

**BHARATI VIDYAPEETH’S**

**INSTITUTE OF COMPUTER APPLICATIONS & MANAGEMENT**

(Affiliated to Guru Gobind Singh Indraprastha University, Approved by AICTE, New Delhi)

OPERATING SYSTEM

WITH

LINUX LAB

**(MCA-163)**

**Practical File**

**Submitted To: Submitted By:**

Dr.Sunil Pratap Singh GARIMA DIXIT {35211604423}

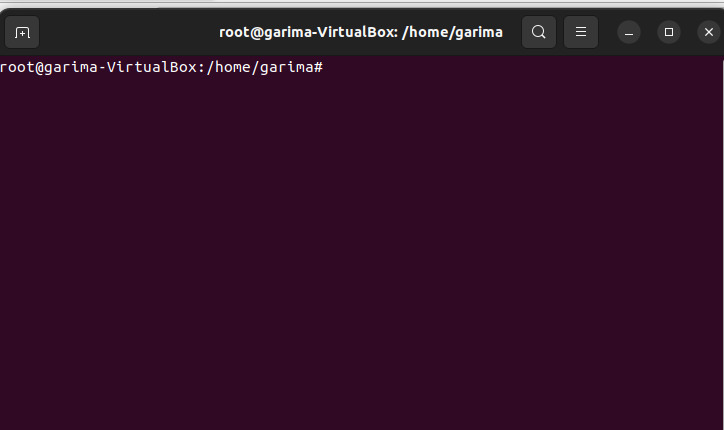
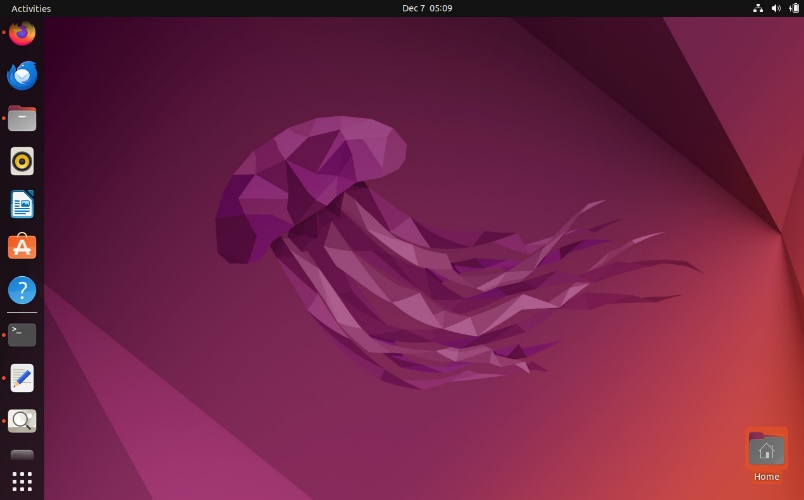
(Associate Professor) MCA 1ST SEM , SEC 2

**INDEX**

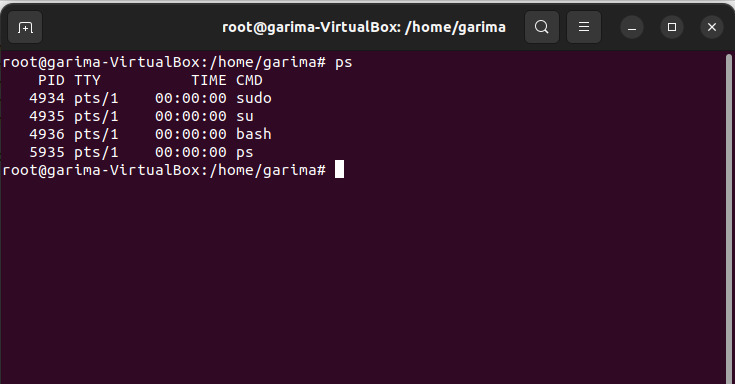
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Week No.** | **Lab ex. No.** | **Description of the Lab Exercise** | **Outcome mapping** | **Page no** | **Signature** |
|  | P1 | Install VirtualBox and then configure Linux (Ubuntu) in VirtualBox. |  |  |  |
|  | P2 | Run ps and note the PID of your shell. Log out and log in again and run ps again. What do you observe? |  |  |  |
|  | P3 | Enter the following commands, and note your observations: (i) who and tty, (ii) tput clear, (iii) id, (iv) ps and echo $$. |  |  |  |
|  | P4 | Run the following commands, and then invoke ls. What do you conclude? echo > README [Enter] echo > readme [Enter] |  |  |  |
|  | P5 | 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? |  |  |  |
|  | P6 | 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? |  |  |  |
|  | P7 | Run $ who am i and then interpret the output. |  |  |  |
|  | P8 | Find out whether the following commands are internal or external: echo, date, pwd, and ls. |  |  |  |
|  | P9 | Display the current date in the form dd/mm/yyyy. |  |  |  |
|  | P10 | 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 |  |  |  |
|  | P11 | 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 ~ |  |  |  |
|  | P12 | Frame ls command to (i) mark directories and executables separately, and  (ii) also display hidden files. |  |  |  |
|  | P13 | Find out the result of following: $ cat mca mca mca |  |  |  |
|  | P14 | 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 |  |  |  |
|  | P15 | How does the command mv mca1 mca2 behave, where both mca1 and mca2 are directories, when (i) mca2 exists, (ii) mca2 doesn‟t exist? |  |  |  |
|  | P16 | 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 .. |  |  |  |
|  | P17 | Apply Peterson algorithm for solving the critical section problem with C/Java multi-threaded programming. Assume appropriate code snippets for critical section. |  |  |  |
|  | P18 | Apply Bakery algorithm for synchronization of processes/threads in a C/Java program. Assume appropriate code snippets for critical sections. |  |  |  |
|  | P19 | Write C/Java program to simulate and solve the Producer-Consumer problem |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | P20 | Implement Semaphore(s) in a C/Java-multithreaded program to simulate the working and solution of Reader-Writer problem. Assume multiple readers and writers. |  |  |  |
|  | P21 | Create a zombie process and an orphan process in a „C‟ program with appropriate system calls. |  |  |  |
|  | P22 | 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. |  |  |  |
|  | P23 | 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 another program (B.c). The logic of program „B.c‟ is to generate all the prime numbers within the specified limit |  |  |  |
|  | P24 | Write an appropriate „C‟ program which implements the concept of dynamic memory allocation (use of malloc(), calloc(), realloc(), and free() system call). |  |  |  |
|  | P25 | 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. |  |  |  |
|  | P26 | 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 |  |  |  |
|  | P27 | 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. |  |  |  |
|  | P28 | 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. |  |  |  |
|  | P29 | 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‟ |  |  |  |
|  | P30 | Demonstrate the use of following system calls by writing an appropriate  „C‟ program. (a) lseek() (b) chmod() (c) umask() (d) access() (e) utime() |  |  |  |

**P1** Install VirtaulBox and then configure Linux (Ubantu) in VirtualBox.

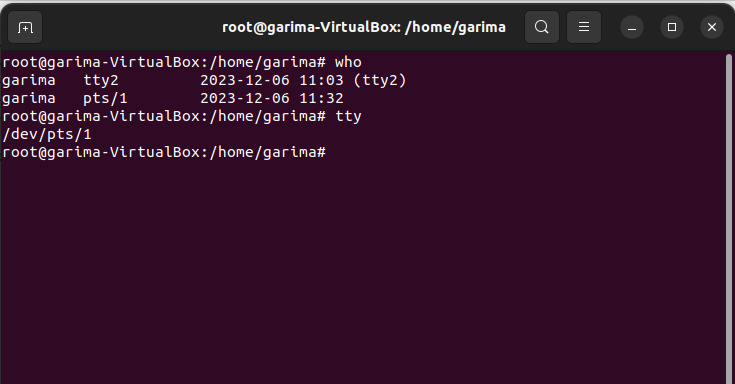


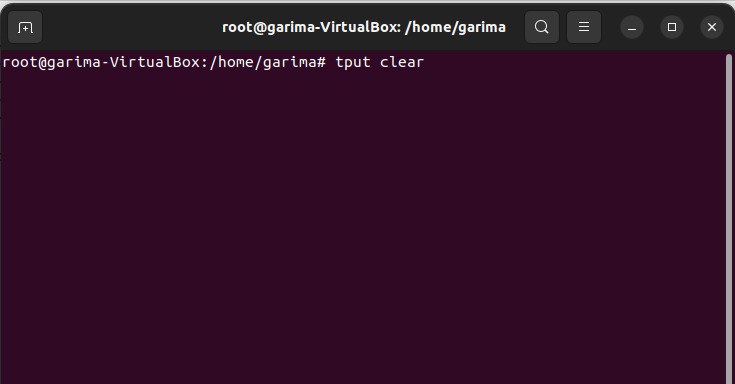
**P2** Run ps and note the PID of your shell. Log out and log in again and run ps again. What do you observe?

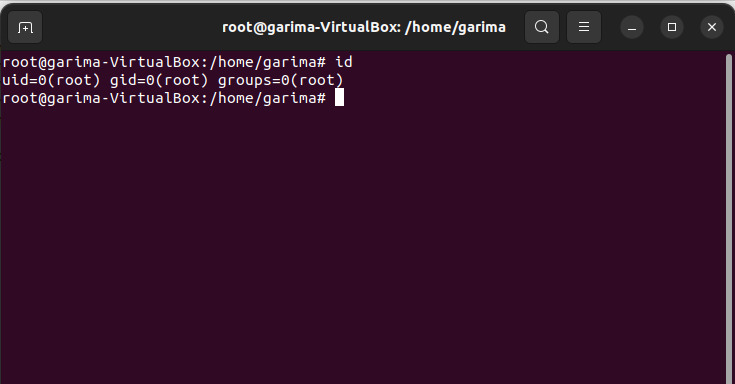


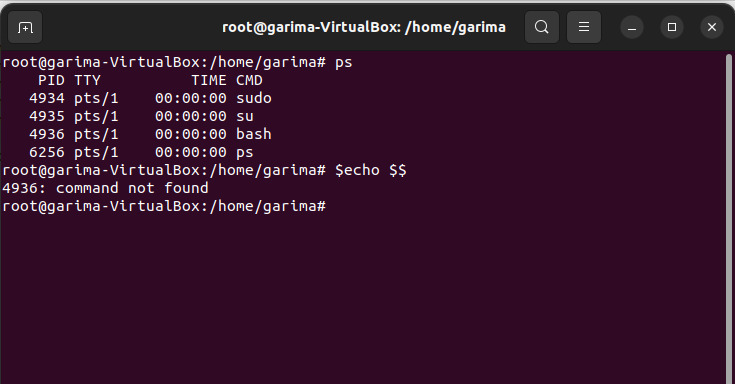
**P3** Enter the following commands, and note your observations:

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



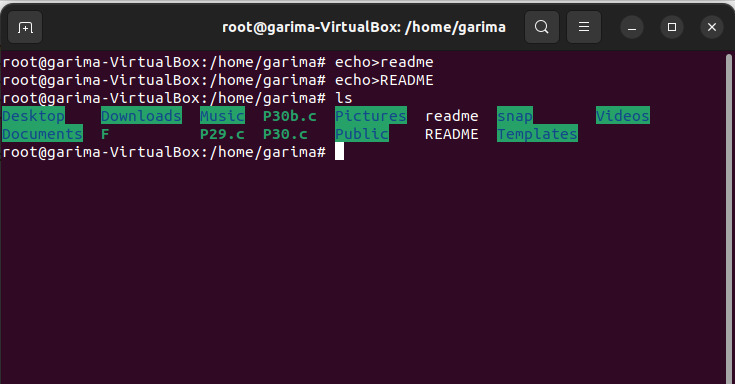




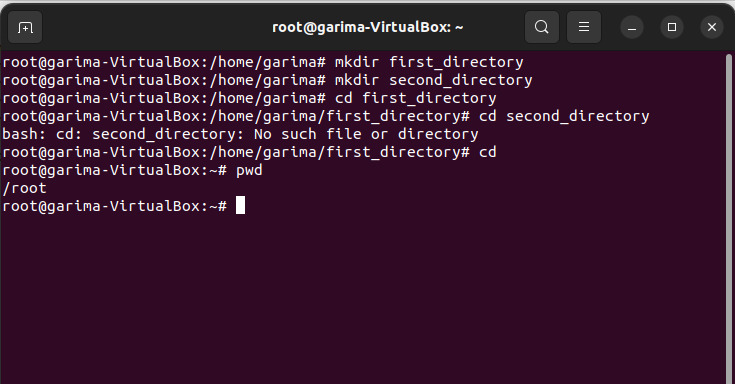


**P4** Run the following commands, and then invoke ls. What do you conclude?

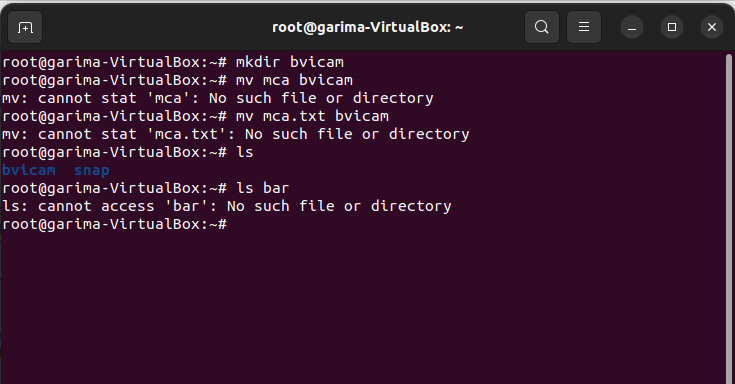
echo > README [Enter] echo > readme [Enter]



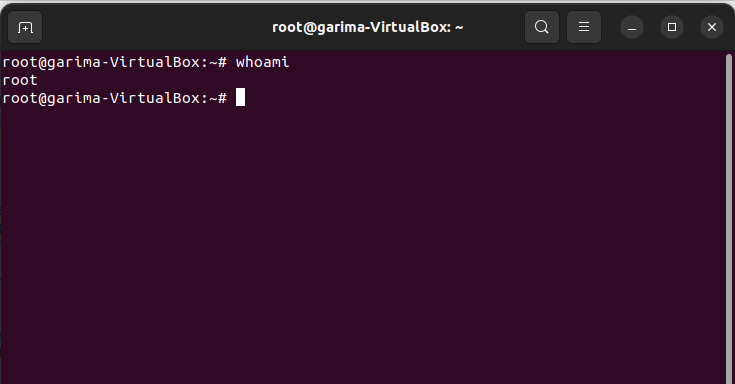
**P5** Create a directory, and change to that directory. Next, create another directory in the new directory, and then change to that directory too. Now, run $ cd without any arguments followed by pwd. What do you conclude?



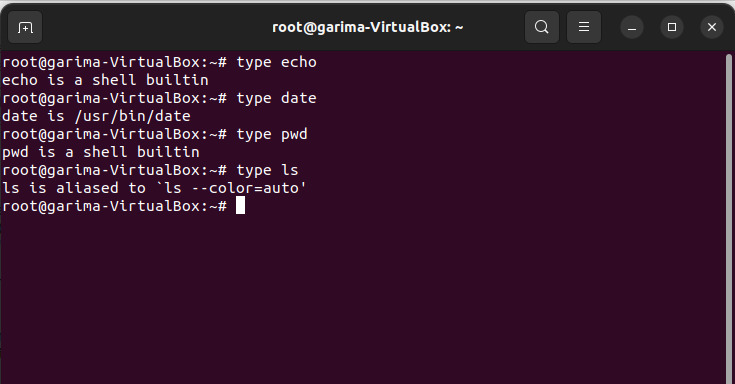
**P6** Create a file mca containing the words “Hello MCA Class!”. Now create a directory bvicam, and then run mv mca bvicam. What do you observe when you run both ls and ls bar?



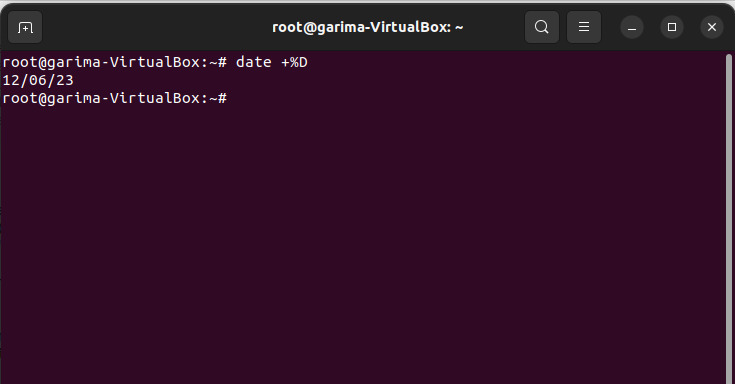
**P7** Run $ who am i and then interpret the output.



**P8** Find out whether the following commands are internal or external: echo, date, pwd, and ls.

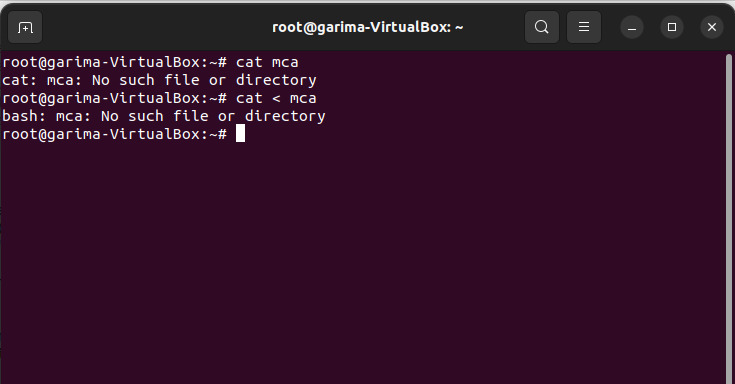


**P9** Display the current date in the form dd/mm/yyyy.



**P10** 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



**P11** Run the following commands, and discuss their output?

(a) $ uname

(b) $ passwd

(c) $ echo $SHELL

(d) $ man man

(e) $ which echo

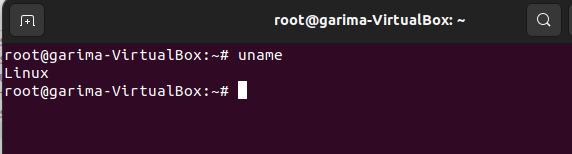
(f) $ type echo

(g) $ whereis ls

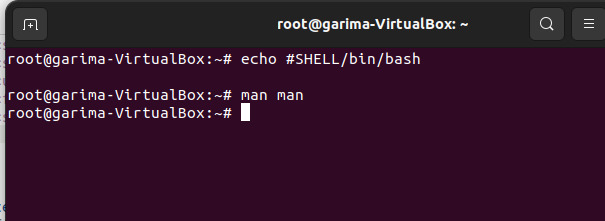
(h) $ cd

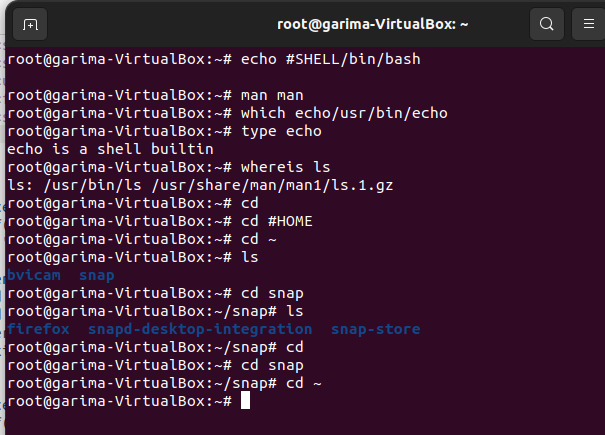
(i) $ cd $HOME

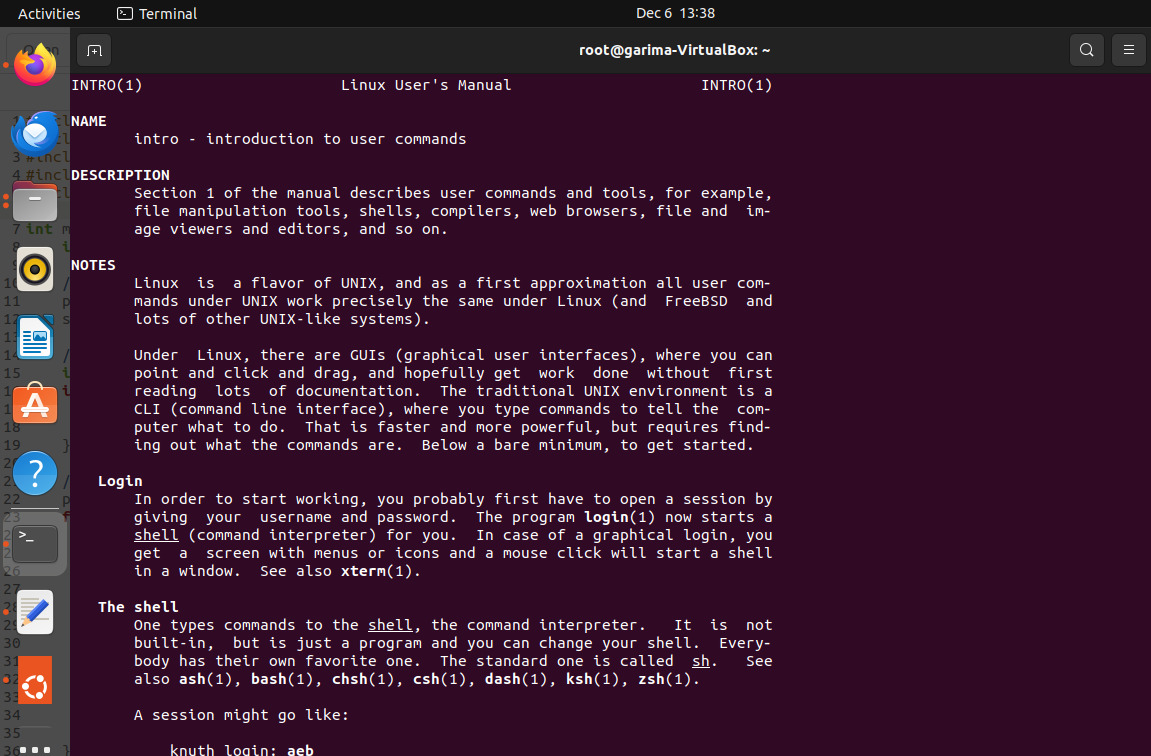
(j) $ cd ~









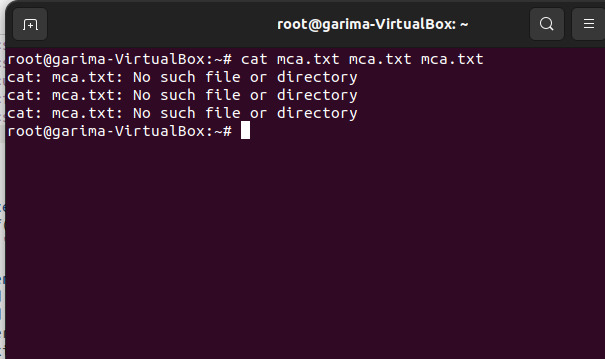


**P12** Frame ls command to (i) mark directories and executables separately, and (ii) also display hidden files.



**P13** Find out the result of following

$ cat mca mca mca



**P14** Run the following and determine which commands will work? Explain with reasons.

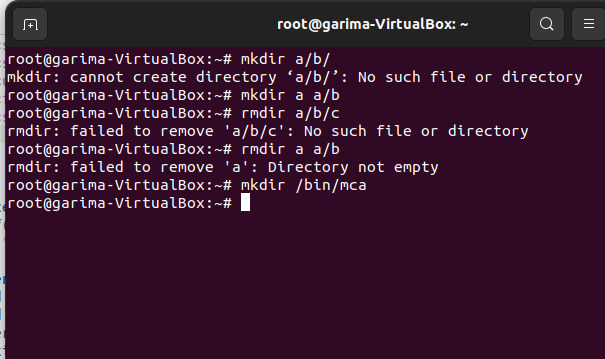
(a) $ mkdir a/b/

(b) $ mkdir a a/b

(c) $ rmdir a/b/c

(d) $ rmdir a a/b

(e) $ mkdir /bin/mca



**P15** How does the command mv mca1 mca2 behave, where both mca1 and mca2 are directories, when (i) mca2 exists, (ii) mca2 doesn‟t exist?

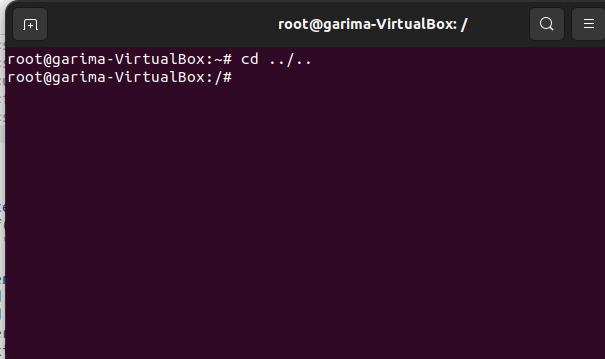
**P16** Assuming that you are positioned in the directory /home/bvicam, what are these commands presumed to do, and explain whether they will work at all:

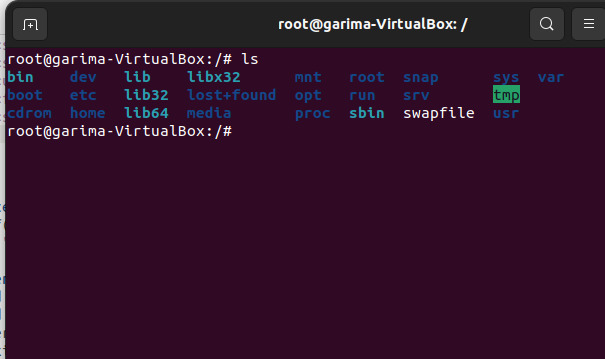
(a) $ cd ../..

(b) $ mkdir ../bin

(c) $ rmdir ..

(d) $ ls ..





**P17** Apply Peterson algorithm for solving the critical section problem with C/Java multi-threaded programming. Assume appropriate code snippet for critical section.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define TRUE 1

#define FALSE 0

int turn;

int flag[2];

void enter\_critical\_section(int process\_id) {

int other = 1 - process\_id;

flag[process\_id] = TRUE;

turn = process\_id;

// Busy-wait until it is your turn and the other process is not in its critical section

while (flag[other] == TRUE && turn == process\_id) {

// Do nothing (busy-wait)

}

}

void leave\_critical\_section(int process\_id) {

flag[process\_id] = FALSE;

}

void \*process(void \*arg) {

int process\_id = \*((int \*)arg);

// Perform non-critical section work

enter\_critical\_section(process\_id);

// Critical section

printf("Process %d entering critical section\n", process\_id);

// Perform critical section work

printf("Process %d leaving critical section\n", process\_id);

leave\_critical\_section(process\_id);

// Perform remainder section work

pthread\_exit(NULL);

}

int main() {

pthread\_t thread[2];

int process\_ids[2] = {0, 1};

// Initialize shared variables

turn = 0;

flag[0] = FALSE;

flag[1] = FALSE;

// Create two threads representing two processes

pthread\_create(&thread[0], NULL, process, (void \*)&process\_ids[0]);

pthread\_create(&thread[1], NULL, process, (void \*)&process\_ids[1]);

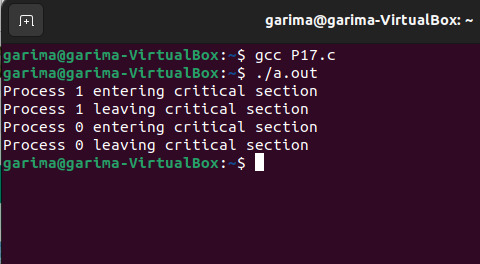
// Wait for threads to finish

pthread\_join(thread[0], NULL);

pthread\_join(thread[1], NULL);

return 0;

}



**P18** Apply Bakery algorithm for synchronization of processes/threads in a C/Java program. Assume appropriate code snippet for critical section.

**CODE:**

#include "pthread.h"

#include "stdio.h"

Importing POSIX Operating System API library

#include "unistd.h"

#include "string.h"

#define MEMBAR \_\_sync\_synchronize()

#define THREAD\_COUNT 8

volatile int tickets[THREAD\_COUNT];

volatile int choosing[THREAD\_COUNT];

volatile int resource;

void lock(int thread)

{

choosing[thread] = 1;

MEMBAR;

int max\_ticket = 0;

for (int i = 0; i < THREAD\_COUNT; ++i) {

int ticket = tickets[i];

max\_ticket = ticket > max\_ticket ? ticket : max\_ticket;

}

tickets[thread] = max\_ticket + 1;

MEMBAR;

choosing[thread] = 0;

MEMBAR;

for (int other = 0; other < THREAD\_COUNT; ++other) {

// Applying the bakery algorithm conditions

while (choosing[other]) {

}

MEMBAR;

while (tickets[other] != 0 && (tickets[other]< tickets[thread] || (tickets[other]

== tickets[thread]&& other < thread))) {

}

}

}

void unlock(int thread)

{MEMBAR;

tickets[thread] = 0;

}

void use\_resource(int thread)

{if (resource != 0) {

printf("Resource was acquired by %d, but is still in-use by %d!\n",

thread, resource);

}

resource = thread;

printf("%d using resource...\n", thread);

MEMBAR;

sleep(2);

resource = 0;

}void\* thread\_body(void\* arg)

{long thread = (long)arg;

lock(thread);

use\_resource(thread);

unlock(thread);

return NULL;

}

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

{

memset((void\*)tickets, 0, sizeof(tickets));

memset((void\*)choosing, 0, sizeof(choosing));

resource = 0;

pthread\_t threads[THREAD\_COUNT];

for (int i = 0; i < THREAD\_COUNT; ++i) {

// Creating a new thread with the function

//"thread\_body" as its thread routine

pthread\_create(&threads[i], NULL, &thread\_body, (void\*)((long)i));

}

for (int i = 0; i < THREAD\_COUNT; ++i) {

// Reaping the resources used by

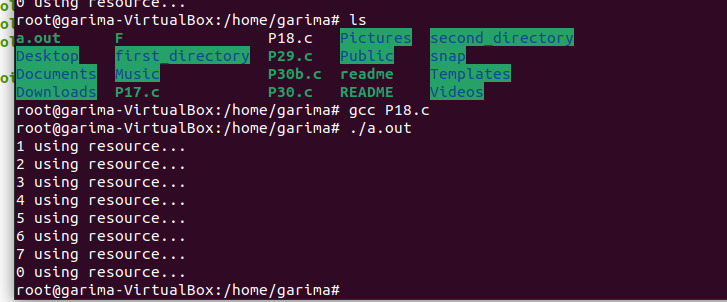
// all threads once their task is completed !

pthread\_join(threads[i], NULL);

}

return 0;

}



**P19** Write C/Java program to simulate and solve the Producer-Consumer problem.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

int mutex = 1;

// Number of full slots as 0

int full = 0;

// Number of empty slots as size

// of buffer

int empty = 10, x = 0;

// Function to produce an item and

// add it to the buffer

void producer()

{

// Decrease mutex value by 1

--mutex;

// Increase the number of full

// slots by 1

++full;

// Decrease the number of empty

// slots by 1

--empty;

// Item produced

x++;

printf("\nProducer produces"

"item %d",

x);

// Increase mutex value by 1

++mutex;

}

void consumer()

{

// Decrease mutex value by 1

--mutex;

// Decrease the number of full

// slots by 1

--full;

// Increase the number of empty

// slots by 1

++empty;

printf("\nConsumer consumes "

"item %d",

x);

x--;

// Increase mutex value by 1

++mutex;

}

// Driver Code

int main()

{

int n, i;

printf("\n1. Press 1 for Producer"

"\n2. Press 2 for Consumer"

for (i = 1; i > 0; i++) {

printf("\nEnter your choice:");

scanf("%d", &n);

// Switch Cases

switch (n) {

case 1:

// If mutex is 1 and empty

// is non-zero, then it is

// possible to produce

if ((mutex == 1)

&& (empty != 0)) {

producer();

}

// Otherwise, print buffer

// is full

else {

printf("Buffer is full!");

}

break;

case 2:

// If mutex is 1 and full

// is non-zero, then it is

// possible to consume

if ((mutex == 1)

&& (full != 0)) {

consumer();

}else {

printf("Buffer is empty!");

}

break;

// Exit Condition

case 3:

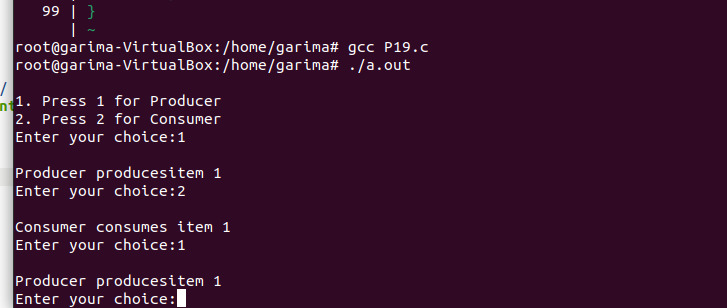
exit(0);

break;

}

}

}



**P20** Implement Semaphore(s) in a C/Java-multithreaded program to simulate the working and solution of Reader-Writer problem. Assume multiple readers and writers.

**CODE:**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

// A shared resource/structure.

struct Shared {

int count;

};

struct Shared shared\_instance = {0};

sem\_t semaphore;

// Function to simulate thread A behavior

void\* threadA(void\* arg) {

printf("Starting A\n");

// First, get a permit.

printf("A is waiting for a permit.\n");

// Acquiring the lock

sem\_wait(&semaphore);

printf("A gets a permit.\n");

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

shared\_instance.count++;

printf("A: %d\n", shared\_instance.count);

// Allowing a context switch -- if possible.

// For thread B to execute

usleep(10);

}

// Release the permit.

printf("A releases the permit.\n");

sem\_post(&semaphore);

return NULL;

}

// Function to simulate thread B behavior

void\* threadB(void\* arg) {

printf("Starting B\n");

// First, get a permit.

printf("B is waiting for a permit.\n");

// Acquiring the lock

sem\_wait(&semaphore);

printf("B gets a permit.\n");

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

shared\_instance.count--;

printf("B: %d\n", shared\_instance.count);

// Allowing a context switch -- if possible.

// For thread A to execute

usleep(10);

}

// Release the permit.

printf("B releases the permit.\n");

sem\_post(&semaphore);

return NULL;

}

int main() {

// Creating a Semaphore object with the number of permits 1

sem\_init(&semaphore, 0, 1);

// Creating two threads with name A and B

pthread\_t thread\_a, thread\_b;

// Starting threads A and B

pthread\_create(&thread\_a, NULL, threadA, NULL);

pthread\_create(&thread\_b, NULL, threadB, NULL);

// Waiting for threads A and B

pthread\_join(thread\_a, NULL);

pthread\_join(thread\_b, NULL);

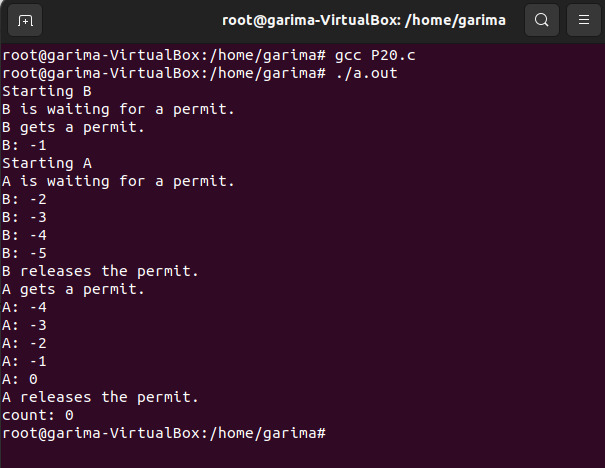
// Destroying the semaphore

sem\_destroy(&semaphore);

printf("count: %d\n", shared\_instance.count);

return 0;

}

****

**P21** Create a zombie process and an orphan process in a „C‟ program with appropriate system calls.

**CODE:**

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include<stdio.h>

int main()

{

// Fork returns process id

// in parent process

pid\_t child\_pid = fork();

// Parent process

if (child\_pid > 0)

sleep(50);

// Child process

else

exit(0);

printf("Woke up after 50 seconds...");

return 0;

}

**CODE:**

#include<stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

// Create a child process

int pid = fork();

if (pid > 0)

printf("in parent process");

// Note that pid is 0 in child process

// and negative if fork() fails

else if (pid == 0)

{

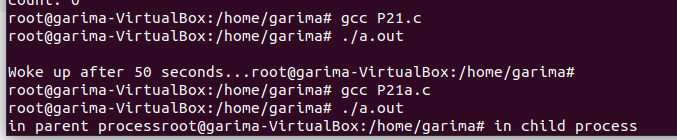
sleep(30);

printf("in child process");

}

return 0;

}



**P22** 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.

**CODE:**

#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()

{

pid\_t pid;

pid=fork();

if(pid==0)

{

printf("\nI'mInChild..");

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

printf("\nChildClosed");

exit(0);

}

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;

}



**P23** 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.

**CODE:**

A :-

#include<stdio>

#include<unistd>

int main(){

printf("I am executing A.c \n");

printf("PID of A.c is %d\n",getpid());

char \*args[]={"./B.c",NULL};

execv(args[0],args);

}

B :-

#include<unistd.h>

#include<sys/types.h>

#include<stdlib.h>

#include<stdio.h>

#include<fcntl.h>

void main()

{

int n;

printf("\nUp to How Many Numbers:");

scanf("%d",&n);

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

{

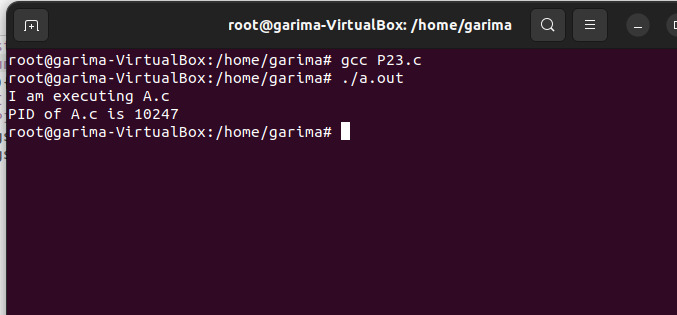
printf("%d is a Prime Number..\n",i);

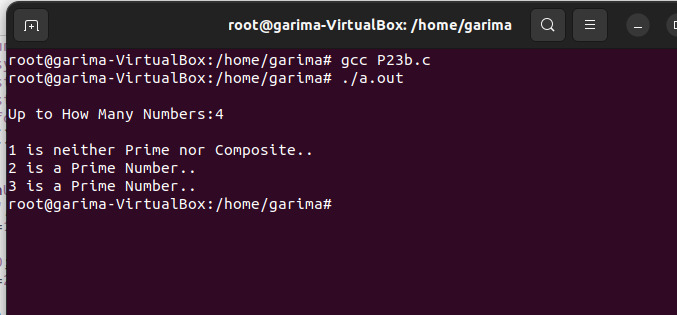
}

}

}

}





**P24** Write an appropriate „C‟ program which implements the concept of dynamic memory allocation (use of malloc(), calloc(), realloc(), and free() system call).

**CODE:**

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);

}

}

**CODE:**

calloc():

#include <stdio.h>

#include <stdlib.h>

int main()

{

int\* ptr;

int n, i;

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 {

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;

}

**CODE:**

realloc():

#include <stdio.h>

#include <stdlib.h>

int main()

{

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));

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]);

}

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;

}

**CODE:**

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;

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

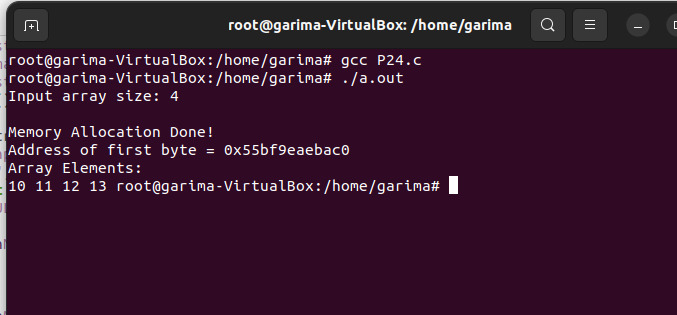
free(ptr1);

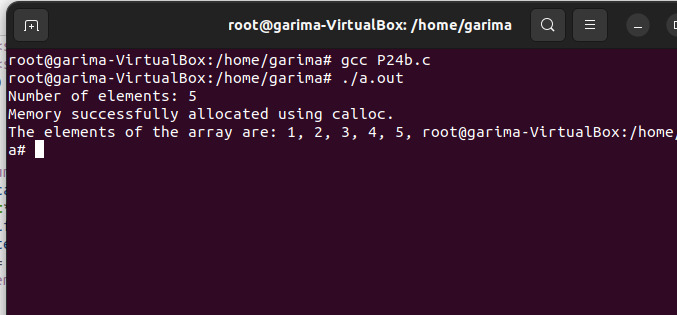
printf("Calloc Memory successfully freed.\n");

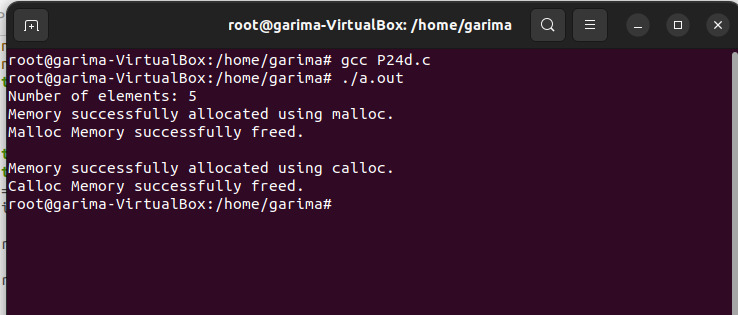
}

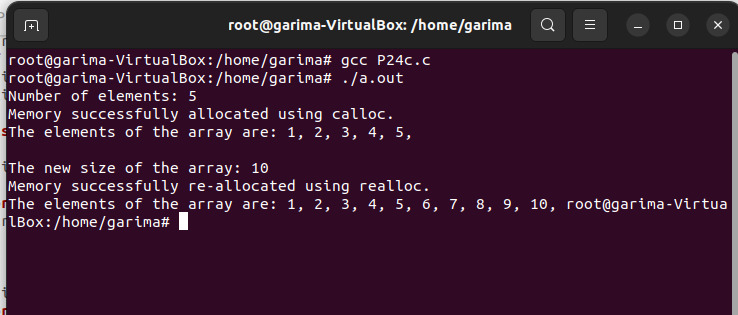
return 0;

}









**P25** 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.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

pid\_t pid;

FILE \*file;

char line[100];

// Create a file with the specified content

file = fopen("courses.txt", "w");

if (file == NULL) {

perror("Error creating file");

exit(EXIT\_FAILURE);

}

fprintf(file, "Java Programming\n");

fprintf(file, "Operating System\n");

fprintf(file, "Discrete Structure\n");

fprintf(file, "Write a C program\n");

fclose(file);

// Fork three child processes

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

pid = fork();

if (pid == -1) {

perror("Fork failed");

exit(EXIT\_FAILURE);

}

if (pid == 0) {

// Child process

file = fopen("courses.txt", "r");

if (file == NULL) {

perror("Error opening file");

exit(EXIT\_FAILURE);

}

// Read the corresponding line

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

fgets(line, sizeof(line), file);

}

// Print the line

printf("Child %d reads: %s", i + 1, line);

fclose(file);

exit(EXIT\_SUCCESS);

}

}

// Parent process waits for all children to finish

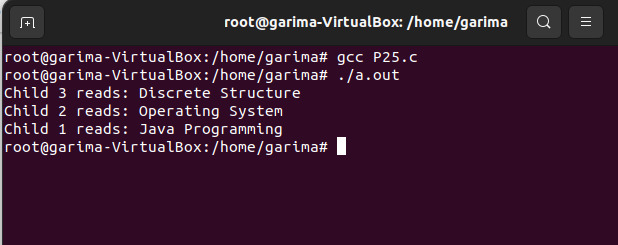
for (int i = 0; i < 3; i++) {

wait(NULL);

}

return 0;

}



**P26** 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

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <time.h>

int main() {

int n = 10; // Change this value to the desired number of integers

// Generate n random integers and store them in All.txt

FILE \*allFile = fopen("All.txt", "w");

if (allFile == NULL) {

perror("Error opening All.txt for writing");

exit(EXIT\_FAILURE);

}

srand((unsigned int)time(NULL)); // Seed for random numbers

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

int randomNumber = rand() % 100; // Generate a random number between 0 and 99

fprintf(allFile, "%d\n", randomNumber);

}

fclose(allFile);

// Retrieve integers from All.txt and categorize them into Odd.txt and Even.txt

allFile = fopen("All.txt", "r");

FILE \*oddFile = fopen("Odd.txt", "w");

FILE \*evenFile = fopen("Even.txt", "w");

if (allFile == NULL || oddFile == NULL || evenFile == NULL) {

perror("Error opening files for reading/writing");

exit(EXIT\_FAILURE);

}

int num;

while (fscanf(allFile, "%d", &num) == 1) {

if (num % 2 == 0) {

fprintf(evenFile, "%d\n", num);

} else {

fprintf(oddFile, "%d\n", num);

}

}

fclose(allFile);

fclose(oddFile);

fclose(evenFile);

// Display the contents of All.txt, Odd.txt, and Even.txt

printf("Contents of All.txt:\n");

FILE \*displayFile = fopen("All.txt", "r");

if (displayFile == NULL) {

perror("Error opening All.txt for display");

exit(EXIT\_FAILURE);

}

char ch;

while ((ch = fgetc(displayFile)) != EOF) {

putchar(ch);

}

fclose(displayFile);

printf("\n\nContents of Odd.txt:\n");

displayFile = fopen("Odd.txt", "r");

if (displayFile == NULL) {

perror("Error opening Odd.txt for display");

exit(EXIT\_FAILURE);

}

while ((ch = fgetc(displayFile)) != EOF) {

putchar(ch);

}

fclose(displayFile);

printf("\n\nContents of Even.txt:\n");

displayFile = fopen("Even.txt", "r");

if (displayFile == NULL) {

perror("Error opening Even.txt for display");

exit(EXIT\_FAILURE);

}

while ((ch = fgetc(displayFile)) != EOF) {

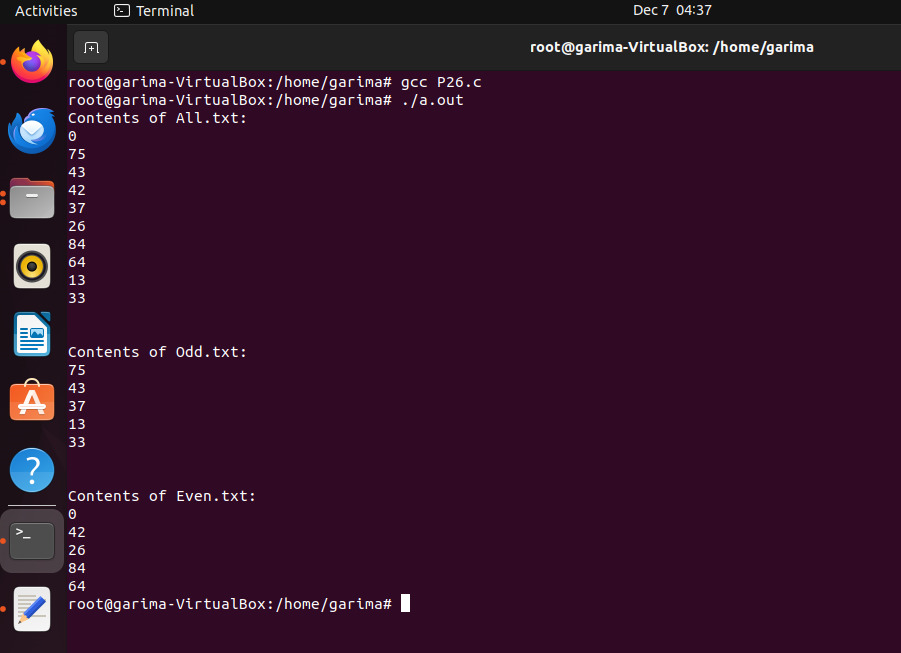
putchar(ch);

}

fclose(displayFile);

return 0;

}



**P27** 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

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

int main() {

char path[100];

mode\_t newPermission;

// Accept file or directory name and new permission from the user

printf("Enter the file or directory name: ");

scanf("%s", path);

printf("Enter the new permission (in octal): ");

scanf("%o", &newPermission);

// Change access rights using chmod system call

if (chmod(path, newPermission) == 0) {

printf("Access rights changed successfully.\n");

} else {

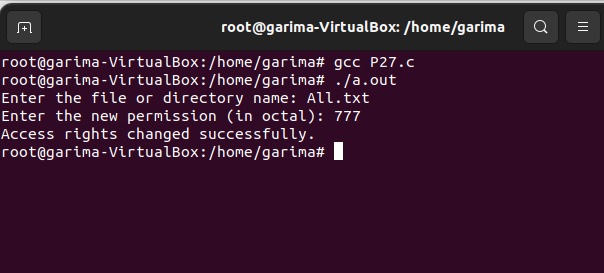
perror("Error changing access rights");

exit(EXIT\_FAILURE);

}

return 0;

}



**P28** 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.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

int main() {

char characters[26];

int fd;

// Generate and store characters from 'a' to 'z'

for (int i = 0; i < 26; i++) {

characters[i] = 'a' + i;

}

// Store characters in a file named "characters.txt"

fd = open("characters.txt", O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (fd == -1) {

perror("Error opening file for writing");

exit(EXIT\_FAILURE);

}

write(fd, characters, sizeof(char) \* 26);

close(fd);

// Display stored characters in an alternative manner

fd = open("characters.txt", O\_RDONLY);

if (fd == -1) {

perror("Error opening file for reading");

exit(EXIT\_FAILURE);

}

char buffer[26];

ssize\_t bytesRead = read(fd, buffer, sizeof(buffer));

if (bytesRead == -1) {

perror("Error reading file");

close(fd);

exit(EXIT\_FAILURE);

}

printf("Stored characters in an alternative manner: ");

for (int i = 0; i < bytesRead; i += 2) {

printf("%c, ", buffer[i]);

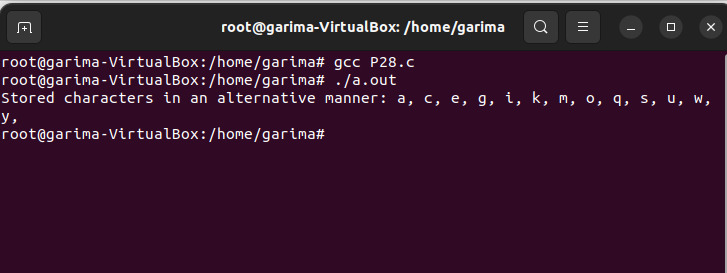
}

printf("\n");

close(fd);

return 0;

}



**P29** 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‟.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/stat.h>

int main() {

int n;

// Accept the number of students

printf("Enter the number of students: ");

scanf("%d", &n);

// Open or create the file "Student.txt" for writing

int fd = open("Student.txt", O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (fd == -1) {

perror("Error opening file for writing");

exit(EXIT\_FAILURE);

}

// Receive roll numbers and names from the user and write to the file

printf("Enter roll numbers and names for %d students:\n", n);

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

int roll;

char name[100];

printf("Student %d:\n", i + 1);

printf("Roll number: ");

scanf("%d", &roll);

printf("Name: ");

scanf("%s", name);

// Write roll number and name to the file

dprintf(fd, "%d %s\n", roll, name);

}

// Close the file

close(fd);

// Display the contents of the file

printf("\nContents of Student.txt:\n");

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

if (fd == -1) {

perror("Error opening file for reading");

exit(EXIT\_FAILURE);

}

char buffer[100];

ssize\_t bytesRead;

while ((bytesRead = read(fd, buffer, sizeof(buffer))) > 0) {

write(STDOUT\_FILENO, buffer, bytesRead);

}

close(fd);

// Display access rights of the file

struct stat fileStat;

if (stat("Student.txt", &fileStat) == 0) {

printf("\nAccess rights of Student.txt: %o\n", fileStat.st\_mode & 0777);

} else {

perror("Error getting file access rights");

exit(EXIT\_FAILURE);

}

return 0;

}



**P30** Demonstrate the use of following system calls by writing an appropriate „C‟ program.

(a) lseek()

(b) chmod()

(c) umask()

(d) access()

(e) utime()

**CODE:**

(a) lseek()

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

int fileDescriptor = open("example.txt", O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);

if (fileDescriptor == -1) {

perror("Error opening file");

return 1;

}

// Seek to the end of the file

off\_t newPosition = lseek(fileDescriptor, 0, SEEK\_END);

if (newPosition == -1) {

perror("Error using lseek");

close(fileDescriptor);

return 1;

}

printf("File size: %ld bytes\n", (long)newPosition);

close(fileDescriptor);

return 0;

}

(b) chmod()

#include <stdio.h>

#include <sys/stat.h>

int main() {

const char \*filename = "example.txt";

if (chmod(filename, S\_IRWXU | S\_IRGRP | S\_IXGRP | S\_IROTH) == -1) {

perror("Error using chmod");

return 1;

}

printf("File permissions changed successfully.\n");

return 0;

}

(d) access()

#include <stdio.h>

#include <unistd.h>

int main() {

const char \*filename = "example.txt";

if (access(filename, F\_OK) == 0) {

printf("File exists.\n");

} else {

perror("File doesn't exist or cannot be accessed");

return 1;

}

return 0;

}

(e) utime()

#include <stdio.h>

#include <utime.h>

int main() {

const char \*filename = "example.txt";

struct utimbuf times;

times.actime = 0; // Access time not modified

times.modtime = 1638810000; // Modification time set to a specific value

if (utime(filename, &times) == -1) {

perror("Error setting file times");

return 1;

}

printf("File times modified successfully.\n");

return 0;

}

