Lab Report No-01: How to install Linux operating system

i)What is Linux Operating system?

Linux is the best-known and most-used open source operating system. As an operating system, Linux is software that sits underneath all of the other software on a computer, receiving requests from those programs and relaying these requests to the computer's hardware.

ii)Different type of linux distribution

Different type of linux distribution are given below:

ubuntu

Linux Mint Cinnamon

Zorin OS

Elementary OS

Linux Mint Mate

Manjaro Linux

iii)How can we install linux operating system?

1.Boot into USB Stick

2.Derive Selection

3.Start installation

4.Complete the installation process

Lab Report No-02: Basic Command of linux Operating System

i)What is linux command?

Linux is a Unix-Like operating system. All the Linux/Unix commands are run in the terminal provided by the Linux system. This terminal is just like the command prompt of Windows OS. Linux/Unix commands are *case-sensitive.* The terminal can be used to accomplish all Administrative tasks. This includes package installation, file manipulation, and user management.

ii)Write 15 commands in linux operating system.

1. **pwd command**
2. **cd command**
3. **ls command**
4. **cat command**
5. **mv command**
6. **cp command**
7. **mkdir command**
8. **rmdir command**
9. **rm command**
10. **touch command**
11. **locate command**
12. **find command**
13. **grep command**
14. **sudo command**
15. **df command**

iii)Describe the operation of linux basic command (screenshot)

PrtSc **– *Save a screenshot of the entire screen to the "Pictures" directory.***
Shift + PrtSc **– *Save a screenshot of a specific region to Pictures.***
Alt + PrtSc **– *Save a screenshot of the current window to Pictures*.**
Ctrl + PrtSc **– *Copy the screenshot of the entire screen to the clipboard.***
Shift + Ctrl + PrtSc **– *Copy the screenshot of a specific region to the clipboard.***
Ctrl + Alt + PrtSc **– *Copy the screenshot of the current window to the clipboard*.**

Lab Report No-03: Threads on operating system.

i)What is thread?

A thread is a path of execution within a process. A process can contain multiple threads

The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces.
Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.

ii)Types of threads

1.User level Thread(ULT)

2.Kernel level Thread(KLT)

iii)Implementation of threads.

The Linux Implementation of Threads

Threads are a popular modern programming abstraction. They provide multiple threads of execution within the same program in a shared memory address space. They can also share open files and other resources. Threads allow for *concurrent programming* and, on multiple processor systems, true *parallelism.*

Linux has a unique implementation of threads. To the Linux kernel, there is *no* concept of a thread. Linux implements all threads as standard processes. The Linux kernel does not provide any special scheduling semantics or data structures to represent threads. Instead, a thread is merely a process that shares certain resources with other processes. Each thread has a unique `task_struct` and appears to the kernel as a normal process (which just happens to share resources, such as an address space, with other processes).
This approach to threads contrasts greatly with operating systems such as Microsoft Windows or Sun Solaris, which have *explicit* kernel support for threads (and sometimes call threads *lightweight processes*). The name "lightweight process" sums up the difference in philosophies between Linux and other systems. To these other operating systems, threads are an abstraction to provide a lighter, quicker execution unit than the heavy process. To Linux, threads are simply a manner of sharing resources between processes (which are already quite lightweight). For example, assume you have a process that consists of four threads. On systems with explicit thread support, there might exist one process descriptor that in turn points to the four different threads. The process descriptor describes the shared resources, such as an address space or open files. The threads then describe the resources they alone possess. Conversely, in Linux, there are simply four processes and thus four normal `task_struct` structures. The four processes are set up to share certain resources.

Threads are created like normal tasks, with the exception that the `clone()` system call is passed flags corresponding to specific resources to be shared:

```
clone(CLONE_VM | CLONE_FS | CLONE_FILES | CLONE_SIGHAND, 0);
```

The previous code results in behavior identical to a normal `fork()`, except that the address space, filesystem resources, file descriptors, and signal handlers are shared. In other words, the new task and its parent are what are popularly called *threads*.
In contrast, a normal `fork()` can be implemented as

```
clone(SIGCHLD, 0);
```

And `vfork()` is implemented as

```
clone(CLONE_VFORK | CLONE_VM | SIGCHLD, 0);
```

Kernel Threads

It is often useful for the kernel to perform some operations in the background. The kernel accomplishes this via *kernel threads*—standard processes that exist solely in kernel-space. The significant difference between kernel threads and normal processes is that kernel threads do not have an address space (in fact, their `mm` pointer is `NULL`). They operate only in kernel-space and do not context switch into user-space. Kernel threads are, however, schedulable and preemptable as normal processes.
Linux delegates several tasks to kernel threads, most notably the *pdflush* task and the *ksoftirqd* task. These threads are created on system boot by other kernel threads. Indeed, a kernel thread can be created only by another kernel thread. The interface for spawning a new kernel thread from an existing one is

```
int kernel_thread(int (*fn)(void *), void * arg, unsigned long
flags)
```

The new task is created via the usual `clone()` system call with the specified `flags` argument. On return, the parent kernel thread exits with a pointer to the child's `task_struct`. The child executes the function specified by `fn` with the given argument `arg`. A special clone flag, `CLONE_KERNEL`, specifies the usual flags for kernel threads: `CLONE_FS`, `CLONE_FILES`, and `CLONE_SIGHAND`. Most kernel threads pass this for their `flags` parameter.

Typically, a kernel thread continues executing its initial function forever (or at least until the system reboots, but with Linux you never know). The initial function usually implements a loop in which the kernel thread wakes up as needed, performs its duties, and then returns to sleep.

Lab Report No- 04: File Operation and PermissionW

i)What is File Operation and File Permission in Linux Operating System?

File ownership is an important component of Unix that provides a secure method for storing files. Every file in Unix has the following attributes −

- **Owner permissions** − The owner's permissions determine what actions the owner of the file can perform on the file.
- **Group permissions** − The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
- **Other (world) permissions** − The permissions for others indicate what action all other users can perform on the file.

ii) Implementations of file operation and file permission.

To change directory permissions in Linux, use the following:

1. chmod +rwx filename to add **permissions**.
2. chmod -rwx directoryname to remove **permissions**.
3. chmod +x filename to **allow** executable **permissions**.
4. chmod -wx filename to take out write and executable.

Lab Report No-05:Connecting a Database(MYSQL) with Linux.

i)Install mysql on Ubuntu.

To install MySQL on your Ubuntu server follow the steps below:

1. First, update the apt package index by typing: sudo apt update.
2. Then **install** the **MySQL** package with the following **command**: sudo apt **install mysql**-server.
3. Once the **installation** is completed, the **MySQL** service will start automatically

ii)Log into Mysql by Linux.

to connect to MySQL from the command line, follow these steps:

1. Log in to A2 Hosting account using SSH.
2. At the command line, type the following command, replacing *USERNAME* with username:

```
mysql -u USERNAME -p
```

3. At the **Enter Password** prompt, type password. When type the correct password, the **mysql>** prompt appears.
4. To display a list of databases, type the following command at the **mysql>** prompt:

```
show databases;
```

5. To access a specific database, type the following command at the **mysql>** prompt, replacing *DBNAME* with the database that want to access:

```
use DBNAME;
```

6. After you access a database, you can run SQL queries, list tables, and so on. Additionally:
   - To view a list of MySQL commands, type help at the **mysql>** prompt.
   - To exit the *mysql* program, type \q at the **mysql>** prompt.

iii)Create Database Table,

Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

iv)Insert Data into Table

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
```

```
    City varchar(255)
);
```

v)Describe table.

Describe table_name;

vi)Alter table

```
ALTER TABLE table_name
ADD column_name datatype;
```

vii)Modify table

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

viii)Drop data From Table

```
DROP TABLE table_name;
```

ix)Update data OF Table

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

x)Delete and Where Operation.

```
DELETE FROM table_name WHERE condition;
```

Lab Report No-06:Linux command For Process

i)How to manage processes from the linux termunal?

The procedure to monitor the running process in Linux using the command line is as follows:

1. Open the terminal window on Linux
2. For remote Linux server use the ssh command for log in purpose

3. Type the ps aux command to see all running process in Linux
4. Alternatively, you can issue the top command or top command to view running process in Linux

ii)

top

top command is used to show the Linux processes. It provides a dynamic real-time view of the running system. Usually, this command shows the summary information of the system and the list of processes or threads which are currently managed by the Linux Kernel.

```
top -n 10
```

htop

htop command in Linux system is a command line utility that allows the user to interactively monitor the system's vital resources or server's processes in real time. *htop* is a newer program compared to top command, and it offers many improvements over top command

```
htop [-dChusv]
```

Ps

Linux provides us a utility called **ps** for viewing information related with the processes on a system which stands as abbreviation for **"Process Status".**

```
ps [options]
```

pstree

- pstree is a **Linux** command that shows the running processes as a tree. It is used as a more visual alternative to the ps command. The root of the tree is either init or the process with the given pid. It can also be installed in other **Unix** systems.
- pstree pid.
- pstree username.

kill

kill command in Linux (located in /bin/kill), is a built-in command which is used to terminate processes manually. *kill* command sends a signal to a process which terminates the process

```
$kill -l
```

pgrep

pgrep is a command-line utility that allows you to find the process IDs of a running program based on given criteria. It can be a full or partial process name, a user running the process, or other attributes.

```
pgrep [OPTIONS] <PATTERN>
```

pkill

```
pkill [OPTIONS] <PATTERN>
```

killall

```
killall [-Z, --context pattern] [-e, --exact] [-g, --
process-group]

        [-i, --interactive] [-o, --older-than TIME]
[-q, --quiet]

        [-r, --regexp] [-s, --signal signal] [-u, --
user user]

        [-v, --verbose] [-w, --wait] [-y, --younger-
than TIME]

        [-I, --ignore-case] [-V, --version] [--] name
```

xkill

```
Xkill is a utility for forcing the X server to close connections to
clients.  This program
       is very dangerous, but is useful for  aborting  programs  that  have
displayed  undesired
       windows  on  a  user's  screen.   If  no resource identifier is
given with -id, xkill will
       display a special cursor as a prompt for the user to select a window
to be killed.   If  a
       pointer  button is pressed over a non-root window, the server will
close its connection to
       the client that created the window.
```

Lab Report No-07:Implementation of FCFS Scheduling Algorithm

i)What is FCFS Scheduling Algorithm?

First Come First Serve (FCFS) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest CPU scheduling algorithm. In this type of algorithm, processes which requests the CPU first get the CPU allocation first. This is managed with a FIFO queue. The full form of FCFS is First Come First Serve.

As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue and, when the CPU becomes free, it should be assigned to the process at the beginning of the queue.

ii)How to implementation in c?

```
#include<stdio.h>

int main()
{
   int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
   printf("Enter total number of processes(maximum 20):");
   scanf("%d",&n);

   printf("\nEnter Process Burst Time\n");
   for(i=0;i<n;i++)
   {
      printf("P[%d]:",i+1);
      scanf("%d",&bt[i]);
   }

   wt[0]=0;    //waiting time for first process is 0

   //calculating waiting time
   for(i=1;i<n;i++)
   {
      wt[i]=0;
      for(j=0;j<i;j++)
         wt[i]+=bt[j];
   }

   printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

   //calculating turnaround time
   for(i=0;i<n;i++)
   {
      tat[i]=bt[i]+wt[i];
      avwt+=wt[i];
      avtat+=tat[i];
      printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
   }

   avwt/=i;
   avtat/=i;
   printf("\n\nAverage Waiting Time:%d",avwt);
   printf("\nAverage Turnaround Time:%d",avtat);

   return 0;
}
```

Lab Report No-08:Implementations of SJF Scheduling Algorithm.

i)What is SJF Scheduling Algorithm?

Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm.

- Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.
- It is a Greedy Algorithm.
- It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.
- It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available.

ii)How to Implementation in c?

```
#include<iostream>
using namespace std;
int mat[10][6];

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void arrangeArrival(int num, int mat[][6])
{
    for(int i=0; i<num; i++)
    {
        for(int j=0; j<num-i-1; j++)
        {
            if(mat[j][1] > mat[j+1][1])
            {
                for(int k=0; k<5; k++)
                {
                    swap(mat[j][k], mat[j+1][k]);
                }
            }
        }
    }
}

void completionTime(int num, int mat[][6])
{
    int temp, val;
    mat[0][3] = mat[0][1] + mat[0][2];
    mat[0][5] = mat[0][3] - mat[0][1];
    mat[0][4] = mat[0][5] - mat[0][2];

    for(int i=1; i<num; i++)
    {
        temp = mat[i-1][3];
        int low = mat[i][2];
        for(int j=i; j<num; j++)
        {
```

```
            if(temp >= mat[j][1] && low >= mat[j][2])
            {
                low = mat[j][2];
                val = j;
            }
        }
        mat[val][3] = temp + mat[val][2];
        mat[val][5] = mat[val][3] - mat[val][1];
        mat[val][4] = mat[val][5] - mat[val][2];
        for(int k=0; k<6; k++)
        {
            swap(mat[val][k], mat[i][k]);
        }
    }
}

int main()
{
    int num, temp;

    cout<<"Enter number of Process: ";
    cin>>num;

    cout<<"...Enter the process ID...\n";
    for(int i=0; i<num; i++)
    {
        cout<<"...Process "<<i+1<<"...\n";
        cout<<"Enter Process Id: ";
        cin>>mat[i][0];
        cout<<"Enter Arrival Time: ";
        cin>>mat[i][1];
        cout<<"Enter Burst Time: ";
        cin>>mat[i][2];
    }

    cout<<"Before Arrange...\n";
    cout<<"Process ID\tArrival Time\tBurst Time\n";
    for(int i=0; i<num; i++)
    {
        cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\n";
    }

    arrangeArrival(num, mat);
    completionTime(num, mat);
    cout<<"Final Result...\n";
    cout<<"Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround
Time\n";
    for(int i=0; i<num; i++)
    {
        cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\t\t"<<mat[
i][4]<<"\t\t"<<mat[i][5]<<"\n";
    }
}
```

Lab Report No-09:Implementation of Priority Scheduling Algorithm

i)What Is Priority Scheduling Algorithm?

Priority Scheduling is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority.

The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority depends upon memory requirements, time requirements, etc.

ii)HOw to implement in C?

```
#include<stdio.h>

int main()
{
   int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
   printf("Enter Total Number of Process:");
   scanf("%d",&n);

   printf("\nEnter Burst Time and Priority\n");
   for(i=0;i<n;i++)
   {
      printf("\nP[%d]\n",i+1);
      printf("Burst Time:");
      scanf("%d",&bt[i]);
      printf("Priority:");
      scanf("%d",&pr[i]);
      p[i]=i+1;
   }


   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(pr[j]<pr[pos])
            pos=j;
      }

      temp=pr[i];
      pr[i]=pr[pos];
      pr[pos]=temp;

      temp=bt[i];
      bt[i]=bt[pos];
      bt[pos]=temp;

      temp=p[i];
      p[i]=p[pos];
      p[pos]=temp;
   }

   wt[0]=0;
```

```
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }

    avg_wt=total/n;
    total=0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }

    avg_tat=total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\nAverage Turnaround Time=%d\n",avg_tat);

            return 0;
}
```

Lab Report No-10:Implementation of Round Robin Scheduling Algorithm.

i)What is Round Robin Scheduling Algorithm?

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.

- It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
- One of the most commonly used technique in CPU scheduling as a core.
- It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
- The disadvantage of it is more overhead of context switching.

ii)How to implement in C?

```
#include<stdio.h>

int main()
```

```
{

  int count,j,n,time,remain,flag=0,time_quantum;
  int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
  printf("Enter Total Process:\t ");
  scanf("%d",&n);
  remain=n;
  for(count=0;count<n;count++)
  {
    printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
    scanf("%d",&at[count]);
    scanf("%d",&bt[count]);
    rt[count]=bt[count];
  }
  printf("Enter Time Quantum:\t");
  scanf("%d",&time_quantum);
  printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
  for(time=0,count=0;remain!=0;)
  {
    if(rt[count]<=time_quantum && rt[count]>0)
    {
      time+=rt[count];
      rt[count]=0;
      flag=1;
    }
    else if(rt[count]>0)
    {
      rt[count]-=time_quantum;
      time+=time_quantum;
    }
    if(rt[count]==0 && flag==1)
    {
      remain--;
      printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
      wait_time+=time-at[count]-bt[count];
      turnaround_time+=time-at[count];
      flag=0;
    }
    if(count==n-1)
      count=0;
    else if(at[count+1]<=time)
      count++;
    else
      count=0;
  }
  printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
  printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);

  return 0;
}
```

Lab Report NO-11:Implementation of FIFO page replacement Scheduling Algorithm.

i)What is FIFO page replacement Scheduling?

First In First Out (FIFO) –
This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for .

ii)How to Implement in C?

```
#include<stdio.h>
int main()
{
        int i,j,n,a[50],frame[10],no,k,avail,count=0;
        printf("\n ENTER THE NUMBER OF PAGES:\n");
        scanf("%d",&n);
        printf("\n ENTER THE PAGE NUMBER :\n");
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        printf("\n ENTER THE NUMBER OF FRAMES :");
        scanf("%d",&no);
        for(i=0;i<no;i++)
        frame[i]= -1;
                j=0;
                printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
                {
                        printf("%d\t\t",a[i]);
                        avail=0;
                        for(k=0;k<no;k++)
if(frame[k]==a[i])
                                avail=1;
                        if (avail==0)
                        {
                                frame[j]=a[i];
                                j=(j+1)%no;
                                count++;
                                for(k=0;k<no;k++)
                                printf("%d\t",frame[k]);
        }
                        printf("\n");
        }
                printf("Page Fault Is %d",count);
                return 0;
}
```