

**BHARATI VIDYAPEETH’S**

**INSTITUTE OF COMPUTER APPLICATIONS & MANAGEMENT**

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

**Operating Systems with Linux Lab.**

**(MCA-163)**

**Practical File**

**Submitted To: Submitted By:**

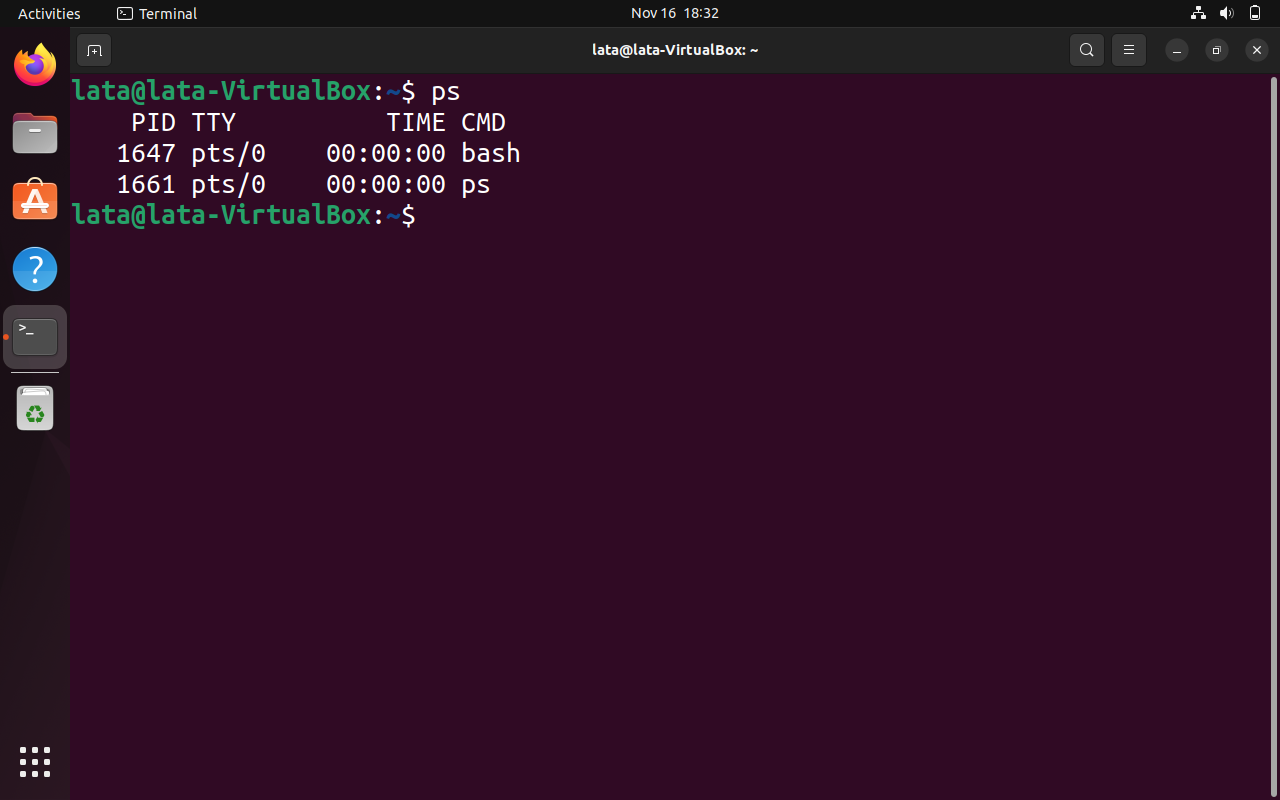
Dr. Sunil Pratap Singh Lata Yadav (T06111604423)

(Associate Professor) (MCA 1st Sem, Sec 2/2)

|  |  |  |  |
| --- | --- | --- | --- |
| **WEEK NO.** | **PROBLEMS WITH DESCRPTION**  **APPENDIX-A** | | |
|  | **Assignment Set: (Basic Linux Commands)** | | |
| 1 | P1 | Install **VirtaulBox** and then configure Linux (**Ubantu**) in VirtualBox | |
|  | [P2](#AP1) | Run ***ps*** and note the PID of your shell. Log out and log in again and run ***ps*** again. What do you observe? | |
| [P3](#AP2) | Enter the following commands, and note your observations:  (i) **who** and **tty**  (ii) **tput clear**  (iii) **id**  (iv) **ps** and **echo $$.** | |
| [P4](#AP3) | Run the following commands, and then invoke ls. What do you conclude?  echo > README [Enter]  echo > readme [Enter] | |
| [P5](#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? | |
| 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:   1. **echo** 2. **date** 3. **pwd** 4. **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? | |
| 1. **$ uname** 2. **$ passwd** 3. **$ echo $SHELL** 4. **$ man man** 5. **$ which echo** | 1. **$ type echo** 2. **$ whereis ls** 3. **$ cd** 4. **$ cd $HOME** 5. **$ 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.   1. **$ mkdir a/b/** 2. **$ mkdir a a/b** 3. **$ rmdir a/b/c** 4. **$ rmdir a a/b** 5. **(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:   1. **$ cd ../..** 2. **$ mkdir ../bin** 3. **$ rmdir ..** 4. **(d) $ ls ..** | |
|  | **Assignment Set: (Process Management)** | | |
| 2 | P17 | Apply **Peterson algorithm** for solving the critical section problem with C/Java multi-threaded programming. Assume appropriate code snippet for critical section | |
| P18 | Apply **Bakery algorithm** for synchronization of processes/threads in a C/Java program. Assume appropriate code snippet for critical section. | |
| 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. | |
|  | **Assignment Set: (Memory Management)** | | |
| 3 | 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 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 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() | |

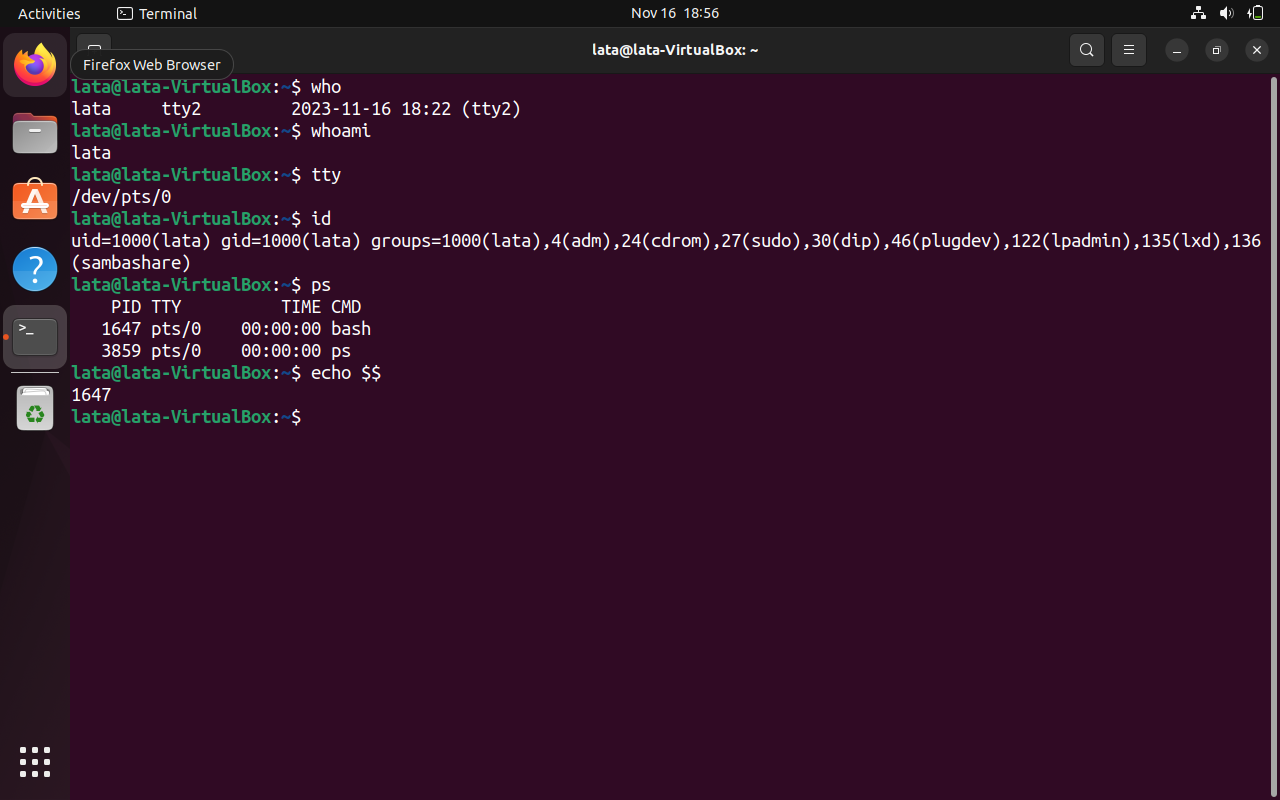
|  |  |
| --- | --- |
| **P1** | Install **VirtaulBox** and then configure Linux (**Ubantu**) in VirtualBox. |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **P2** | Run ***ps*** and note the PID of your shell. Log out and log in again and run ***ps*** again. What do you observe? |
|  | ps  cntrl+d - logout  ssh - login  ps |

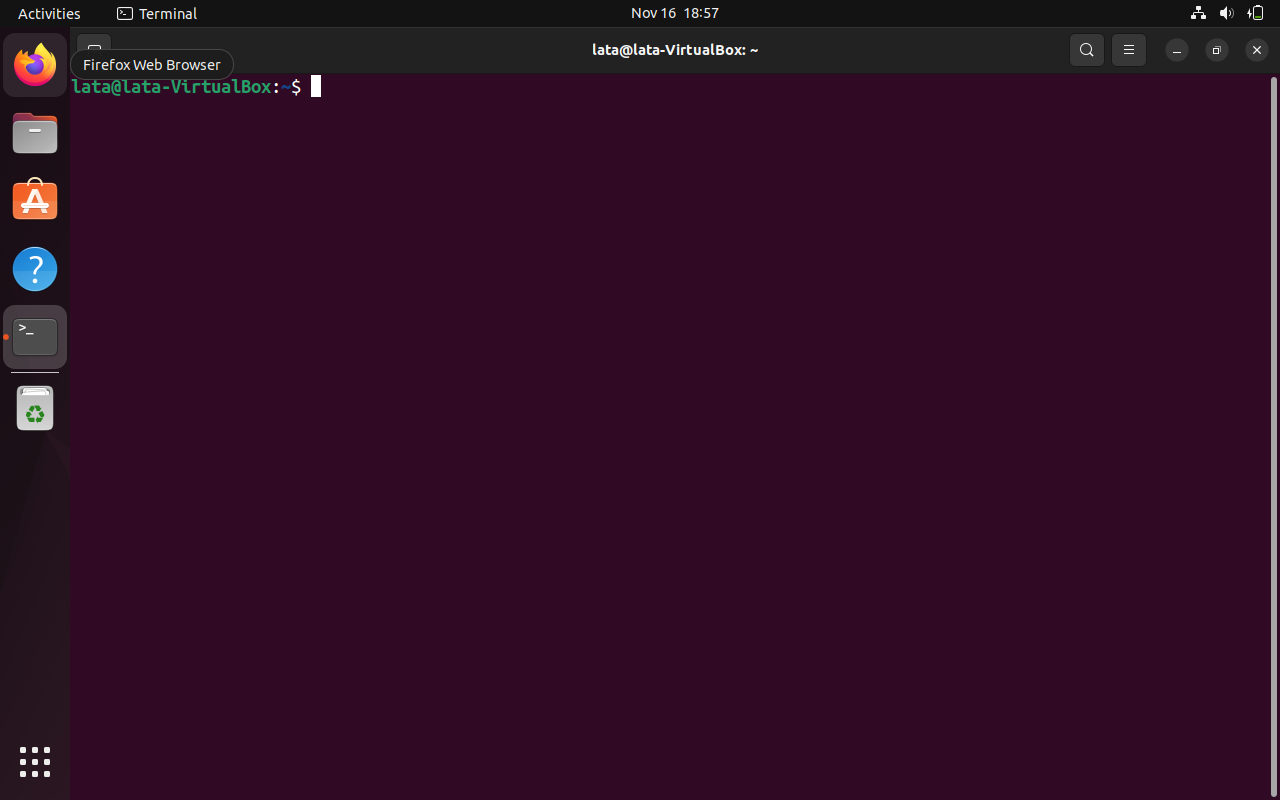
Ans: 

|  |  |
| --- | --- |
| **P3** | Enter the following commands, and note your observations:  (i) **who** and **tty**  (ii) **tput clear**  (iii) **id**  (iv) **ps** and **echo $$.** |

Ans:

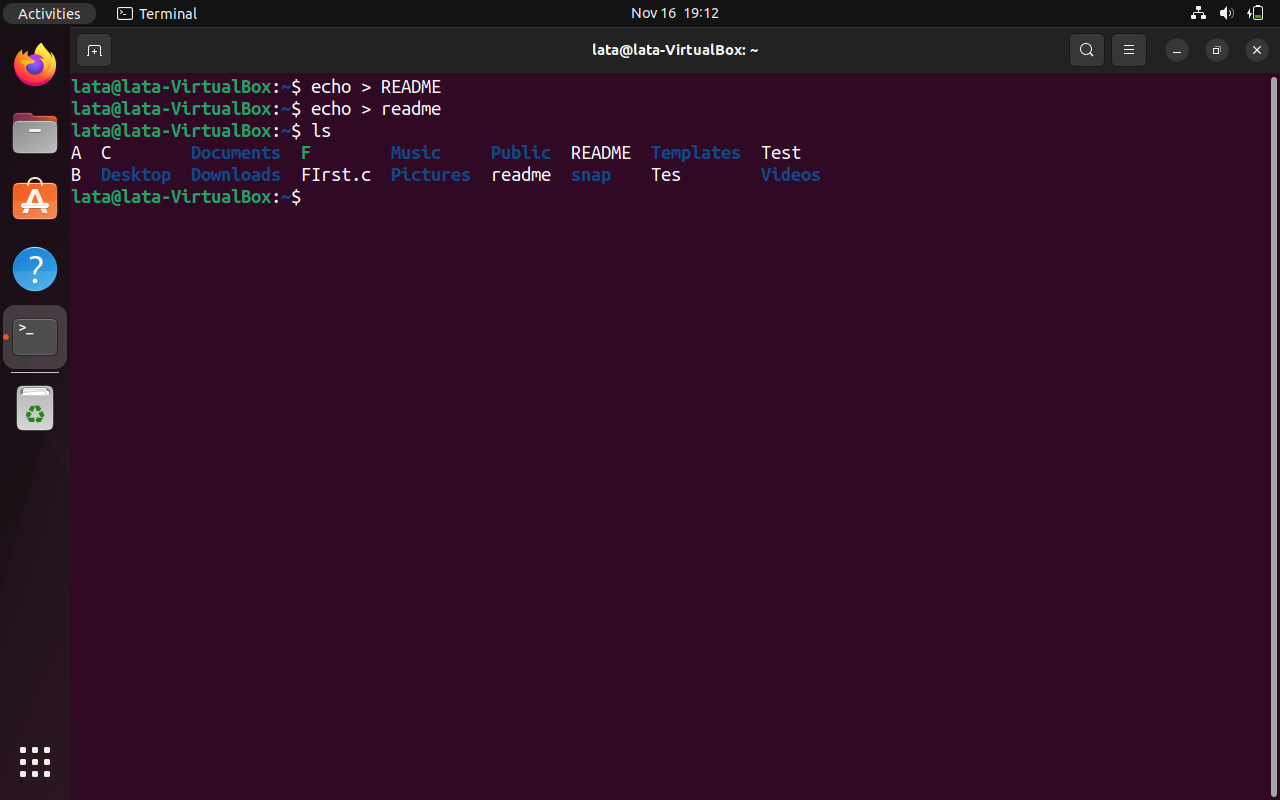


tput clear:



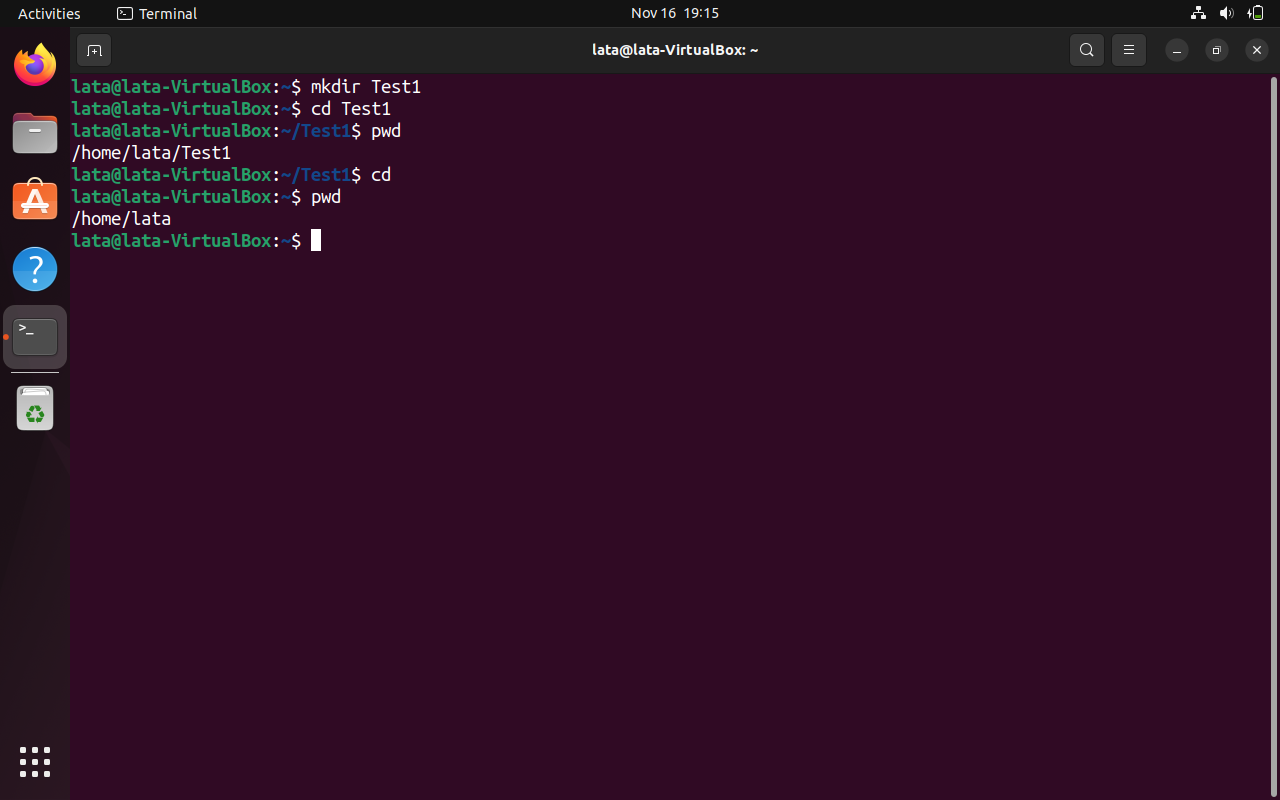
|  |  |
| --- | --- |
| **P4** | Run the following commands, and then invoke ls. What do you conclude?  echo > README [Enter]  echo > readme [Enter] |

Ans:



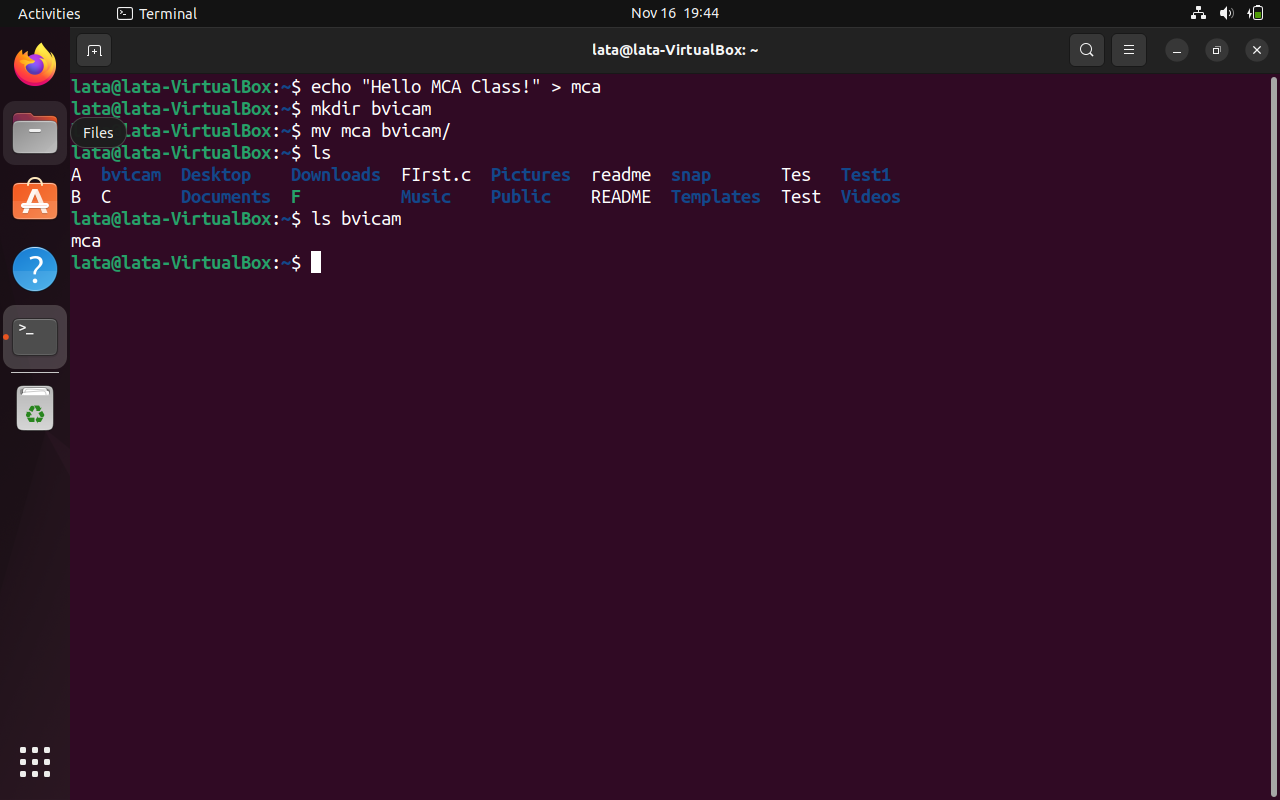
|  |  |
| --- | --- |
| **P5** | Create a directory, and change to that directory. Next, create another directory in the new directory, and then change to that directory too. Now, run ***$ cd*** without any arguments followed by ***pwd***. What do you conclude? |

Ans:



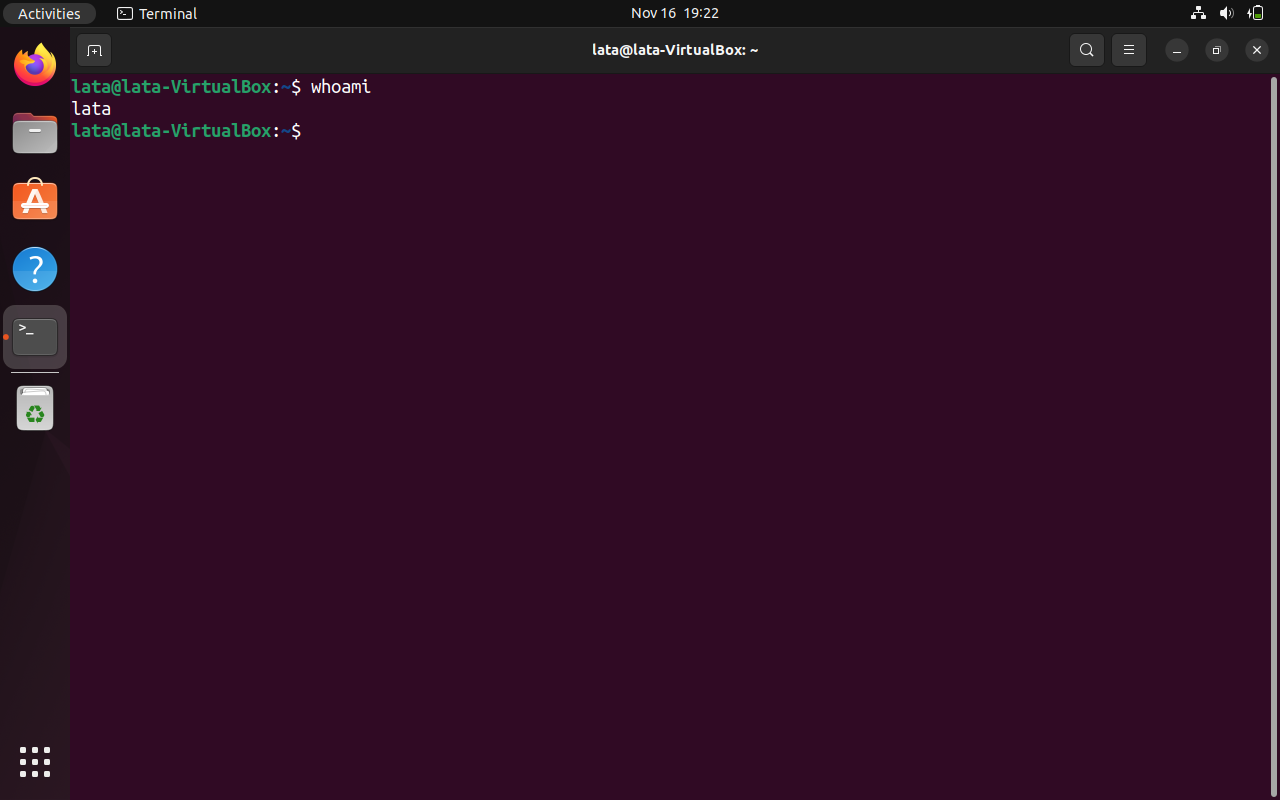
|  |  |  |
| --- | --- | --- |
| **P6** | Create a file mca containing the words “**Hello MCA Class!**”. Now create a directory bvicam, and  then run **mv mca bvicam**. What do you observe when you run both **ls** and **ls** **bar**? | |
| **Ans** | Now, let's analyze the outcome when you run both **ls** and **ls bvicam**:   * Running **ls** will show the contents of the current directory. Since you moved the "mca" file to the "bvicam" directory, it will not be listed in the current directory. * Running **ls bvicam** will display the contents of the "bvicam" directory. You should see the "mca" file in that directory.   In summary, after these commands, you will observe that the "mca" file is no longer in the current directory but has been successfully moved to the "bvicam" directory. | |
| **Methods** | cat > mca.txt  Hello MCA Class!  ctrl+d  ls  mkdir bvicam  mv mca.txt bvicam  ls  cd bvicam  ls bvicam | echo "Hello MCA Class!" > mca  mkdir bvicam  mv mca bvicam/  ls  ls bvicam |

Ans:



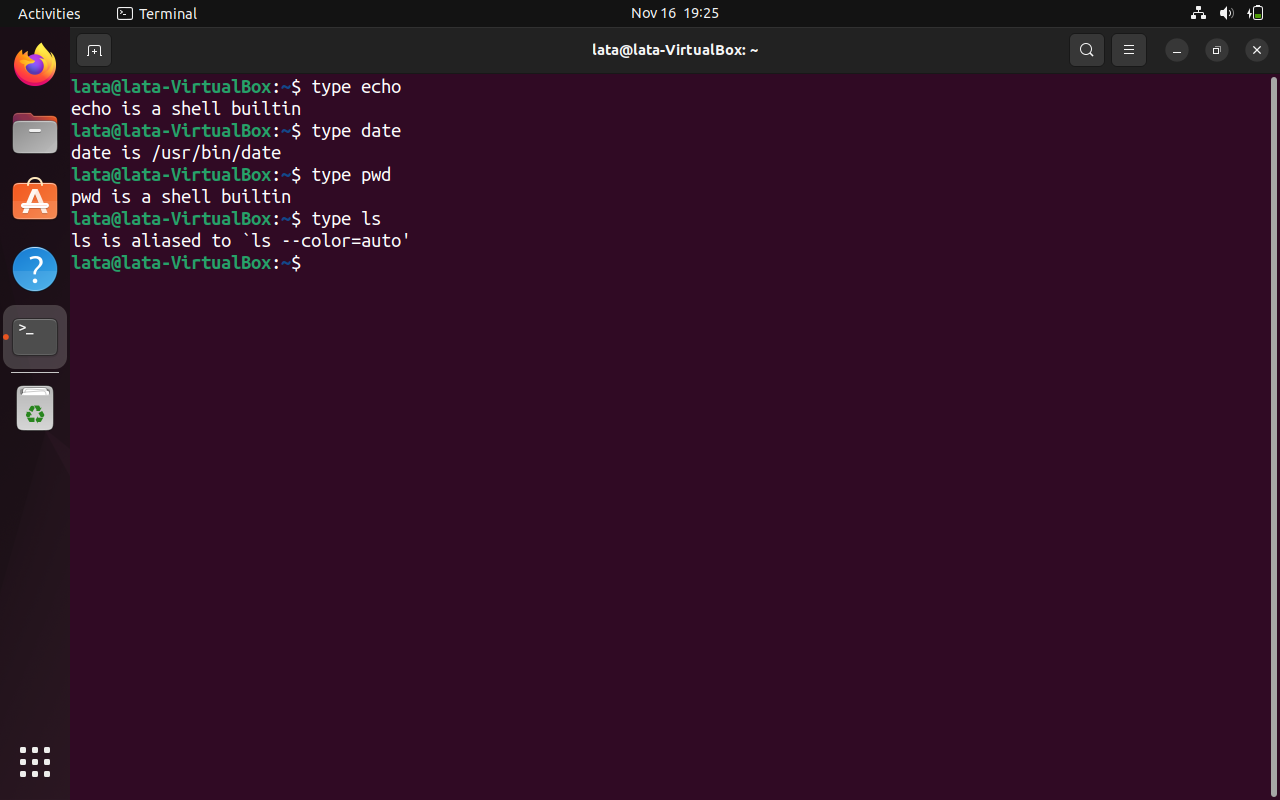
|  |  |
| --- | --- |
| **P7** | Run ***$ whoami*** and then interpret the output. |

Ans:



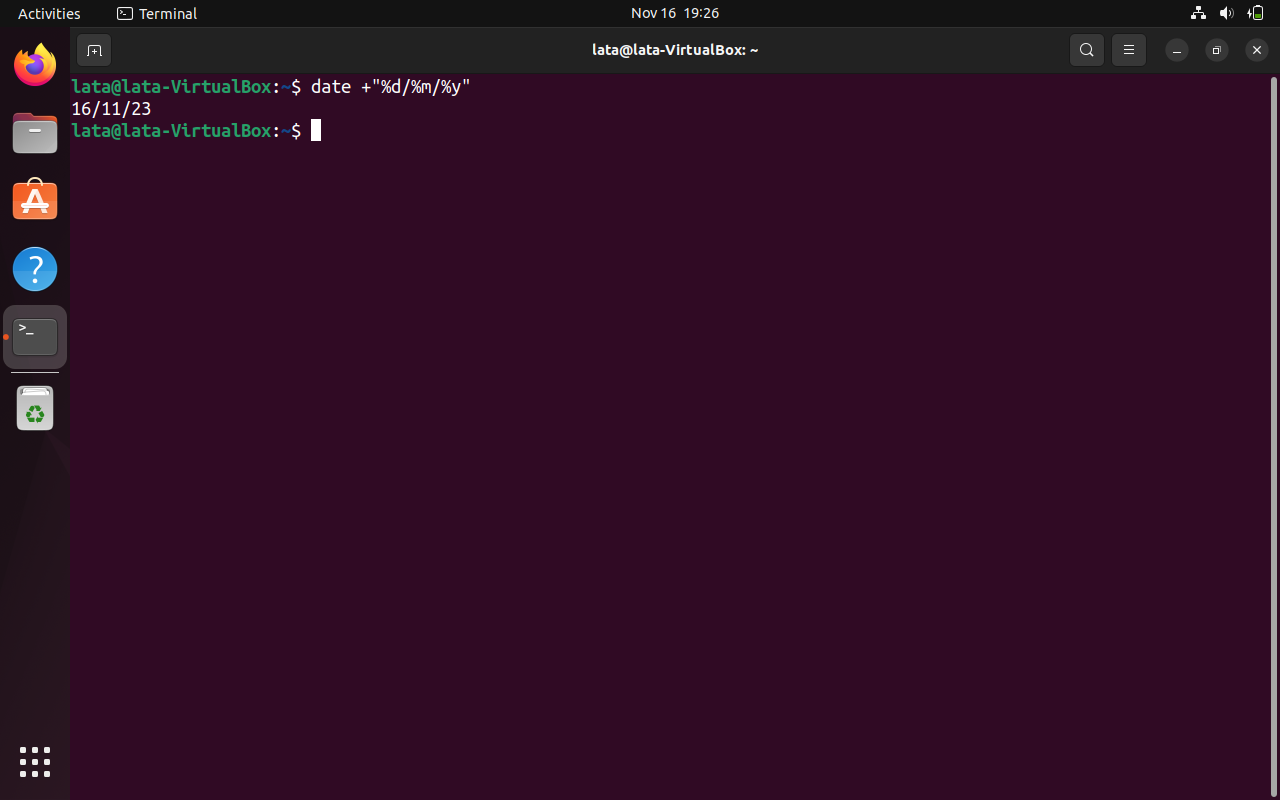
|  |  |
| --- | --- |
| **P8** | Find out whether the following commands are internal or external:  **echo**  **date**  **pwd**  **ls** |
|  | 1. **echo**: Internal command. 2. **date**: External command. 3. **pwd**: External command. 4. **ls**: External command. |

Ans:



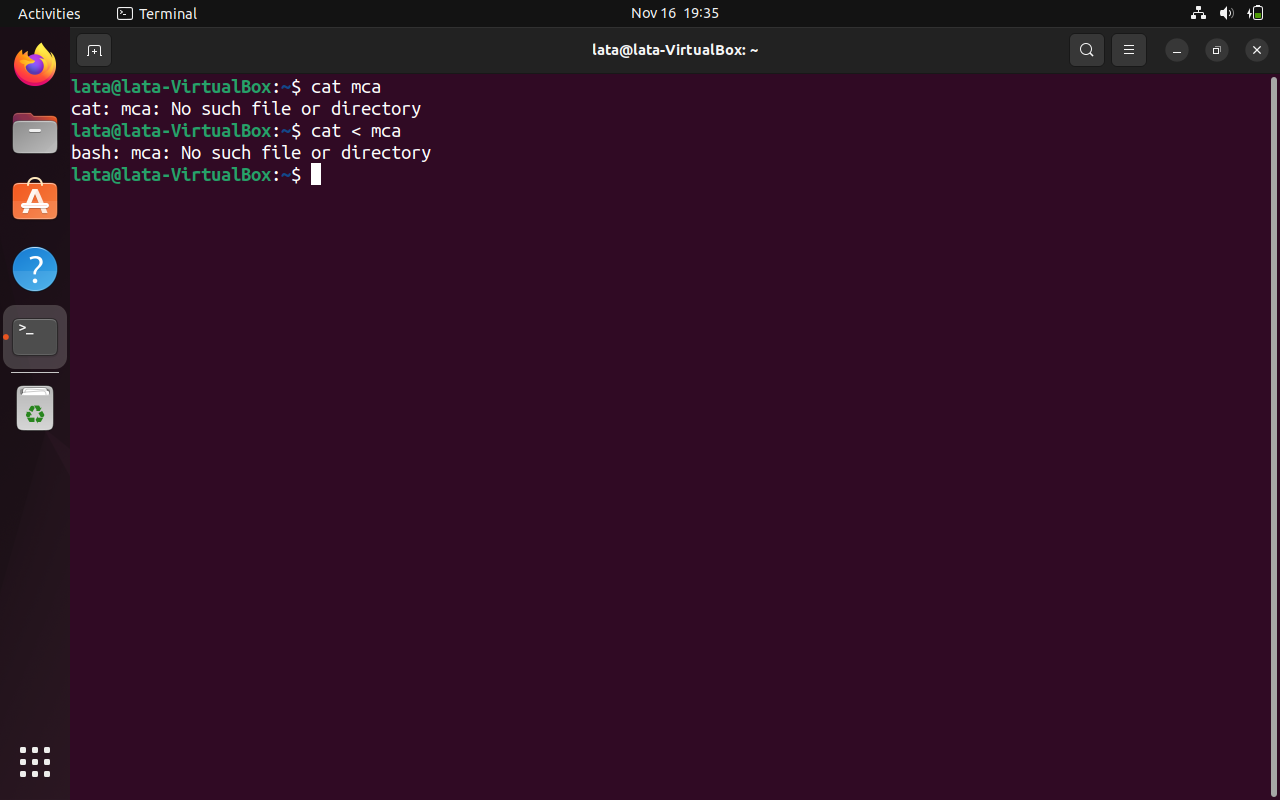
|  |  |
| --- | --- |
| **P9** | Display the current date in the form dd/mm/yyyy. |

Ans:



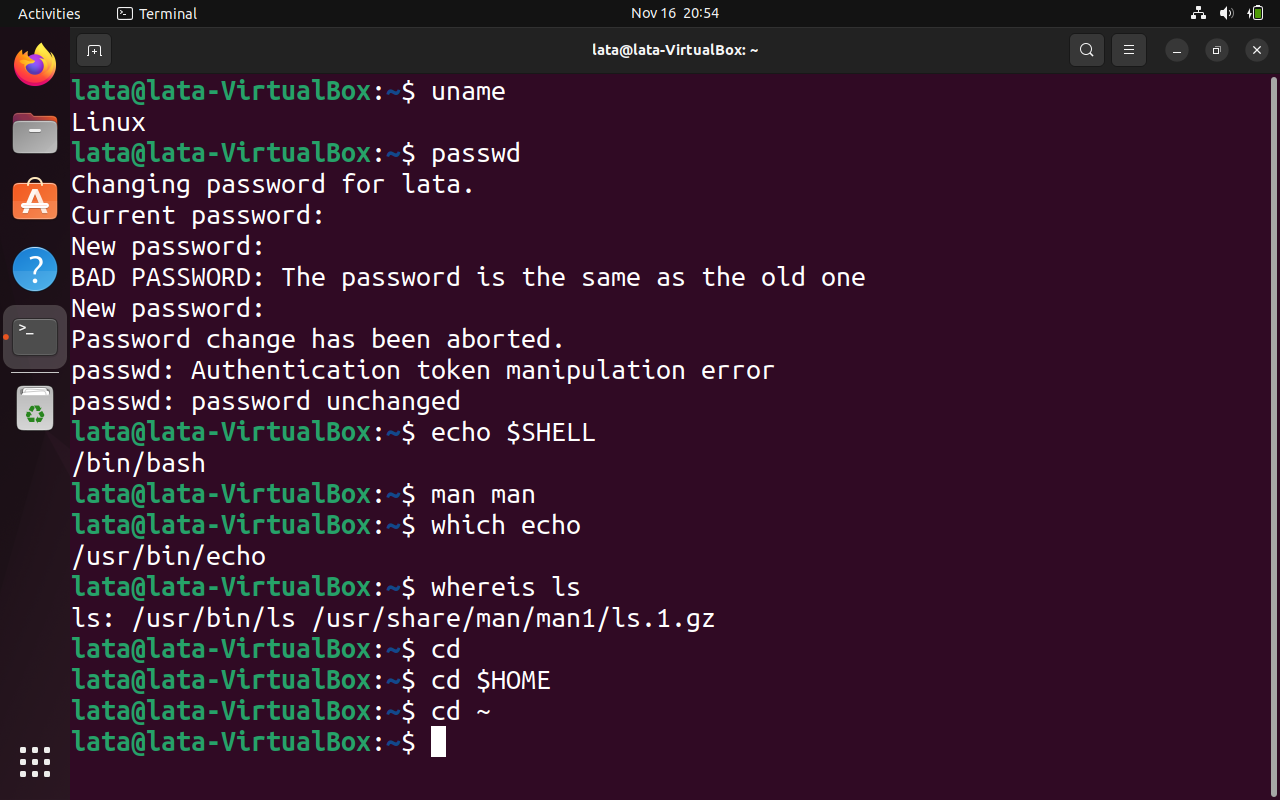
|  |  |
| --- | --- |
| **P10** | Both of the following commands try to open the file mca, but the error messages are a little different. What could be the reason?  *$ cat mca*  *cat: mca: No such file or directory*  *$ cat < mca*  *bash: mca: No such file or directory* |
|  | 1. The error messages indicate that the file "mca" is not found in the current directory for both commands. The absence of the file or a typo in the filename might be the reason for the errors.  2. The first command (`$ cat mca`) tries to display the contents of the file "mca," resulting in the error "cat: mca: No such file or directory." The second command (`$ cat < mca`) attempts input redirection but encounters the same error, "bash: mca: No such file or directory," indicating that the file is not present in the specified location. |

Ans:



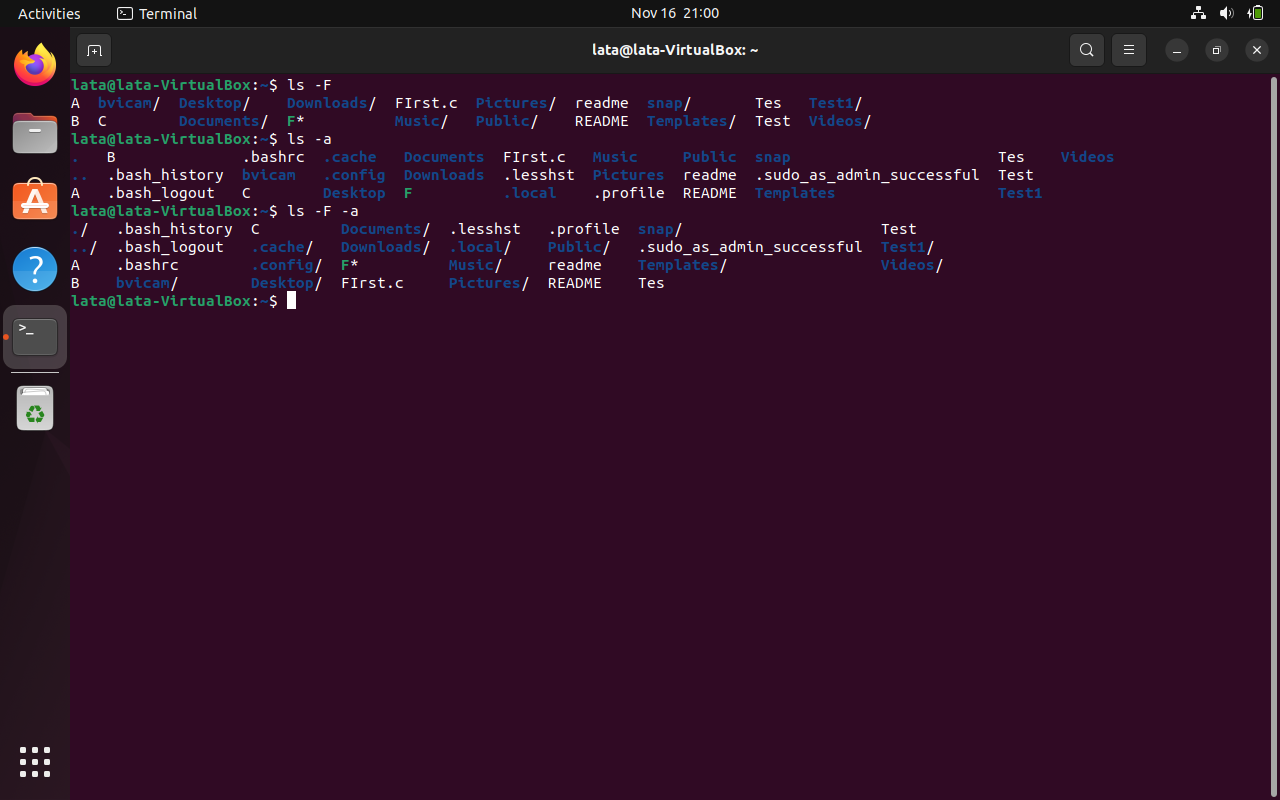
|  |  |  |
| --- | --- | --- |
| **P11** | Run the following commands, and discuss their output? | |
|  | **$ uname**  **$ passwd**  **$ echo $SHELL**  **$ man man**  **$ which echo** | **$ type echo**  **$ whereis ls**  **$ cd**  **$ cd $HOME**  **$ cd ~** |
|  | uname  passwd  echo $SHELL  man man  q  which echo  whereis ls  cd  cd $HOME  cd ~ | |

Ans:



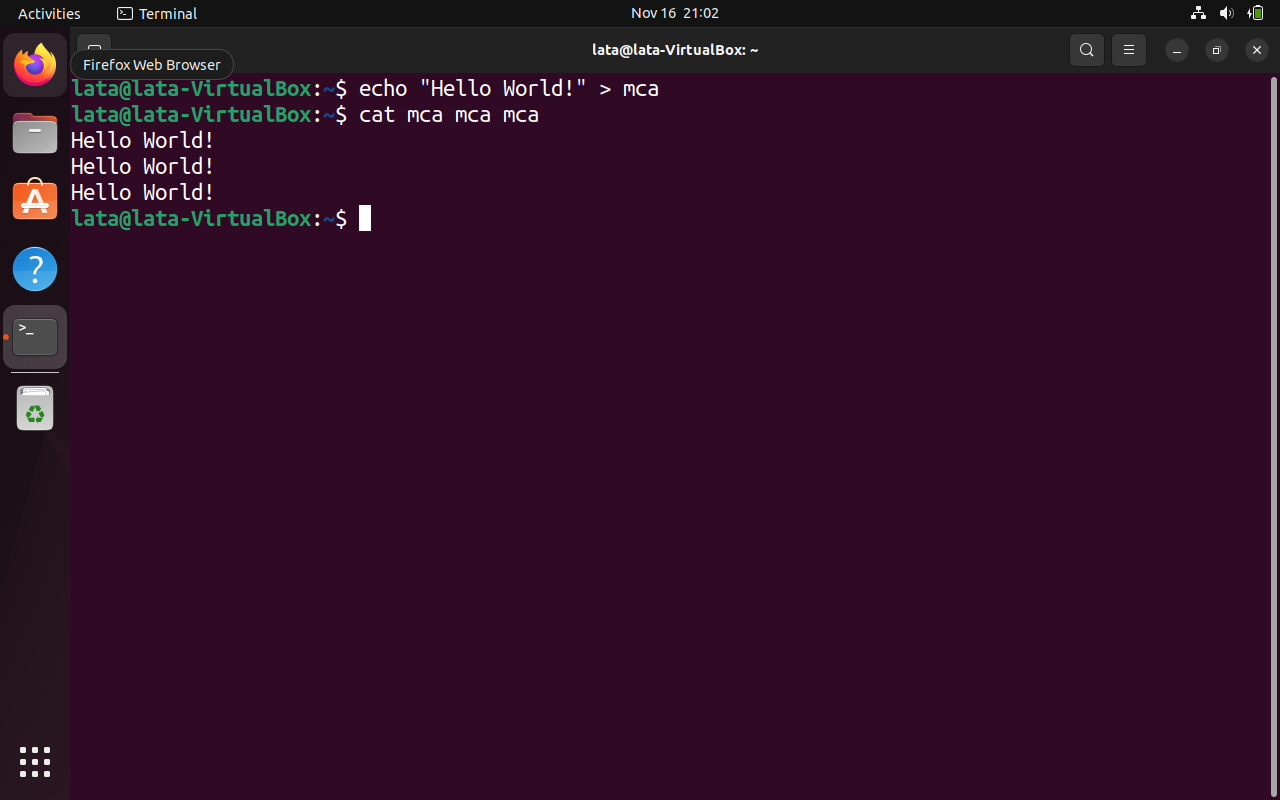
|  |  |
| --- | --- |
| **P12** | Frame **ls** command to  (i) mark directories and executables separately, and  (ii) also display hidden files. |

Ans:



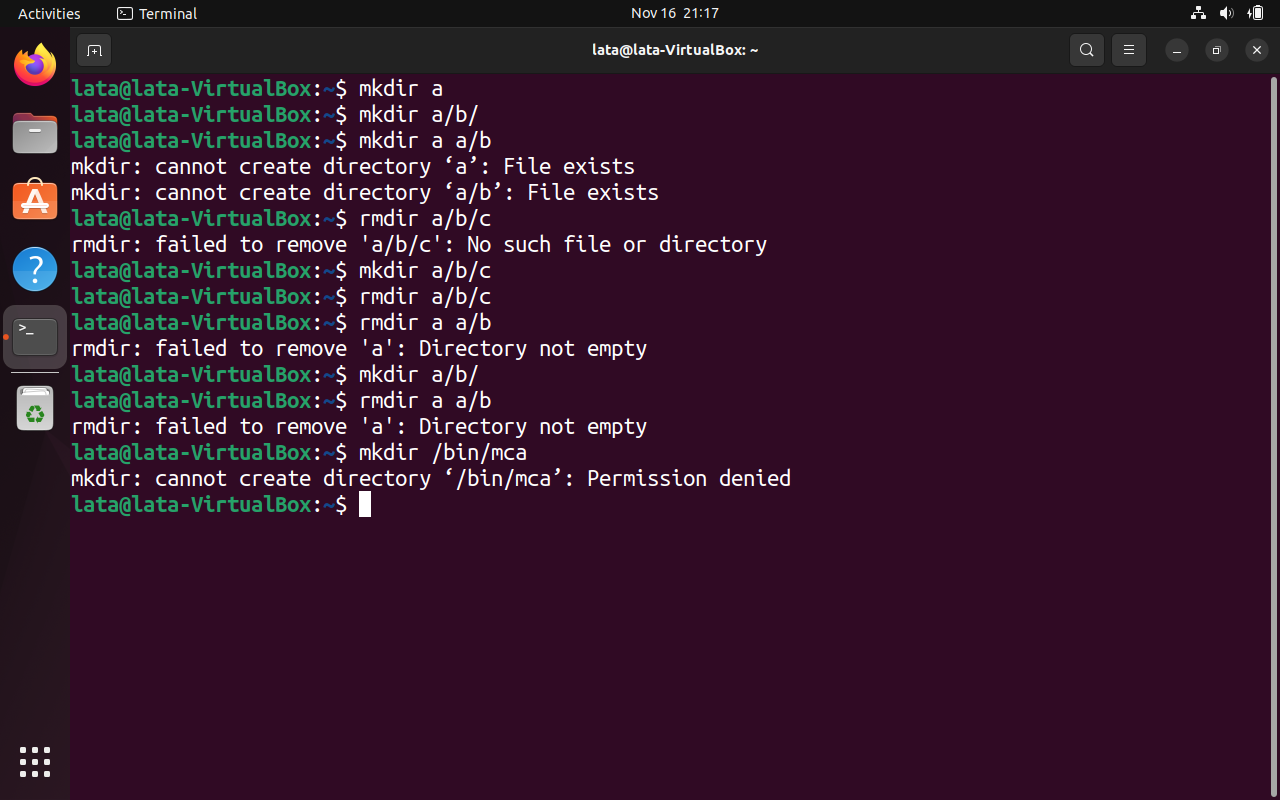
|  |  |
| --- | --- |
| **P13** | Find out the result of following: ***$ cat mca mca mca*** |

Ans:



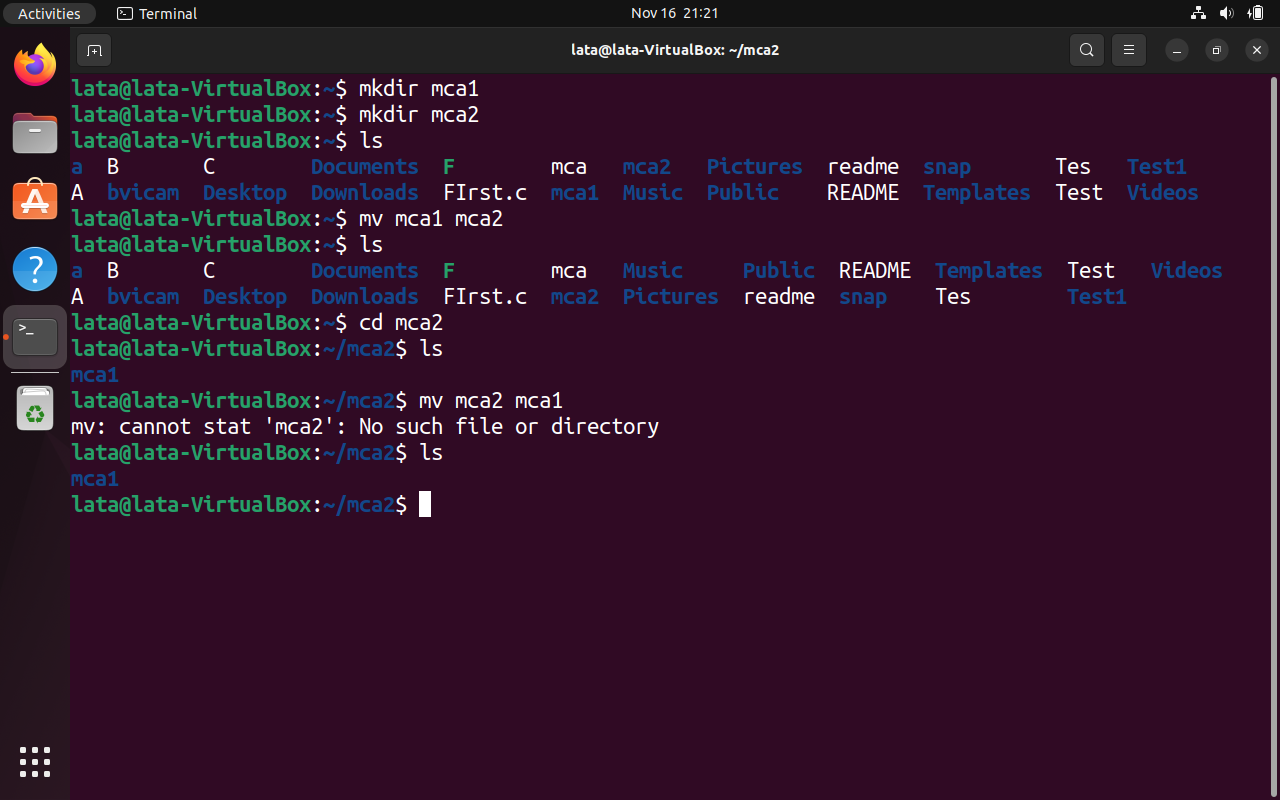
|  |  |
| --- | --- |
| **P14** | Run the following and determine which commands will work? Explain with reasons.   1. **$ mkdir a/b/** 2. **$ mkdir a a/b** 3. **$ rmdir a/b/c** 4. **$ rmdir a a/b** 5. **$ mkdir /bin/mca** |
|  | (i) $ mkdir a/b/  - This command creates a directory named `b` inside the directory `a`. It **will work** as expected.  (ii) $ mkdir a a/b  - This command tries to create two directories: `a` and `a/b`. The first directory (`a`) will be created successfully, but the second (`a/b`) will not be created as the directory `a` does not exist yet. To create nested directories, you should create the parent directory first. Therefore, this command will **partially work**.  (iii) $ rmdir a/b/c  - This command attempts to remove the directory `c` inside the directory `b` inside the directory `a`. If the directory structure exists as specified, it will work. However, if any of the directories (`a`, `b`, or `c`) does not exist, it will fail. Ensure that the directory structure exists before using `rmdir`.  =>**Will work if the directory structure exists; otherwise, it will fail**  (iv) $ rmdir a a/b  - This command tries to remove two directories: `a` and `a/b`. The first directory (`a`) will be removed successfully, but the second (`a/b`) will not be removed as it is no longer present after the removal of `a`. Therefore, this command will **partially work**.  (v) $ mkdir /bin/mca  - This command creates a directory named `mca` inside the `/bin` directory. It will work if the user executing the command has the necessary permissions to create directories in the `/bin` directory. Ensure that you have the required permissions.  => **Will work if the user has the necessary permissions to create directories in `/bin`.** |

Ans:



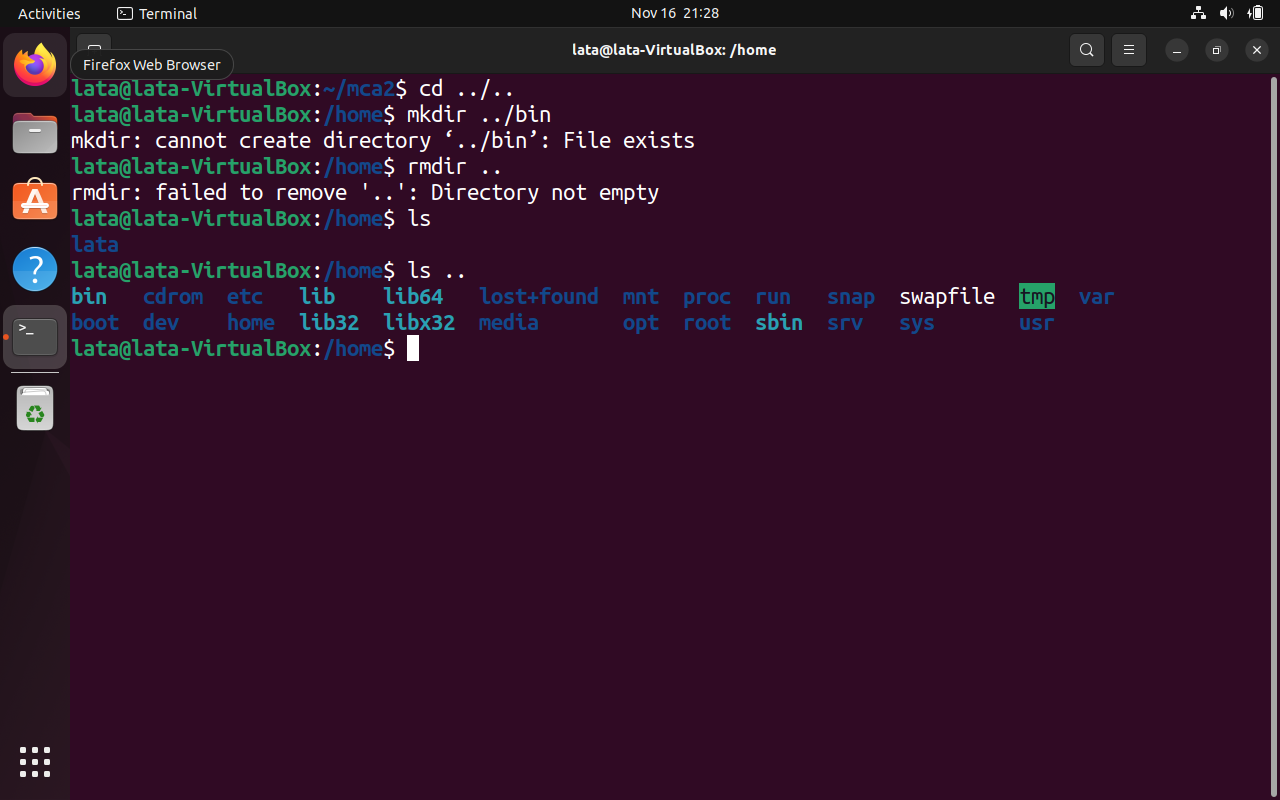
|  |  |
| --- | --- |
| **P15** | How does the command **mv mca1 mca2** behave, where both mca1 and mca2 are directories, when  (i) mca2 exists  (ii) mca2 doesn‟t exist? |
|  | * (i) If **mca2** exists, the contents of **mca1** will be moved into **mca2**. * (ii) If **mca2** doesn't exist, **mca1** will be renamed to **mca2** |

Ans:



|  |  |
| --- | --- |
| **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:   1. **$ cd ../..** 2. **$ mkdir ../bin** 3. **$ rmdir ..** 4. **$ ls ..** |
|  | (i) $ cd ../..  - This command attempts to move two levels up in the directory structure. If the structure allows it (if `/home` exists), it will change the current directory to `/home`.  (ii) $ mkdir ../bin  - This command attempts to create a directory named `bin` in the parent directory of the current directory (`/home`). It will work if the user has the necessary permissions to create a directory in `/home`.  (iii) $ rmdir ..  - This command attempts to remove the parent directory of the current directory (`/home/bvicam`). It will likely fail, as you cannot remove the parent directory of the current working directory while you are in it.  (iv) $ ls ..  - This command lists the contents of the parent directory (`/home`). It will work, showing the files and directories present in `/home`. |

Ans:



|  |  |
| --- | --- |
| **P17** | Apply **Peterson algorithm** for solving the critical section problem with C/Java multi-threaded programming. Assume appropriate code snippet for critical section. |
|  | #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  #include <pthread.h>  #include <unistd.h>  bool flag[2] = {false, false};  int turn;  void \*Process (void \*threadid) {  int wait = rand() % 5 + 1;  sleep(wait);  long myturn;  myturn = (long)threadid;  long other = (myturn + 1) % 2;  flag[myturn] = true;  turn = other;  while (flag[other] && turn == other);  wait = rand() % 5 + 1;  printf("Thread %ld is in critical section and will take %ds of time \n", myturn, wait);  sleep(wait);  printf("Thread %ld is out of critical section \n", myturn);  flag[myturn] = false;  pthread\_exit(NULL);  }  int main (int argc, char \*argv[]) {  // Peterson algorithm  pthread\_t threads[2];  int rc;  int i;  for (i = 0; i < 2; i++) {  printf("In main: creating thread %d \n", i);  rc = pthread\_create(&threads[i], NULL, Process, (void \*)i);  if (rc) {  printf("ERROR; return code from pthread\_create() is %d \n", rc);  exit(-1);  }  }  pthread\_exit(NULL);  } |
|  |  |

|  |  |
| --- | --- |
| **P18** | Apply **Bakery algorithm** for synchronization of processes/threads in a C/Java program. Assume appropriate code snippet for critical section. |
|  | #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  #include <pthread.h>  #include <unistd.h>  #define NUM\_THREADS 3  #define MAX 2  bool choosing[NUM\_THREADS];  int number[NUM\_THREADS];  void \*Process(void \*threadid) {  int runs = 0;  int myturn = (int)threadid;  do {  choosing[myturn] = true;  int max = 0;  for (int i = 0; i < NUM\_THREADS; i++) {  if (number[i] > max) {  max = number[i];  }  }  number[myturn] = max + 1;  choosing[myturn] = false;  for (int i = 0; i < NUM\_THREADS; i++) {  while (choosing[i]);  while (number[i] != 0 && (number[i] < number[myturn] || (number[i] == number[myturn] && i < myturn)));  }  int wait = rand() % 5 + 1;  printf("Thread %d is in critical section and will take %ds of time \n", myturn, wait);  sleep(wait);  printf("Thread %d is out of critical section \n", myturn);  number[myturn] = 0;  runs++;  } while (runs < MAX);  pthread\_exit(NULL);  }  int main(int argc, char const \*argv[]) {  pthread\_t threads[NUM\_THREADS];  int rc;  for(int i = 0; i < NUM\_THREADS; i++) {  choosing[i] = false;  number[i] = 0;  }  for (int i = 0; i < NUM\_THREADS; i++) {  rc = pthread\_create(&threads[i], NULL, Process, (void \*)i);  if (rc) {  printf("ERROR; return code from pthread\_create() is %d \n", rc);  exit(-1);  }  }  pthread\_exit(NULL);  } |
|  |  |

|  |  |
| --- | --- |
| **P19** | Write C/Java program to simulate and solve the **Producer-Consumer problem.** |
|  | #include <stdlib.h>  #include <stdio.h>  #include <pthread.h>  #include <unistd.h>  #define BUFFER\_SIZE 5  #define MAX\_LOOP 5  int buffer[BUFFER\_SIZE];  int semaphore\_mutex = 1;  int semaphore\_empty = BUFFER\_SIZE;  int semaphore\_full = 0;  int in = 0, out = 0;  void Produce(int \*item) {  int wait = rand() % 2 + 1;  sleep(wait);  \*item = rand() % 50000;  printf("Produced item #%d \n", \*item);  }  void Consume(int \*item) {  int wait = rand() % 2 + 1;  sleep(wait);  printf("Consumed item #%d \n", \*item);  }  void wait(int \*semaphore) {  while(\*semaphore <= 0);  \*semaphore = \*semaphore - 1;  }  void signal(int \*semaphore) {  \*semaphore = \*semaphore + 1;  }  void \*Producer(void \*threadid) {  printf("Producer is running \n");  int item;  for(int i = 0; i < MAX\_LOOP; i++) {  Produce(&item);  wait(&semaphore\_empty);  wait(&semaphore\_mutex);  buffer[in] = item;  in = (in + 1) % BUFFER\_SIZE;  signal(&semaphore\_mutex);  signal(&semaphore\_full);  }  pthread\_exit(NULL);  }  void \*Consumer(void \*threadid) {  printf("Consumer is running \n");  for(int i = 0; i < MAX\_LOOP; i++) {  wait(&semaphore\_full);  wait(&semaphore\_mutex);  int item = buffer[out];  out = (out + 1) % BUFFER\_SIZE;  signal(&semaphore\_mutex);  signal(&semaphore\_empty);  Consume(&item);  }  pthread\_exit(NULL);  }  int main(int argc, char \*argv[]) {  pthread\_t threads[2];  int rc;  rc = pthread\_create(&threads[0], NULL, Producer, NULL);  if (rc) {  printf("ERROR; return code from pthread\_create() is %d \n", rc);  exit(-1);  }  rc = pthread\_create(&threads[1], NULL, Consumer, NULL);  if (rc) {  printf("ERROR; return code from pthread\_create() is %d \n", rc);  exit(-1);  }  pthread\_exit(NULL);  } |
|  |  |

|  |  |  |
| --- | --- | --- |
| **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. | |
|  | BP4.c | hashTable.h |
|  | #include "hashTable.h"  #include <pthread.h>  #include <unistd.h>  #define SIZE 5  #define MAX\_WRITE 2  #define MAX\_READ 2  #define MAX\_THREAD 4  hashTable\* map;  int readcount = 0;  int semaphore\_mutex = 1;  int semaphore\_write = 1;  int rc;  void wait(int \*semaphore) {  while(\*semaphore <= 0);  \*semaphore = \*semaphore - 1;  }  void signal(int \*semaphore) {  \*semaphore = \*semaphore + 1;  }  void Write(long id) {  int wait = rand() % 2 + 1;  sleep(wait);  int key = rand() % SIZE + 1;  int value = rand() % 100 + 1;  setHashTable(map, key, value);  printf("Writer #%ld key %d with value %d \n", id, key, value);  }  void \*Writer(void \*threadid) {  int write = 0;  do {  wait(&semaphore\_write);  Write((long)threadid);  write++;  signal(&semaphore\_write);  usleep(100);  } while(write <= MAX\_WRITE);  pthread\_exit(NULL);  }  void Read(long id) {  int wait = rand() % 2 + 1;  sleep(wait);  int key = rand() % SIZE + 1;  int value = lookupHashTable(map, key);  printf("Reader #%ld key %d with value %d \n", id, key, value);  }  void \*Reader(void \*threadid) {  int reads = 0;  do {  wait(&semaphore\_mutex);  readcount++;  if(readcount == 1)  wait(&semaphore\_write);  signal(&semaphore\_mutex);  Read((long)threadid);  wait(&semaphore\_mutex);  reads++;  readcount--;  if(readcount == 0)  signal(&semaphore\_write);  signal(&semaphore\_mutex);  usleep(100);  } while(reads < MAX\_READ);  pthread\_exit(NULL);  }  int main (int argc, char \*argv[]) {  pthread\_t threads[MAX\_THREAD];  map = newHashTable(SIZE);  for(int i = 0; i < SIZE; i++)  insertHashTable(map, i + 1, 1);  for(int i = 0; i < MAX\_THREAD / 2; i++)  rc = pthread\_create(&threads[i + MAX\_THREAD / 2], NULL, Writer, (void \*)(i + MAX\_THREAD / 2));  for(int i = 0; i < MAX\_THREAD / 2; i++)  rc = pthread\_create(&threads[i], NULL, Reader, (void \*)i);    for(int i = 0; i < MAX\_THREAD; i++)  pthread\_join(threads[i], NULL);  freeHashTable(map);  pthread\_exit(NULL);  } | #ifndef HASHTABLE\_H  #define HASHTABLE\_H  #include <stdbool.h>  #include <stdlib.h>  #include <stdio.h>  #include <assert.h>  typedef struct hashTable hashTable;  struct hashTable {  int size;  int \*keys;  int \*values;  };  hashTable\* newHashTable(int size) {  hashTable\* ht = (hashTable\*) malloc(sizeof(struct hashTable));  assert(ht != NULL);  ht->size = size;  ht->keys = (int \*) malloc(size \* sizeof(int));  assert(ht->keys != NULL);  ht->values = (int \*) malloc(size \* sizeof(int));  assert(ht->values != NULL);  return ht;  }  void freeHashTable(hashTable\* ht) {  free(ht->keys);  free(ht->values);  free(ht);  }  void insertHashTable(hashTable\* ht, int key, int value) {  int i = 0;  while (i < ht->size && ht->keys[i] != 0) {  if (ht->keys[i] == key) {  ht->values[i] = value;  return;  }  i++;  }  if (i == ht->size) {  printf("Hash table is full \n");  return;  }  ht->keys[i] = key;  ht->values[i] = value;  }  int lookupHashTable(hashTable\* ht, int key) {  int i = 0;  while (i < ht->size && ht->keys[i] != 0) {  if (ht->keys[i] == key) {  return ht->values[i];  }  i++;  }  printf("Key not found \n");  return 0;  }  void deleteHashTable(hashTable\* ht, int key) {  int i = 0;  while (i < ht->size && ht->keys[i] != 0) {  if (ht->keys[i] == key) {  ht->keys[i] = 0;  ht->values[i] = 0;  return;  }  i++;  }  printf("Key not found \n");  }  bool inHashTable(hashTable\* ht, int key) {  int i = 0;  while (i < ht->size && ht->keys[i] != 0) {  if (ht->keys[i] == key) {  return true;  }  i++;  }  return false;  }  void setHashTable(hashTable\* ht, int key, int value) {  int i = 0;  while (i < ht->size && ht->keys[i] != 0) {  if (ht->keys[i] == key) {  ht->values[i] = value;  return;  }  i++;  }  printf("Key not found \n");  } |
|  |  | |

|  |  |
| --- | --- |
| **P21** | Create a **zombie process** and an **orphan process** in a “C” program with appropriate system calls. |
|  | #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <sys/wait.h>  int main() {  // Create a zombies process  pid\_t zombies\_pid = fork();  if (zombies\_pid == 0) {  // Child process  printf("Child process (zombie) with pid %d\n", getpid());  exit(0);  }  else {  // Parent process  printf("Parent process with pid %d\n", getpid());  sleep(500);  wait(NULL);  }  // Create an orphan process  pid\_t orphan\_pid = fork();  if (orphan\_pid == 0) {  // Child process  printf("Child process (orphan) with pid %d\n", getpid());  sleep(500);  }  else {  // Parent process  printf("Parent process with pid %d\n", getpid());  exit(0);  }  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. |
|  | #include <stdio.h>  #include <unistd.h>  #include <sys/wait.h>  int main(int argc, char const \*argv[]) {  int id = fork();  if (id == 0) {  printf("Child process with pid %d\n", getpid());  sleep(10);  printf("Child process with pid %d is done\n", getpid());  return 0;  }  else {  printf("Parent process with pid %d\n", getpid());  sleep(5);  wait(NULL);  printf("Parent process with pid %d is done\n", getpid());  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 another program (B.c). The logic of program “B.c” is to generate **all the prime numbers** within the specified limit. | |
|  | **A.c** | **B.c** |
|  | #include <stdio.h>  #include <stdlib.h>  #include <sys/types.h>  #include <sys/wait.h>  #include <unistd.h>  int main() {  pid\_t child\_pid;  // Fork a child process  if ((child\_pid = fork()) == -1) {  perror("fork");  exit(EXIT\_FAILURE);  }  // Code for the parent process  if (child\_pid > 0) {  // Wait for the child process to complete  wait(NULL);  printf("Parent process exiting.\n");  }  // Code for the child process  else if (child\_pid == 0) {  // Execute the code from B.c  execlp("./B", "./B", (char \*)NULL);  // The following code is executed only if execlp fails  perror("execlp");  exit(EXIT\_FAILURE);  }  return 0;  } | #include <stdio.h>  int is\_prime(int num) {  if (num < 2) {  return 0; // Not prime  }  for (int i = 2; i \* i <= num; ++i) {  if (num % i == 0) {  return 0; // Not prime  }  }  return 1; // Prime  }  int main() {  int limit;  printf("Enter the limit for prime numbers: ");  scanf("%d", &limit);  printf("Prime numbers up to %d:\n", limit);  for (int i = 2; i <= limit; ++i) {  if (is\_prime(i)) {  printf("%d\n", i);  }  }  return 0;  } |
|  |  | |

|  |  |
| --- | --- |
| **P24** | Write an appropriate “C” program which implements the concept of **dynamic memory allocation** (use of malloc(), calloc(), realloc(), and free() system call). |
|  | #include <stdio.h>  #include <stdlib.h>  int main() {  // Use malloc() to allocate memory for an integer  int \*ptr1 = (int \*) malloc(sizeof(int));  \*ptr1 = 10;  printf("Value at ptr1: %d\n", \*ptr1);  // Use calloc() to allocate memory for an array of 5 integers  int \*ptr2 = (int \*) calloc(5, sizeof(int));  for (int i = 0; i < 5; i++) {  ptr2[i] = i + 1;  }  printf("Values at ptr2: ");  for (int i = 0; i < 5; i++) {  printf("%d ", ptr2[i]);  }  printf("\n");  // Use realloc() to change the size of the memory allocated for ptr2  ptr2 = (int \*) realloc(ptr2, 10 \* sizeof(int));  for (int i = 5; i < 10; i++) {  ptr2[i] = i + 1;  }  printf("Values after realloc at ptr2: ");  for (int i = 0; i < 10; i++) {  printf("%d ", ptr2[i]);  }  printf("\n");  // Use free() to deallocate memory  free(ptr1);  free(ptr2);  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. | |
|  | **BP9.c** | **courses.txt** |
|  | #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/wait.h>  int main() {  pid\_t child\_pids[3];  FILE \*file;  char line[100];  // Open the file  file = fopen("courses.txt", "r");  if (file == NULL) {  perror("Error opening file");  return 1;  }  // Fork child processes  for (int i = 0; i < 3; i++) {  fgets(line, sizeof(line), file);  child\_pids[i] = fork();  if (child\_pids[i] == 0) {  // Child process  printf("Child %d: %s", i+1, line);  exit(0);  }  }  // Wait for child processes to finish  for (int i = 0; i < 3; i++) {  waitpid(child\_pids[i], NULL, 0);  }  // Close the file  fclose(file);  return 0;  } | Java Programming  Operating System  Discrete Structure |
|  |  | |

|  |  |
| --- | --- |
| **P26** | Write an appropriate “C” program which implements the concept of **dynamic memory allocation** (use of malloc(), calloc(), realloc(), and free() system call). |
|  | #include <stdlib.h>  #include <stdio.h>  void printarr(int \*array, int size) {  printf("[");  for(int i = 0; i < size - 1; i++)  printf("%d, ", array[i]);  printf("%d]\n", array[size - 1]);  }  int main() {  // Allocating array of size 5 using malloc  int \*arr1 = (int \*) malloc(sizeof(int) \* 5);  for(int i = 0; i < 5; i++) arr1[i] = i;  printarr(arr1, 5);  // increasing the size using realloc  arr1 = (int \*) realloc(arr1, sizeof(int) \* 10);  for(int i = 5; i < 10; i++) arr1[i] = i;  printarr(arr1, 10);  // Allocating array of size 10 using calloc  // It Initialize all the values to 0 unlike malloc  int \*arr2 = (int \*) calloc(10, sizeof(int));  printarr(arr2, 10);  // free(void \*\_ptr) is used to free the memory  free(arr1);  free(arr2);  } |
|  |  |

|  |  |
| --- | --- |
| **P27** | 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. |
|  | #include <stdlib.h>  #include <stdio.h>  void printarr(int \*array, int size) {  printf("[");  for(int i = 0; i < size - 1; i++)  printf("%d, ", array[i]);  printf("%d]\n", array[size - 1]);  }  int main() {  // Allocating array of size 5 using malloc  int \*arr1 = (int \*) **malloc**(sizeof(int) \* 5);  for(int i = 0; i < 5; i++) arr1[i] = i;  printarr(arr1, 5);  // increasing the size using realloc  arr1 = (int \*) **realloc**(arr1, sizeof(int) \* 10);  for(int i = 5; i < 10; i++) arr1[i] = i;  printarr(arr1, 10);  // Allocating array of size 10 using calloc  // It Initialize all the values to 0 unlike malloc  int \*arr2 = (int \*) **calloc**(10, sizeof(int));  printarr(arr2, 10);  // free(void \*\_ptr) is used to free the memory  free(arr1);  free(arr2);  } |
|  |  |

|  |  |  |
| --- | --- | --- |
| **P28** | 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. | |
|  | **Method - 1** | **Method - 2** |
|  | #include <stdio.h>  #include <stdlib.h>  #include <sys/stat.h>  int main(int argc, char \*argv[]) {  if (argc != 3) {  printf("Usage: %s <file/directory> <permission>\n", argv[0]);  exit(1);  }  char \*file = argv[1];  char \*permission = argv[2];  int mode = strtol(permission, 0, 8);  if (chmod(file, mode) == -1) {  perror("chmod");  exit(1);  }  printf("Access rights of %s changed to %s\n", file, permission);  return 0;  } | #include <stdio.h>  #include <stdlib.h>  #include <sys/stat.h>  int main() {  char filename[100];  unsigned int permissions;  // Get file/directory name from user  printf("Enter the file or directory name: ");  scanf("%s", filename);  // Get new permissions from user  printf("Enter the new permissions (in octal): ");  scanf("%o", &permissions);  // Change the access rights using chmod system call  if (chmod(filename, permissions) == 0) {  printf("Access rights changed successfully.\n");  } else {  perror("Error in changing access rights");  return 1;  }  return 0;  } |
|  |  |  |

|  |  |
| --- | --- |
| **P29** | 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. |
|  | #include <stdlib.h>  #include <stdio.h>  int main() {  char\* arr = (char \*) malloc(26 \* sizeof(char));  for (int i = 0; i < 26; i++)  arr[i] = 'a' + i;  for (int i = 0; i < 24; i += 2)  printf("%c, ", arr[i]);  printf("%c\n", arr[24]);  free(arr);  return 0;  } |
|  |  |

|  |  |
| --- | --- |
| **P30** | 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”. |
|  | #include <stdio.h>  #include <stdlib.h>  struct student {  int rollno;  char name[200];  };  int writef(FILE \*file, struct student \*s) {  int out = fprintf(  file,  "%d\n%s\n",  s->rollno, s->name  );  fflush(file);  return out;  }  int readf(FILE \*file, struct student \*s) {  if (fscanf(file, "%d", &s->rollno) == EOF) return 0;  if (fscanf(file, "%s", s->name) == EOF) return 0;  return 1;  }  int main() {  int n;  printf("Enter the number of students: ");  scanf("%d", &n);  struct student s;  FILE \*file = fopen("Student.txt", "w");  for (int i = 0; i < n; i++) {  printf("Enter the roll no of student %d: ", i);  scanf("%d", &s.rollno);  printf("Enter the name of student %d: ", i);  scanf("%s", s.name);  writef(file, &s);  }  fclose(file);  file = fopen("Student.txt", "r");  int i = 0;  while (readf(file, &s)) {  printf("Student #%d: %s\n", s.rollno, s.name);  if(i == 5) break;  i++;  }  fclose(file);  return 0;  } |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **CP6** | Demonstrate the use of following system calls by writing an appropriate “C” program.  (a) lseek() (b) chmod() (c) umask() (d) access() (e) utime() | | |
|  | (a) lseek() | Sets the file offset for the open file descriptor. | lseek(fd, offset, SEEK\_SET); |
| (b) chmod() | Changes the permissions of a file. | chmod("file.txt", S\_IRWXU | S\_IRGRP | S\_IXGRP | S\_IROTH); |
| (c) umask() | Sets the file mode creation mask. | umask(S\_IWOTH); |
| (d) access() | Checks whether the calling process can access a file. | if (access("file.txt", R\_OK | W\_OK) == 0) { /\* permission granted \*/ } |
| (e) utime() | Changes the access and modification times of a file. | utime("file.txt", &timespec); |
|  | #include <stdlib.h>  #include <stdio.h>  #include <unistd.h>  #include <utime.h>  #include <time.h>  #include <sys/stat.h>  #include <sys/types.h>  struct Ex {  int A;  char B;  };  int writef(FILE \*file, struct Ex \*e) {  int out = fprintf(  file,  "\nA:%d\nB:%c",  e->A, e->B  );  fflush(file);  return out;  }  // using fseek to skip read pointer in the file  int readf(FILE \*file, struct Ex \*e) {  fseek(file, 3, SEEK\_CUR);  if (fscanf(file, "%d", &e->A) == EOF) return 0;  fseek(file, 3, SEEK\_CUR);  if (fscanf(file, "%s", &e->B) == EOF) return 0;  return 1;  }  int print\_time(char\* filename) {  struct stat sb;  if (stat(filename, &sb) == 0) {  printf("Modification time for %s: %s", filename, ctime(&sb.st\_mtime));  printf("Access time for %s: %s", filename, ctime(&sb.st\_atime));  return 1;  }  return 0;  }  int print\_file\_access(char\* filename) {  struct stat sb;  if (stat(filename, &sb) != -1) {  printf("File access mode for file %s: %o\n", filename, sb.st\_mode & 0777);  return 1;  }  return 0;  }  int main() {  // Creating a example file and writing data to it  char\* filename = "example.txt";  struct Ex s;  FILE \*file = fopen(filename, "w");  for (int i = 0; i < 5; i++) {  s.A = i + 1;  s.B = i + 'A';  writef(file, &s);  }  fclose(file);  // lseek -> fseek  printf("\nlseek:\n");  file = fopen(filename, "r");  while (readf(file, &s)) {  printf("A: %d\n", s.A);  printf("B: %c\n", s.B);  }  fclose(file);  filename = "test.txt";  // berfore changing umask  printf("\nno umask:\n");  remove(filename);  fclose(fopen(filename, "w"));  print\_file\_access(filename);  // after changing umask  printf("\numask:\n");  umask(S\_IRUSR | S\_IEXEC);  remove(filename);  fclose(fopen(filename, "w"));  print\_file\_access(filename);  // changing the mode of the file  printf("\nchmode:\n");  chmod(filename, S\_IWUSR | S\_IXUSR);  print\_file\_access(filename);  // Access cheking the read, write access  printf("\naccess:\n");  if (access(filename, R\_OK) == 0) {  printf("File is readable\n");  } else {  printf("File is not readable\n");  }  if (access(filename, W\_OK) == 0) {  printf("File is writable\n");  } else {  printf("File is not writable\n");  }  // Getting the time modifier of a file test.txt  printf("\nBefore Changing Time modifier: \n");  print\_time(filename);  struct utimbuf times;  time\_t current\_time = time(NULL);  times.actime = current\_time + (24 \* 60 \* 60);  times.modtime = current\_time - (24 \* 60 \* 60);  // Changing the time modifier of the file  if (utime(filename, &times) == 0) {  printf("\nAfter Changing Time modifier: \n");  print\_time(filename);  }  return 0;  } | | |
|  |  | | |