

    static int sharedSum;

    public static void main(String[] args)

{

        sharedSum = 0;

        for (int i = 0; i < 10; i++)

{

            threads[i] = new BakeryAlgo(i);

            flag[i] = false;

        }

        for (int i = 0; i < 10; i++)

{

            threads[i].start();

        }

    }

static class BakeryAlgo extends Thread

{

        int PID;

        BakeryAlgo(int PID)

{

            this.PID = PID;

        }

        void lock(int i)

{

            flag[i] = true;

            processID[i] = Arrays.stream(processID).max().getAsInt() + 1;

            flag[i] = false;

            for (int process = 0; process < 10; process++)

{

                while (flag[process])

{

                    System.out.print("");

                }

 while ((processID[process] != 0) && ((processID[process] < processID[i]) || (process < i)))

{

                    System.out.print("");

                }

            }

        }

        void unlock(int i)

{

            processID[i] = 0;

        }

        @Override

        public void run() {

            // Locking other processes

            lock(PID);

            // Entering critical section

            System.out.println("Critical Section being using by P" + PID);

            for (int i = 0; i <= 100; i++)

{

                sharedSum += i;

            }

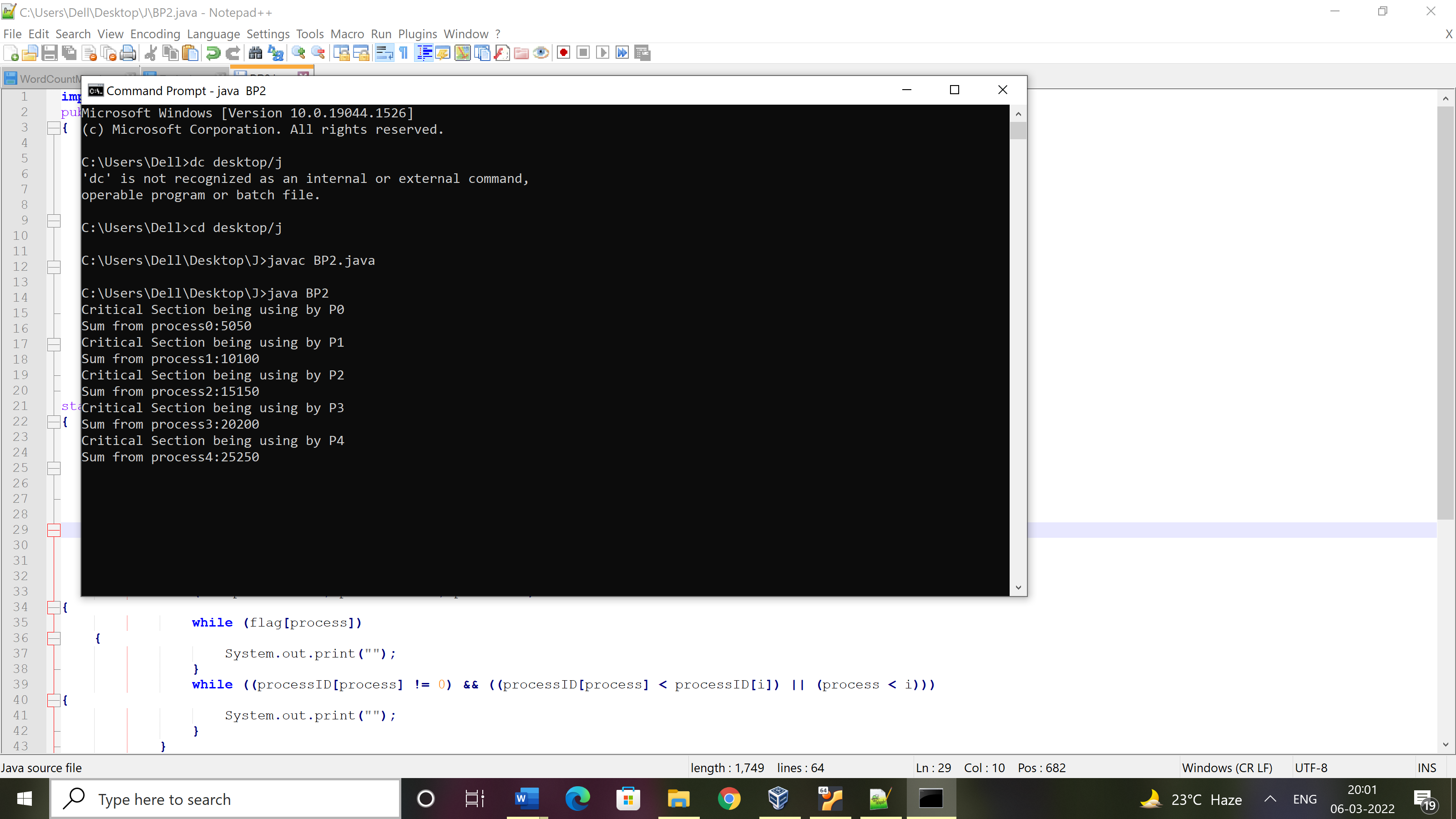
            System.out.println("Sum from process" + PID + ":" + sharedSum);

            // Unlocking other processes

            unlock(PID);

        }

}

}

BP3:- Write C/Java program to simulate and solve the Producer-Consumer problem.

Output:-

public class ProducerConsumer

{

public static void main(String[] args)

{

Shop c = new Shop();

Producer p1 = new Producer(c, 1);

Consumer c1 = new Consumer(c, 1);

p1.start();

c1.start();

}

}

class Shop

{

private int materials;

private boolean available = false;

public synchronized int get()

{

while (available == false)

{

try

{

wait();

}

catch (InterruptedException ie)

{

}

}

available = false;

notifyAll();

return materials;

}

public synchronized void put(int value)

{

while (available == true)

{

try

{

wait();

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

materials = value;

available = true;

notifyAll();

}

}

class Consumer extends Thread

{

private Shop Shop;

private int number;

public Consumer(Shop c, int number)

{

Shop = c;

this.number = number;

}

public void run()

{

int value = 0;

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

{

value = Shop.get();

System.out.println("Consumed value " + this.number+ " got: " + value);

}

try

{

Thread.sleep(1000);

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

}

class Producer extends Thread

{

private Shop Shop;

private int number;

public Producer(Shop c, int number)

{

Shop = c;

this.number = number;

}

public void run()

{

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

{

Shop.put(i);

System.out.println("Produced value " + this.number+ " put: " + i);

try

{

Thread.sleep(1000);

}

catch (InterruptedException ie)

{

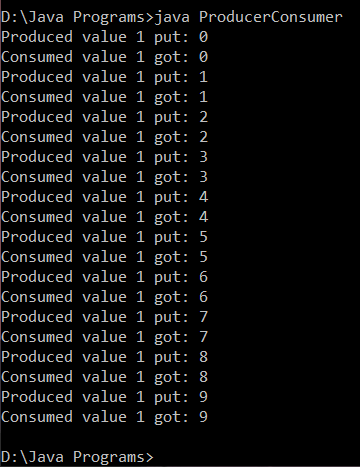
ie.printStackTrace();

}

}

}

}

******

BP4:- Implement Semaphore(s) in a C/Java-multithreaded program to simulate the working and solution of Reader-Writer problem. Assume multiple readers and writers.

Output:-

import java.util.concurrent.\*;

//A shared resource/class.

class Shared

{

static int count = 0;

}

class MyThread extends Thread

{

Semaphore sem;

String threadName;

public MyThread(Semaphore sem, String threadName)

{

super(threadName);

this.sem = sem;

this.threadName = threadName;

}

@Override

public void run() {

// run by thread A

if(this.getName().equals("A"))

{

System.out.println("Starting " + threadName);

try

{

// First, get a permit.

System.out.println(threadName + " is waiting for a permit.");

// acquiring the lock

sem.acquire();

System.out.println(threadName + " gets a permit.");

// Now, accessing the shared resource.

// other waiting threads will wait, until this

// thread release the lock

for(int i=0; i < 5; i++)

{

Shared.count++;

System.out.println(threadName + ": " + Shared.count);

// Now, allowing a context switch -- if possible.

// for thread B to execute

Thread.sleep(10);

}

} catch (InterruptedException exc) {

System.out.println(exc);

}

// Release the permit.

System.out.println(threadName + " releases the permit.");

sem.release();

}

// run by thread B

else

{

System.out.println("Starting " + threadName);

try

{

// First, get a permit.

System.out.println(threadName + " is waiting for a permit.");

// acquiring the lock

sem.acquire();

System.out.println(threadName + " gets a permit.");

// Now, accessing the shared resource.

// other waiting threads will wait, until this

// thread release the lock

for(int i=0; i < 5; i++)

{

Shared.count--;

System.out.println(threadName + ": " + Shared.count);

// Now, allowing a context switch -- if possible.

// for thread A to execute

Thread.sleep(10);

}

} catch (InterruptedException exc) {

System.out.println(exc);

}

// Release the permit.

System.out.println(threadName + " releases the permit.");

sem.release();

}

}

}

// Driver class

public class SemaphoreDemo

{

public static void main(String args[]) throws InterruptedException

{

// creating a Semaphore object

// with number of permits 1

Semaphore sem = new Semaphore(1);

// creating two threads with name A and B

// Note that thread A will increment the count

// and thread B will decrement the count

MyThread mt1 = new MyThread(sem, "A");

MyThread mt2 = new MyThread(sem, "B");

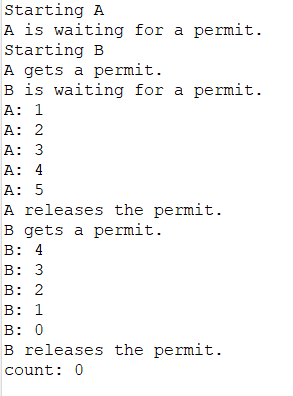
// stating threads A and B

mt1.start();

mt2.start();

// waiting for threads A and B

mt1.join();

 mt2.join();

// count will always remain 0 after

// both threads will complete their execution

System.out.println("count: " + Shared.count);

}

}

BP5:- Create a zombie process and an orphan process in a ‘C’ program with appropriate system calls.

Output:-

* Zombie Process

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<sys/types.h>

int main()

{

int i;

int pid = fork();

if (pid==0)

{

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

printf("I am Child\n");

}

else

{

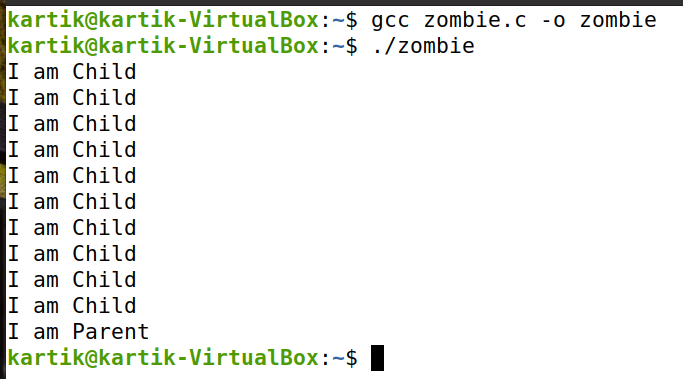
sleep(15);

printf("I am Parent\n");

}

return 0;

}



* Orphan Process

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<sys/types.h>

int main()

{

int i;

int pid = fork();

if (pid>0)

{

for (i=0; i<5; i++)

printf("I am Parent\n");

}

else

{

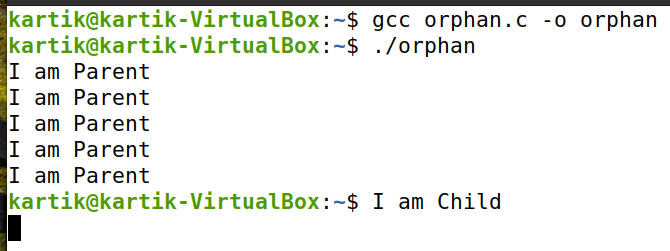
sleep(15);

printf("I am Child\n");

}

return 0;

}



BP6:- Write a ‘C’ program which creates a new process and allows both, child and parent, to report their identification numbers (ids).The parent process should wait for the termination of the child process.

Output:-

#include<stdio.h>

#include<fcntl.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<sys/wait.h>

int main()

{

int i;

pid\_t pid;

pid = fork();

if (pid==0)

{

for (i=0; i<5; i++)

printf("\nChild Process Count: %d",i);

printf("\nChild Process Id:%d",getpid());

printf("\nChild Closed");

}

else

{

wait(NULL);

printf("\nFrom Parent Process..");

printf("\nParent Process Id: %d",getpid());

printf("\nParent's Child Id: %d",pid);

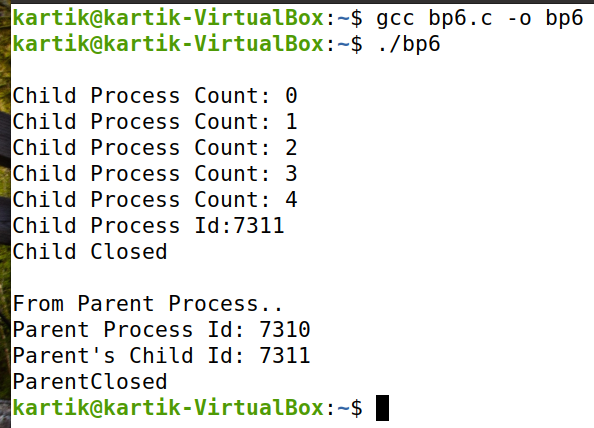
printf("\nParentClosed");

}

printf("\n");

return 0;

}



BP7:- Write two ‘C’ programs (a.c and b.c) where one program (a.c) creates a child process and then that child process executes the code of other program (b.c). The logic of program ‘b.c’ is to generate all the prime numbers within the specified limit.

Output:-

bp7.c

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

int main()

{

pid\_t pid;

pid = fork();

if (pid==0)

{

execl("./b", "B Program", NULL);

}

else

{

wait(NULL);

printf("\nFrom Parent Process..");

printf("\nParent Closed");

}

printf("\n");

return 0;

}

b.c

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<stdlib.h>

#include<stdio.h>

#include<fcntl.h>

void main()

{

int n;

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

{

int flag=0;

for(int j=2; j<=i/2; j++)

{

if(i%j==0)

{

flag=1;

break;

}

}

if(i==1)

{

printf("\n1 is neither Prime nor Composite..\n");

}

else

{

if(flag==0)

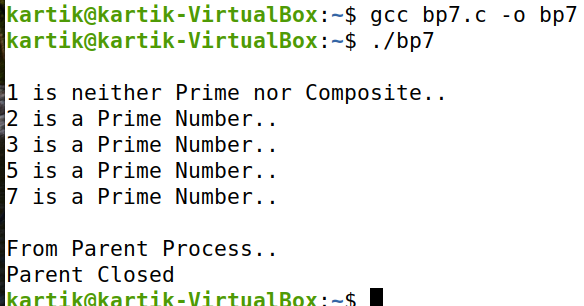
{

printf("%d is a Prime Number..\n",i);

}

}

}

}

BP8. Write an appropriate ‘C’ program which implements the concept of dynamic memory allocation (use of malloc(), calloc(), realloc(), and free() system call)**.**

Output:-

* Demonstration of malloc()

#include<stdio.h>

#include<malloc.h>

// #include<stdlib.h>

void main()

{

int n, \*ptr, i;

printf("Input array size: ");

scanf("%d",&n);

ptr = (int \*)malloc(n\*sizeof(int));

if(ptr==NULL)

{

printf("\nNo Allocation of memory");

}

else

{

printf("\nMemory Allocation Done!");

printf("\nAddress of first byte = %p", ptr);

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

{

ptr[i] = i+10;

}

}

printf("\nArray Elements: \n");

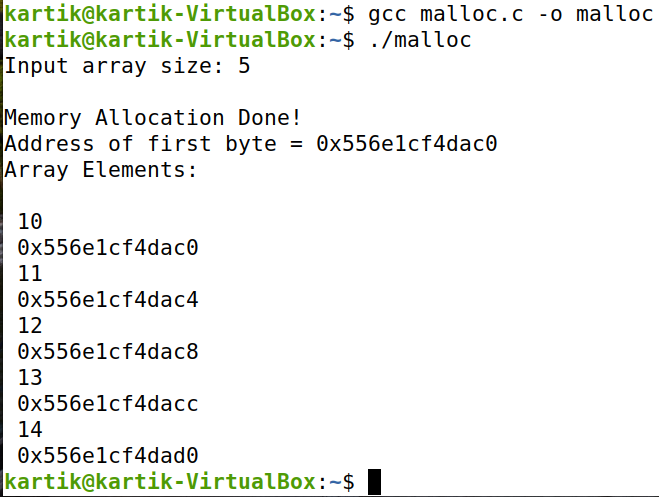
for(i=0; i<n; i++)

{

printf("%d ", ptr[i]);

// printf("%p ", ptr+i);

}

}

* Demonstration of calloc()

#include <stdio.h>

#include <stdlib.h>

int main()

{

// This pointer will hold the

// base address of the block created

int\* ptr;

int n, i;

// Get the number of elements for the array

n = 5;

printf("Number of elements: %d\n", n);

// Dynamically allocate memory using calloc()

ptr = (int\*)calloc(n, sizeof(int));

// Check if the memory has been successfully

// allocated by calloc or not

if (ptr == NULL) {

printf("Memory not allocated.\n");

exit(0);

}

else {

// Memory has been successfully allocated

printf("Memory successfully allocated using calloc.\n");

// Get the elements of the array

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

ptr[i] = i + 1;

}

// Print the elements of the array

printf("The elements of the array are: ");

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

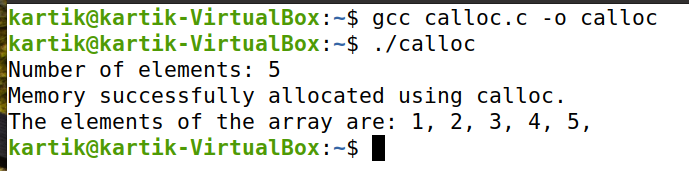
printf("%d, ", ptr[i]);

}

}

return 0;

}



* Demonstration of realloc()

#include <stdio.h>

#include <stdlib.h>

int main()

{

// This pointer will hold the

// base address of the block created

int\* ptr;

int n, i;

// Get the number of elements for the array

n = 5;

printf("Number of elements: %d\n", n);

// Dynamically allocate memory using calloc()

ptr = (int\*)calloc(n, sizeof(int));

// Check if the memory has been successfully

// allocated by malloc or not

if (ptr == NULL) {

printf("Memory not allocated.\n");

exit(0);

}

else {

// Memory has been successfully allocated

printf("Memory successfully allocated using calloc.\n");

// Get the elements of the array

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

ptr[i] = i + 1;

}

// Print the elements of the array

printf("The elements of the array are: ");

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

printf("%d, ", ptr[i]);

}

// Get the new size for the array

n = 10;

printf("\n\nThe new size of the array: %d\n", n);

// Dynamically re-allocate memory using realloc()

ptr = realloc(ptr, n \* sizeof(int));

// Memory has been successfully allocated

printf("Memory successfully re-allocated using realloc.\n");

// Get the new elements of the array

for (i = 5; i < n; ++i) {

ptr[i] = i + 1;

}

// Print the elements of the array

printf("The elements of the array are: ");

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

printf("%d, ", ptr[i]);

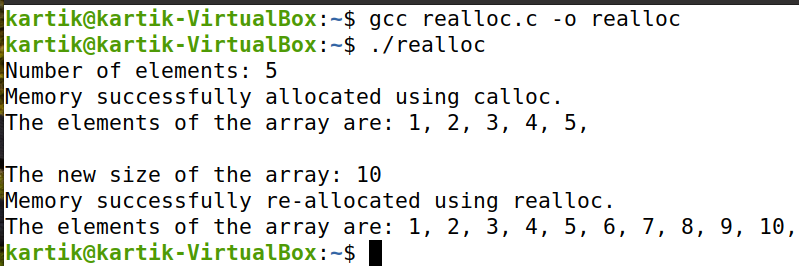
}

free(ptr);

}

return 0;

}



* Demonstration of free()

#include <stdio.h>

#include <stdlib.h>

int main()

{

// This pointer will hold the

// base address of the block created

int \*ptr, \*ptr1;

int n, i;

// Get the number of elements for the array

n = 5;

printf("Number of elements: %d\n", n);

// Dynamically allocate memory using malloc()

ptr = (int\*)malloc(n \* sizeof(int));

// Dynamically allocate memory using calloc()

ptr1 = (int\*)calloc(n, sizeof(int));

// Check if the memory has been successfully

// allocated by malloc or not

if (ptr == NULL || ptr1 == NULL) {

printf("Memory not allocated.\n");

exit(0);

}

else {

// Memory has been successfully allocated

printf("Memory successfully allocated using malloc.\n");

// Free the memory

free(ptr);

printf("Malloc Memory successfully freed.\n");

// Memory has been successfully allocated

printf("\nMemory successfully allocated using calloc.\n");

// Free the memory

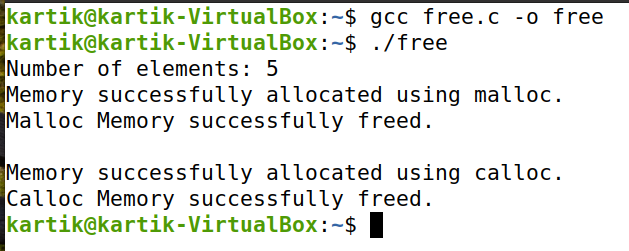
free(ptr1);

printf("Calloc Memory successfully freed.\n");

}

return 0;

}

******

BP9:- Create a text file, named as ‘courses.txt’ that contains the following four lines:

Write a ‘C’ program that forks three other processes. After forking, the parent process goes into wait state and waits for the children to finish their execution. Each child process reads a line from the ‘course.txt’ file (Child 1 Reads Line 1, Child 2 Reads Line 2, and Child 3 Reads Line 3) and each prints the respective line. The lines can be printed in any order.

Output:-

#include<fcntl.h>

#include<unistd.h>

#include<sys/stat.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<stdio.h>

void main()

{

pid\_t pid;

int fd;

int linecount[3]={16,17,21};

int startpoint[3]={0,17,33};

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

{

pid=fork();

if(pid==0)

{

char read\_value[linecount[i]];

fd=open("check.txt",O\_RDWR);

lseek(fd,startpoint[i],SEEK\_SET);

read(fd, read\_value,linecount[i]);

printf("%s\n",read\_value);

close(fd);

exit(0);

}

else

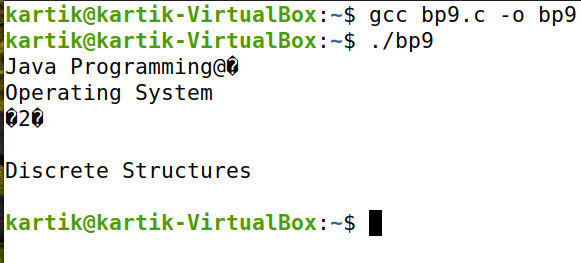
{

wait(NULL);

}

}

}

****

CP1:- Write an appropriate ‘C’ program which implements the concept of dynamic memory allocation (use of malloc(), calloc(), realloc(), and free() system call)**.**

* Demonstration of malloc()

#include<stdio.h>

#include<malloc.h>

// #include<stdlib.h>

void main()

{

int n, \*ptr, i;

printf("Input array size: ");

scanf("%d",&n);

ptr = (int \*)malloc(n\*sizeof(int));

if(ptr==NULL)

{

printf("\nNo Allocation of memory");

}

else

{

printf("\nMemory Allocation Done!");

printf("\nAddress of first byte = %p", ptr);

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

{

ptr[i] = i+10;

}

}

printf("\nArray Elements: \n");

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

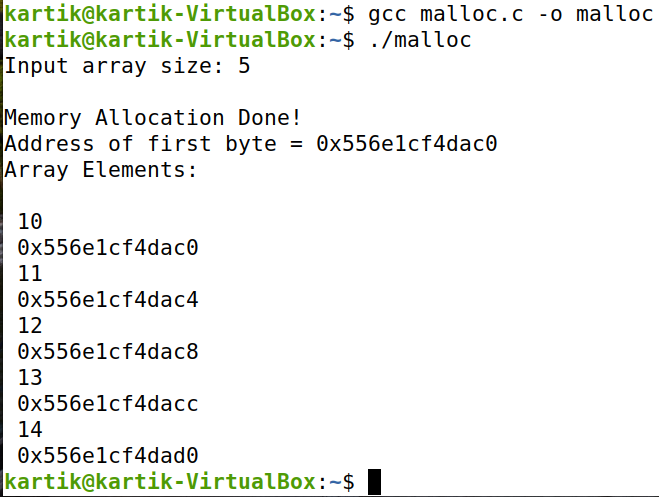
{

printf("%d ", ptr[i]);

// printf("%p ", ptr+i);

}

}



* Demonstration of calloc()

#include <stdio.h>

#include <stdlib.h>

int main()

{

// This pointer will hold the

// base address of the block created

int\* ptr;

int n, i;

// Get the number of elements for the array

n = 5;

printf("Number of elements: %d\n", n);

// Dynamically allocate memory using calloc()

ptr = (int\*)calloc(n, sizeof(int));

// Check if the memory has been successfully

// allocated by calloc or not

if (ptr == NULL) {

printf("Memory not allocated.\n");

exit(0);

}

else {

// Memory has been successfully allocated

printf("Memory successfully allocated using calloc.\n");

// Get the elements of the array

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

ptr[i] = i + 1;

}

// Print the elements of the array

printf("The elements of the array are: ");

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

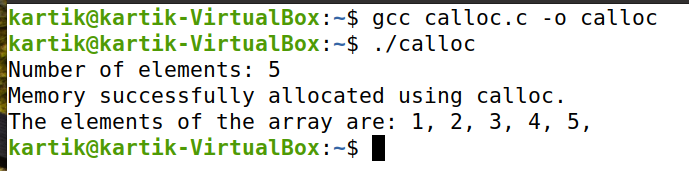
printf("%d, ", ptr[i]);

}

}

return 0;

}



* Demonstration of realloc()

#include <stdio.h>

#include <stdlib.h>

int main()

{

// This pointer will hold the

// base address of the block created

int\* ptr;

int n, i;

// Get the number of elements for the array

n = 5;

printf("Number of elements: %d\n", n);

// Dynamically allocate memory using calloc()

ptr = (int\*)calloc(n, sizeof(int));

// Check if the memory has been successfully

// allocated by malloc or not

if (ptr == NULL) {

printf("Memory not allocated.\n");

exit(0);

}

else {

// Memory has been successfully allocated

printf("Memory successfully allocated using calloc.\n");

// Get the elements of the array

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

ptr[i] = i + 1;

}

// Print the elements of the array

printf("The elements of the array are: ");

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

printf("%d, ", ptr[i]);

}

// Get the new size for the array

n = 10;

printf("\n\nThe new size of the array: %d\n", n);

// Dynamically re-allocate memory using realloc()

ptr = realloc(ptr, n \* sizeof(int));

// Memory has been successfully allocated

printf("Memory successfully re-allocated using realloc.\n");

// Get the new elements of the array

for (i = 5; i < n; ++i) {

ptr[i] = i + 1;

}

// Print the elements of the array

printf("The elements of the array are: ");

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

printf("%d, ", ptr[i]);

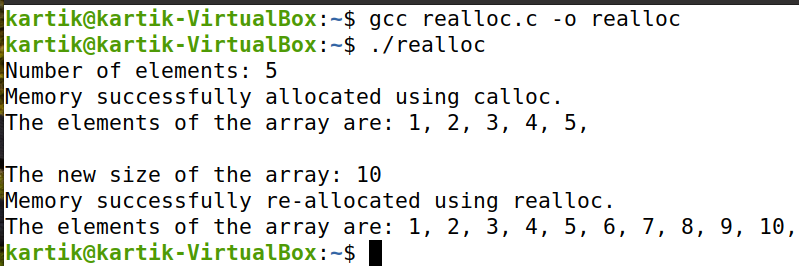
}

free(ptr);

}

return 0;

}



* Demonstration of free()

#include <stdio.h>

#include <stdlib.h>

int main()

{

// This pointer will hold the

// base address of the block created

int \*ptr, \*ptr1;

int n, i;

// Get the number of elements for the array

n = 5;

printf("Number of elements: %d\n", n);

// Dynamically allocate memory using malloc()

ptr = (int\*)malloc(n \* sizeof(int));

// Dynamically allocate memory using calloc()

ptr1 = (int\*)calloc(n, sizeof(int));

// Check if the memory has been successfully

// allocated by malloc or not

if (ptr == NULL || ptr1 == NULL) {

printf("Memory not allocated.\n");

exit(0);

}

else {

// Memory has been successfully allocated

printf("Memory successfully allocated using malloc.\n");

// Free the memory

free(ptr);

printf("Malloc Memory successfully freed.\n");

// Memory has been successfully allocated

printf("\nMemory successfully allocated using calloc.\n");

// Free the memory

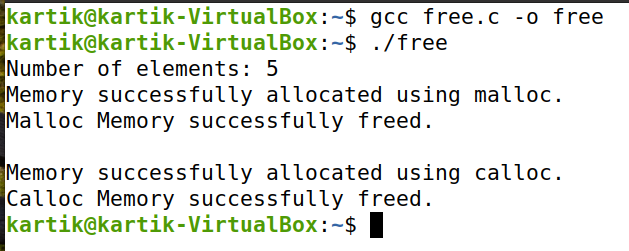
free(ptr1);

printf("Calloc Memory successfully freed.\n");

}

return 0;

}

******

CP2:- Write a ‘C’ program (using appropriate system calls of Linux) that generates ‘n’ integers and stores them in a text file, named as ‘All.txt’. Then, retrieve the stored integers from this file and copy to ‘Odd.txt’ and ‘Even.txt’ based upon the type of number, i.e. if the retrieved integer if odd number then store in ‘Odd.txt’ file or if the retrieved integer is even then store in ‘Even.txt’ file. Finally, display the contents of all three files on the screen.

Output:-

#include<stdio.h>

#include<fcntl.h>

#include<unistd.h>

#include<sys/stat.h>

int main()

{

int n;

printf("Enter value of n: ");

scanf("%d", &n);

char write\_value[n], read\_value[n];

int fd, fd\_odd,fd\_even;

printf("Writing to All.txt: ");

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

{

write\_value[i] = (i+1) + '0'; printf("%c",write\_value[i]);

}

printf("\n");

fd = open("All.txt",O\_CREAT | O\_WRONLY, 0777); write(fd, write\_value, n);

close(fd);

fd = open("All.txt",O\_RDWR); read(fd, read\_value, n);

char write\_odd[n/2], write\_even[n/2]; int ind\_odd=0,ind\_even=0;

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

{

int temp = read\_value[i];

if (temp%2==0)

{

write\_even[ind\_even++] = read\_value[i];

}

else

{

write\_odd[ind\_odd++] = read\_value[i];

}

}

printf("Writing to odd.txt: ");

for(int j = 0; j <= n/2; j++)

{

printf("%c ", write\_odd[j]);

}

printf("\n");

printf("Writing to even.txt: ");

for(int j = 0; j < n/2; j++)

{

printf("%c ", write\_even[j]);

}

printf("\n");

fd\_odd = open("odd.txt",O\_CREAT | O\_RDWR, 0777);

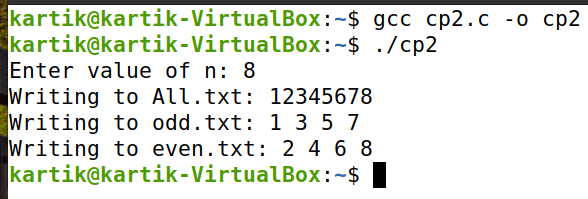
write(fd\_odd, write\_odd, n/2);

close(fd\_odd);

fd\_even = open("even.txt",O\_CREAT | O\_RDWR, 0777); write(fd\_even, write\_even, n/2);

close(fd\_even);

}



CP3:- Write a program in „C‟ which accepts the file or directory name and permission (access rights) from the user and then changes the access rights accordingly. Use appropriate system call(s) of Linux.

Output:-

# #include<stdio.h>

# #include<fcntl.h>

# #include<unistd.h>

# #include<sys/stat.h>

# int main()

# {

# int inp,check,fd; unsigned int perm; char name[51] = "";

# printf("Select\n1.Create Directory\n2.Create File\n");

# scanf("%d", &inp);

# printf("Enter permission: ");

# scanf("%o", &perm);

# switch(inp)

# {

# case 1: printf("Enter dir name: ");

# scanf("%50s", name);

# check = mkdir(name, perm);

# if (!check)

# {

# printf("Directory created\n");

# }

# else

# {

# printf("Unable to create directory\n");

# }

# break;

# case 2: printf("Enter file name: ");

# scanf("%50s", name);

# fd = open(name,O\_CREAT | O\_RDWR, perm);

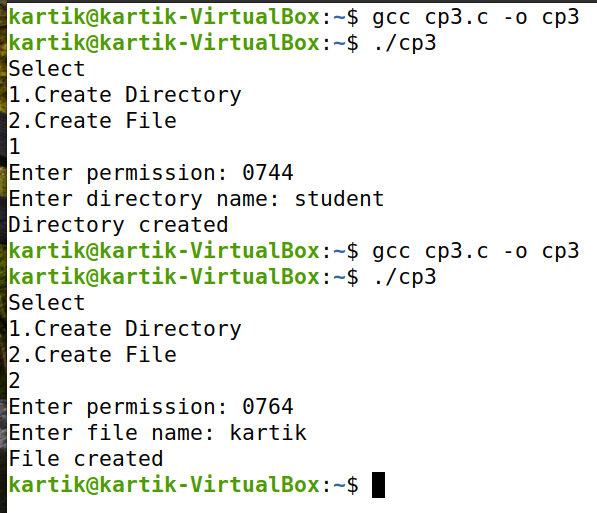
# printf("File created\n");

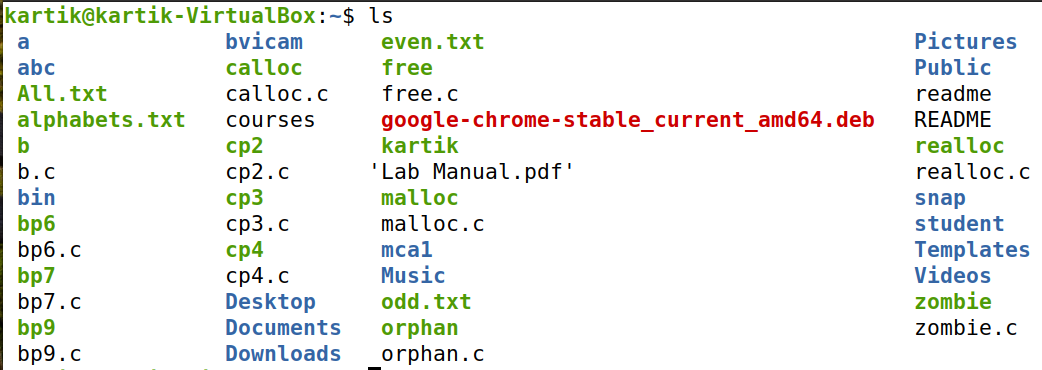
# close(fd);

# break;

# }

# }



****

CP4:- Write a ‘C’ program (using appropriate system calls of Linux) which generates and stores the characters from ‘a’ to ‘z’. Then, display the stored characters in alternative manner, like: a, c, e, g, …, etc.

Output:-

#include<stdio.h>

#include<fcntl.h>

#include<unistd.h>

#include<sys/stat.h>

void main()

{

int n=26,ind=0;

int fd,sizeRead,sizeWrite;

char write\_val[n], read\_val[n];

for(int i=97;i<123;i++)

{

write\_val[ind]=i;

ind++;

}

fd=open("alphabets.txt",O\_CREAT |O\_RDWR, 0777);

sizeWrite=write(fd, write\_val,n);

close(fd);

fd=open("alphabets.txt",O\_RDWR);

sizeRead=read(fd,read\_val,n);

close(fd);

for(int i=0;i<n;i+=2)

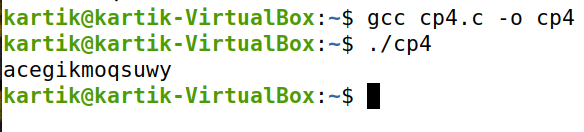
{

printf("%c",read\_val[i]);

}

printf("\n");

}



CP5:- Write a ‘C’ program (using appropriate system calls of Linux) which receives roll number and names of ‘n’ students, from the user one-by-one and then stores them in a text file, named as ‘Student.txt’. After inserting all ‘n’ roll numbers and names, display the contents of file. Also, display the access rights of the file ‘Student.txt’.

Output:-

#include<stdio.h>

#include<fcntl.h>

#include<unistd.h>

#include<sys/stat.h>

#include<string.h>

int main()

{

int n;

int fd, sRead, sWrite;

printf("Enter number of student: ");

scanf("%d", &n);

char write\_value[n][61], read\_value[n][61];

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

{

printf("Enter student name,roll no: ");

scanf("%60s", write\_value[i]);

}

fd = open("Student\_Infos.txt",O\_CREAT | O\_RDWR, 0777);

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

{

sWrite = write(fd, write\_value[i], 60);

}

close(fd);

fd = open("Student\_Infos.txt",O\_RDWR);

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

{

sRead = read(fd, read\_value[i], 60);

}

printf("Student data: \n");

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

{

printf("%s",read\_value[i]); printf("\n");

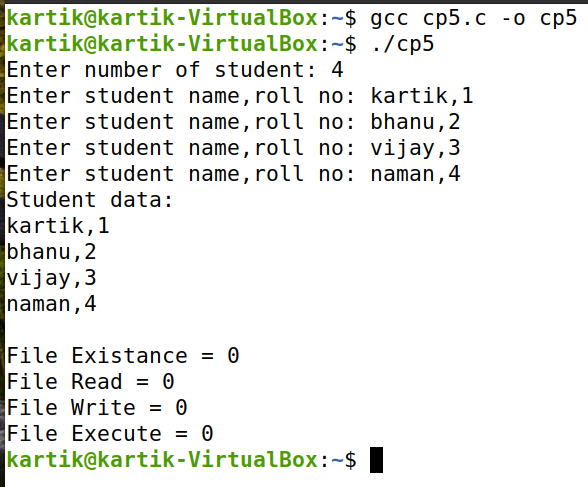
}

printf("\nFile Existance = %d" , access("Student\_Infos.txt", F\_OK));

printf("\nFile Read = %d", access("Student\_Infos.txt", R\_OK));

printf("\nFile Write = %d", access("Student\_Infos.txt", W\_OK));

printf("\nFile Execute = %d\n", access("Student\_Infos.txt", X\_OK));

}

C:\Users\krast.DESKTOP-UQL8ILS\Downloads\Linux File Content\cp5a.png

CP6:- Demonstrate the use of following system calls by writing an appropriate ‘C’ program.

Output:-

* Iseek()

#include<unistd.h>

#include<stdio.h>

#include<fcntl.h>

int main()

{

int fd;

char buffer[80];

char msg[50]="hello i am using linux distro ubuntu";

fd=open("check.txt",O\_RDWR);

printf("fd=%d",fd); if(fd!=-1)

{

printf("\ncheck.txt opened wih read write access\n");

write(fd,msg,sizeof(msg));

lseek(fd,0,SEEK\_SET);

read(fd,buffer,sizeof(msg));

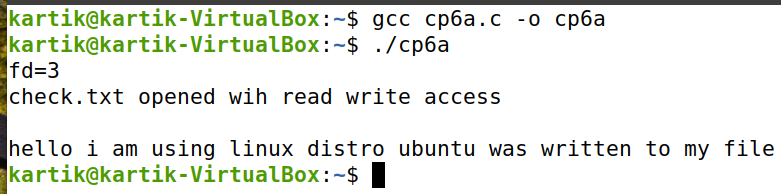
printf("\n%s was written to my file\n",buffer);

close(fd);

}

return 0;

}



* chmod()

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <sys/stat.h>

#include<fcntl.h>

#include<unistd.h>

int main()

{

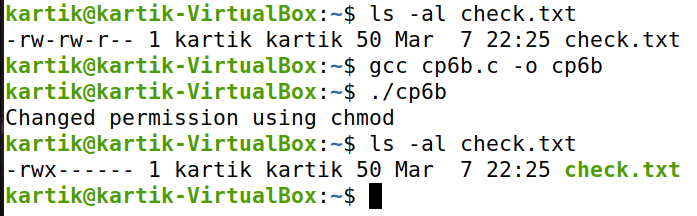
int fd;

fd=open("check.txt",O\_CREAT | O\_RDWR,0664);

printf("Changed permission using chmod\n");

chmod("check.txt",S\_IRWXU);

}



* umask()

#define \_POSIX\_SOURCE

#include <fcntl.h>

#include <sys/stat.h>

#include <unistd.h>

#undef \_POSIX\_SOURCE

#include <stdio.h>

main(){

int fd;

mode\_t old;

printf("Old mask in: %i\n", old=umask(S\_IRWXG));

if((fd = creat("new.txt", S\_IRWXU|S\_IRWXG)) < 0){

perror("creat() error");

}

else{

system("ls -l new.txt");

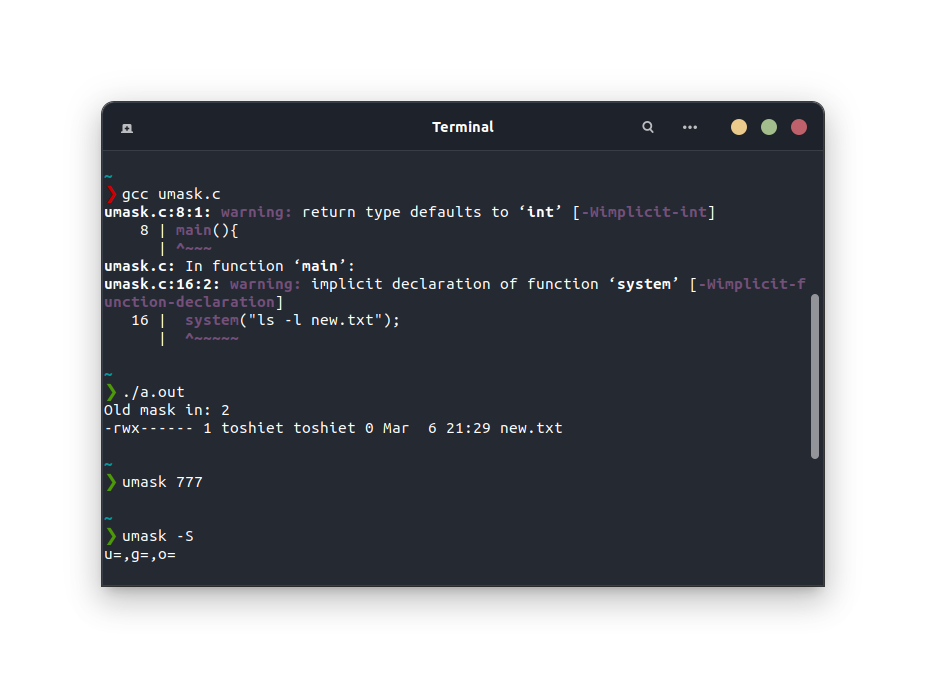
close(fd);

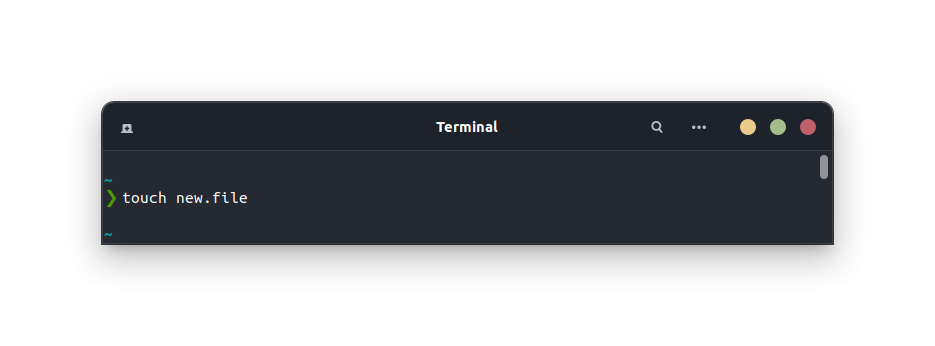
unlink("new.txt");

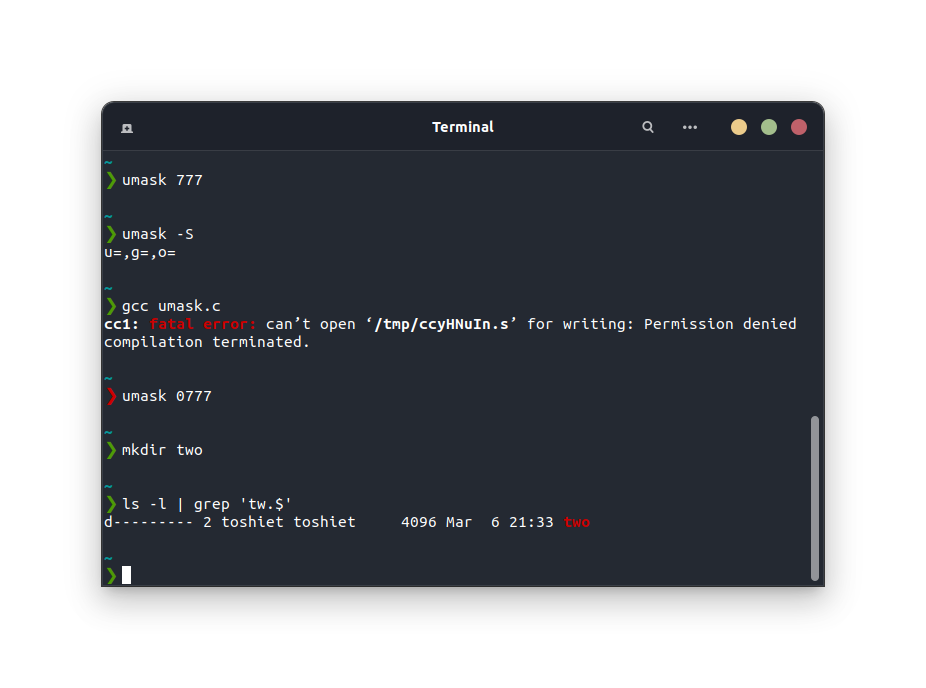
}

umask(old);

}







* access()

#include<stdio.h>

#include<unistd.h>

#include<errno.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

extern int errno;

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

{

int fd=access("check.txt",F\_OK);

if(fd==-1)

{

printf("Error Number:%d\n",errno);

perror("Error Description");

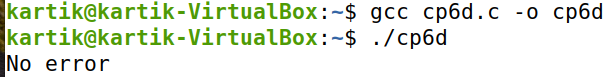
}

else

printf("No error\n");

return 0;

}



* utime()

#include <stdio.h>

#include <string.h>

#include<stdlib.h>

int main(void)

{

FILE\* uptimefile;

char uptime\_chr[28];

long uptime=0;

if((uptimefile=fopen("/proc/uptime","r"))==NULL)

perror("supt"),exit(EXIT\_FAILURE);

fgets(uptime\_chr, 12, uptimefile);

fclose(uptimefile);

uptime=strtol(uptime\_chr,NULL,10);

printf("System up for %ld seconds,%ld hours \n",uptime,uptime/ 3600);

exit(EXIT\_SUCCESS);

}

