

Subject Name: Operating System

Subject Code: 22CS005

Session: 2022-23

Department: DCSE



**PRACTICAL FILE
OF
OPERATING SYSTEM
SUBMITTED TO
DEPARTMENT OF COMPUTER SCIENCE
UNDER THE SUPERVISION OF
Dr. Sunil Gupta**

**Submitted By:
Simran Kaur
2210990852**

**CHITKARA UNIVERSITY
CHANDIGARH-PATIALA NATIONAL HIGHWAY, RAJPURA,
DISTT. PATIALA, PUNJAB, INDIA**

Index

S. No.	Experiments	Page Number
1	Declaration	1
2	Introduction	2
3	Installation: Configuration & Customizations of Linux	3-8
4	Introduction to GCC compiler: Basics of GCC, Compilation of program, Execution of program, Time stamping, Automating the execution using Make.	9
5	Implement Process concepts using C language by Printing process Id, Execute Linux command as sub process, Creating and executing process using fork and exec system calls.	10-11
6	Implement FCFS, SJF, priority scheduling, and RR scheduling algorithms in C language.	12-17
7	Implement the basic and user status commands like: su, sudo, man, help, history, who, whoami, id, uname, uptime, free, tty, cal, date, hostname, reboot, clear	18-20

DECLARATION

I hereby declare that the lab practical file submitted as part of Term 3 OPERATING SYSTEM subject (sem2) bachelor's degree in CSE, at Chitkara University, Punjab, is an authentic record of my own work carried out under the supervision of Dr. Sunil Gupta.

Signature(s):

Name: Simran Kaur

Roll No: 2210990852

OPERATING SYSTEM INTRODUCTION

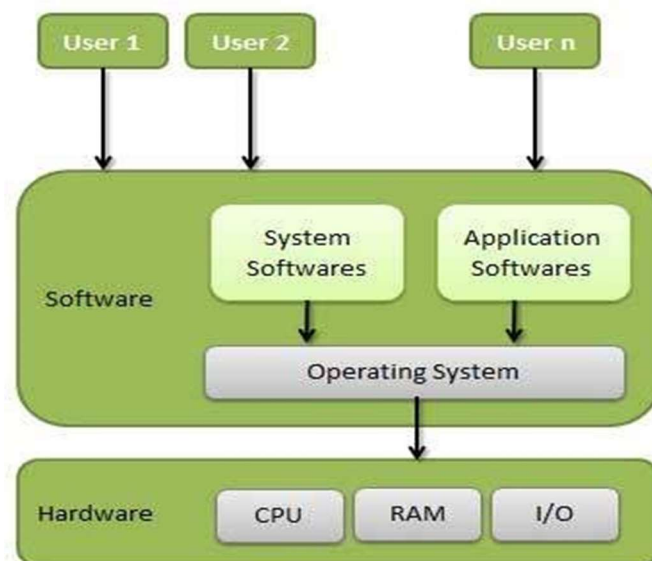
An **Operating System** (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

An operating system is software that enables applications to interact with a computer's hardware. The software that contains the core components of the operating system is called the **kernel**.

The primary purposes of an **Operating System** are to enable applications (spftwares) to interact with a computer's hardware and to manage a system's hardware and software resources.

Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Operating system architecture:



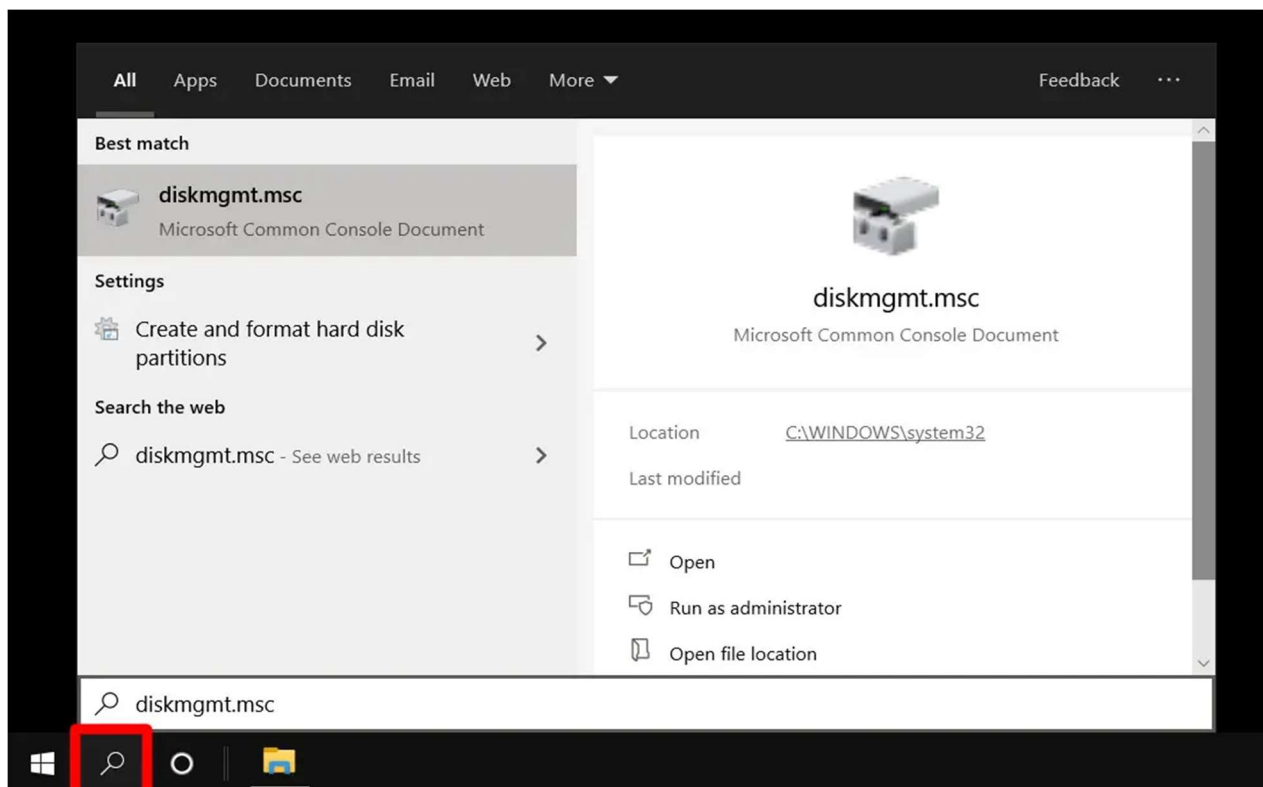
1.) Installation: Configuration & Customizations of Linux.

Linux is a family of open-source operating systems. They are based on the Linux kernel and are free to download. They can be installed on either a Mac or Windows computer. Here's how to install Linux on a Windows 10 PC.

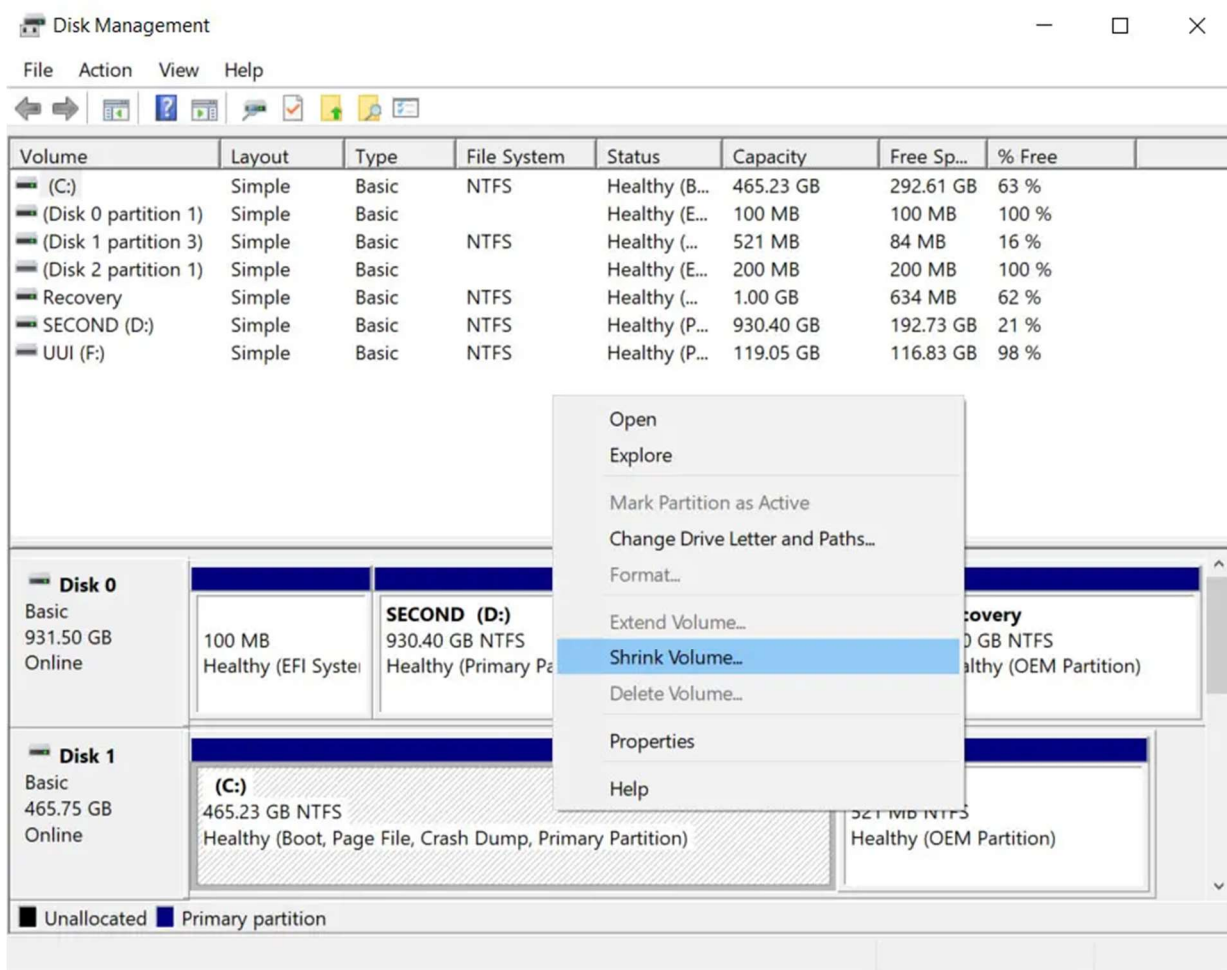
If you want to dual boot Linux and Windows, you will need to create a space for your Linux OS to live. To do this, you will have to partition your main hard drive. Here's how to do that:

How to Partition a Hard Drive in Windows 10

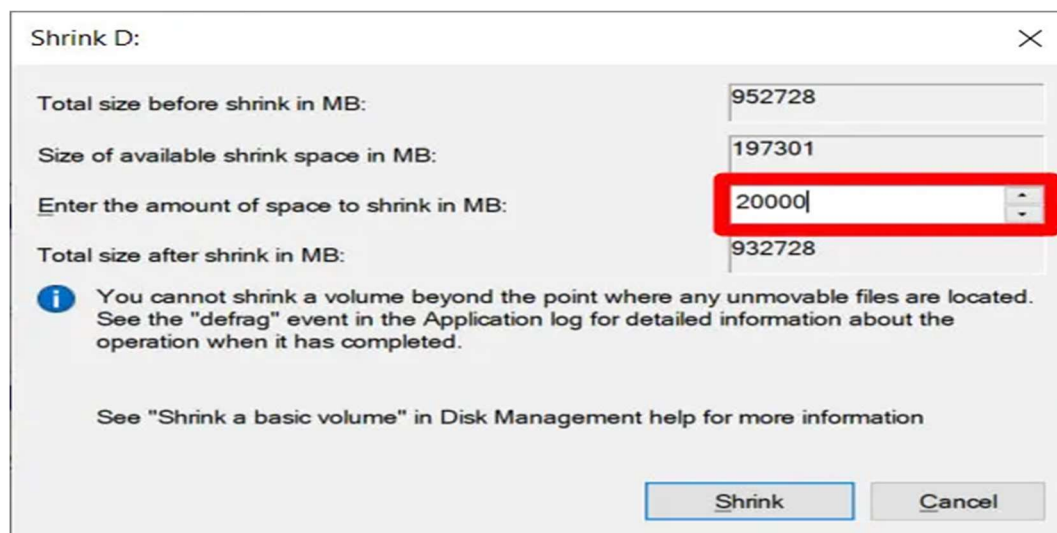
1. **Open the Windows Search Bar.** This is the magnifying glass-shaped icon in the bottom-left corner of your screen.
2. **Then type "DISKMGMT.MSC" in the search bar and hit enter.**



3. **Right-click on your main hard drive and select Shrink Volume.** If you have more than one drive, make sure to choose the one that says Primary Partition. This will usually be labeled as the C: drive.



4. **Then choose how much you want to shrink your drive.** It is recommended that you set aside at least 20GB (20,000MB) for Linux.

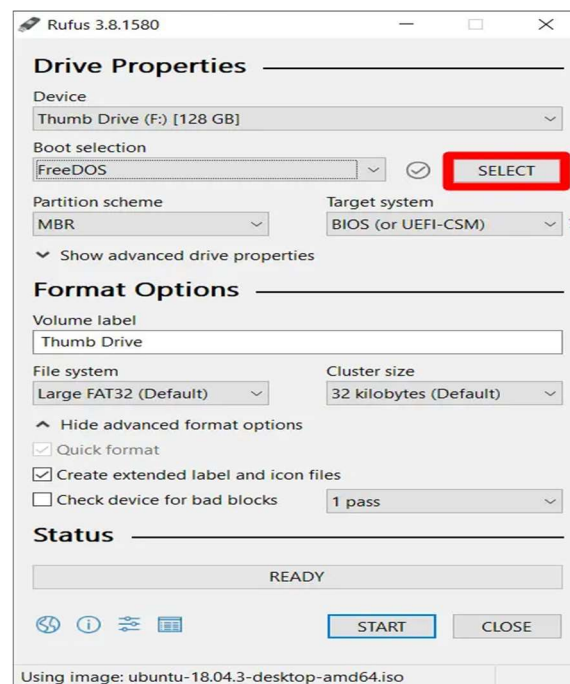


5. **Finally, click Shrink.**

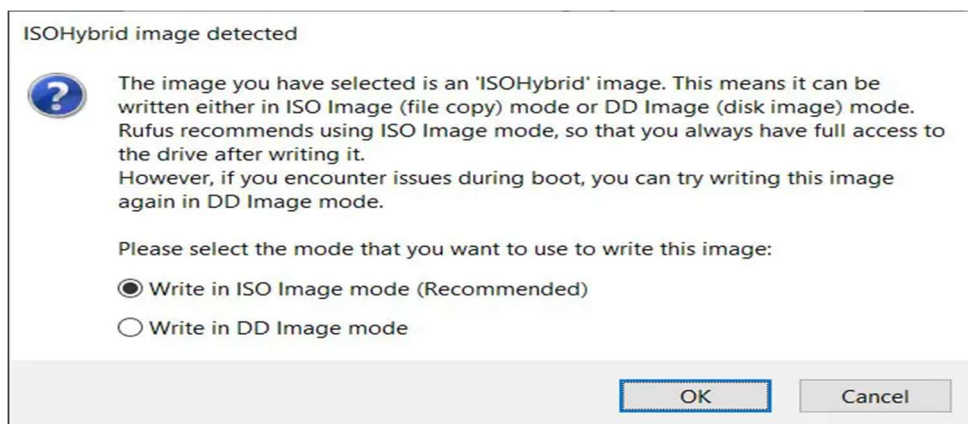
Once you have a designated space to install Linux, you'll need to write a Linux Distro onto a USB thumb drive or external drive 4GB or larger. Here's how to do that:

How to Make a Linux Bootable USB

1. **Download a Linux distro in ISO format.** An ISO file is a disk image. Some of the top options are Ubuntu, Mint, or Fedora. They are free to download from each distribution's main website. For this article, we are using Ubuntu.
2. **Insert the USB drive into your computer.** You might be asked to format your drive. This will erase all the data stored on your drive, so make sure to back up your files before you begin.
3. **Download Rufus.** You can find the latest version of the application here.
4. **Open Rufus and select your USB drive from the Device list.** If you don't know which drive to use, eject all other drives until you only have one to choose from.
5. **Under Boot Selection, click the Select button and choose the ISO file you downloaded earlier.** Don't change the other default settings.



6. **Finally, click Start.** If you get a pop-up message asking you to select a mode that you want to use to write the image, choose ISO.



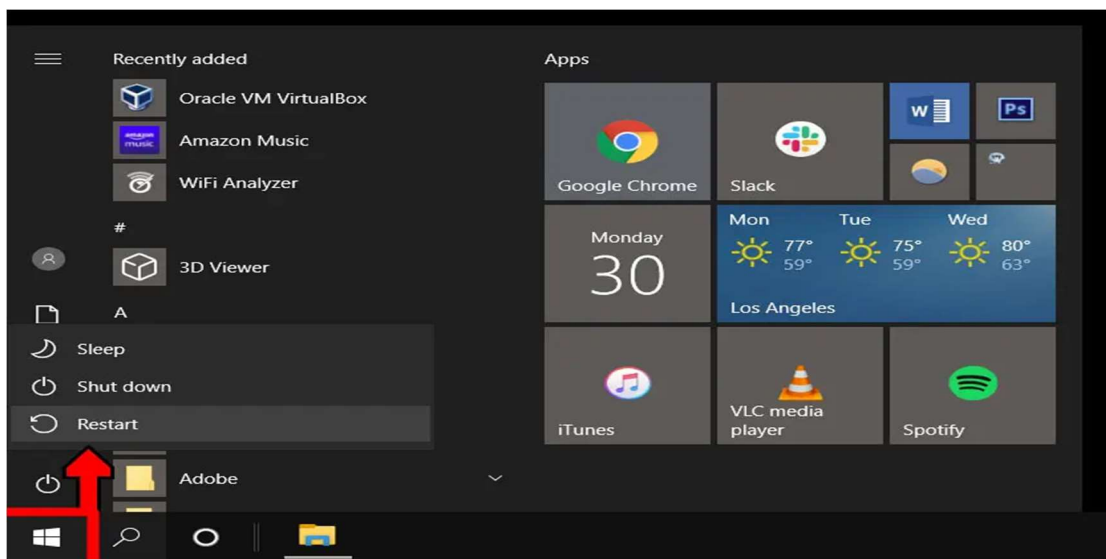
Then wait for Rufus to mount your ISO file onto your drive. This might take some time, so be patient if the progress bar gets stuck.

Warning: This will erase all the data on your drive, so make sure to back up any important files.

How to Install Linux from USB

Now that you have your Linux distro on a USB, here's how to

1. **Insert a bootable Linux USB drive.**
2. **Click the start menu.** This is the button in the lower-left corner of your screen that looks like the Windows logo.
3. **Then hold down the SHIFT key while clicking Restart.** This will take you into the Windows Recovery Environment.



4. **Then select Use a Device.**

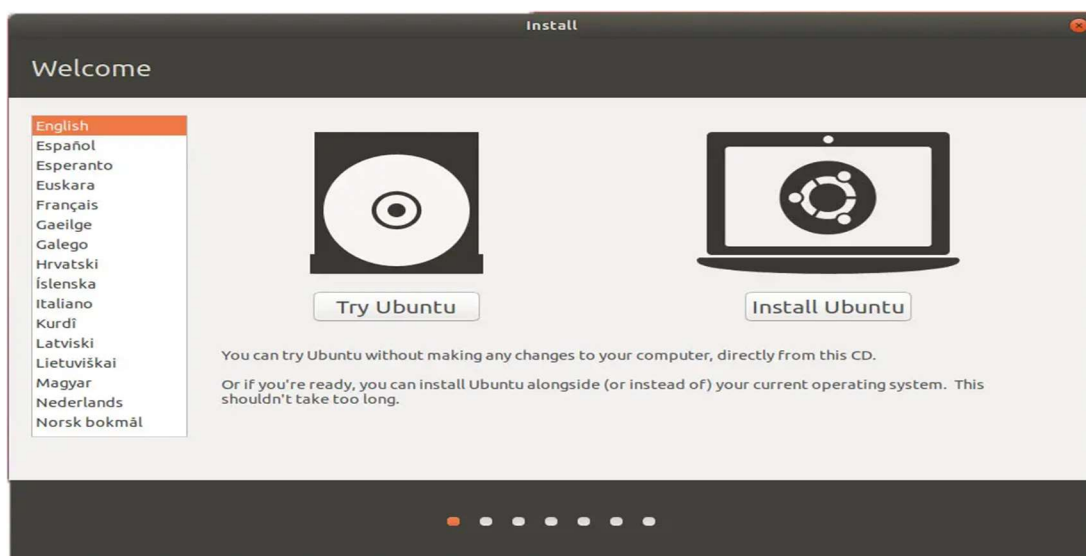


5. **Find your device in the list.** If you don't see your drive, choose EFI USB Device, then pick your drive from the next screen.



6. Your computer will now boot Linux. If your computer reboots Windows, there was either an issue with your drive, or you might have to change settings in your BIOS. Warning: Changing BIOS settings can damage your computer if you don't know what you're doing.

7. Select Install Linux. Some distros also let you try out the OS before installing it here.



8. Go **through the installation process**. This will differ depending on which distro you are trying to install. These details might include your WiFi network, language, time zone, keyboard layout, etc. You might also be required to create an account with a username and password. Make sure to write down any details, as you will likely need them in the future.

9. Most distros will allow you to partition your drive or erase it and do a clean install during the installation. Warning: Erasing your disk will mean you will lose your settings, files, and Windows operating system. Only select Erase if you have saved copies of all your files before starting the install process.

10. Reboot your computer when prompted. If you have more than one OS in your system, you will be taken to a GNU GRUB screen after rebooting. This screen allows you to select which OS you want to boot.

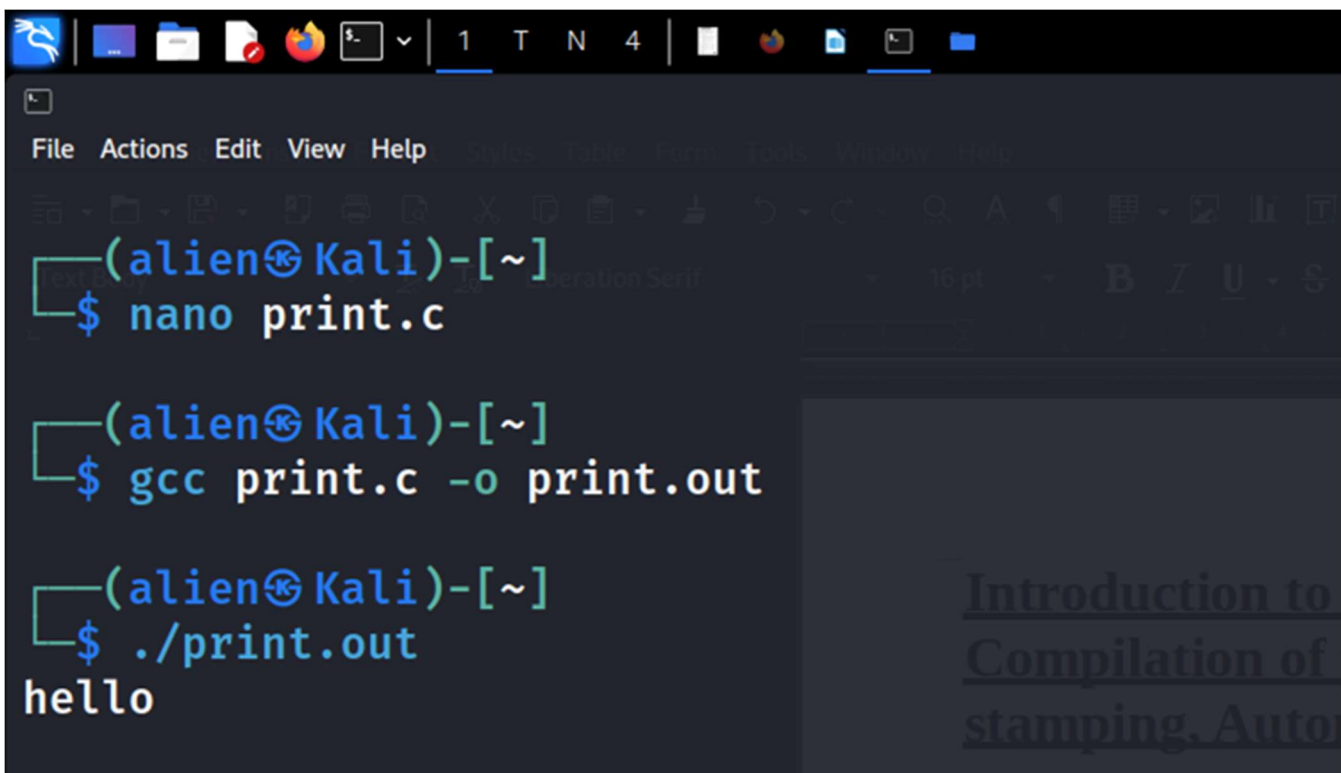


If you do not see a GRUB screen when you boot up your computer, you can try moving your Linux distro higher on your boot list in BIOS.

When you're done, you can do a hardware check. In some cases, you may need to download additional drivers to make some hardware work. The option to download drivers can be found in the systems Settings of your new Linux OS. After verifying that your hardware is working properly, you can start exploring and using your Linux distro.

2.) Introduction to GCC compiler: Basics of GCC, Compilation of program, Execution of program, Time stamping, Automating the execution using Make file.

- GCC (GNU Compiler Collection) is a set of compilers for various programming languages, including C, C++, Objective-C, Fortran, Ada, and others.
- To compile a C program using GCC, use the command "gcc -o <output_file> <input_file.c>".
- To execute the compiled program, use the command ". ./<output_file>".
- Timestamping refers to the process of recording the date and time when a program was compiled.
- Makefile is a script used to automate the process of compiling and executing programs. It contains instructions on how to compile the program, what files to compile, and how to link them. To execute a Makefile, use the command "make".



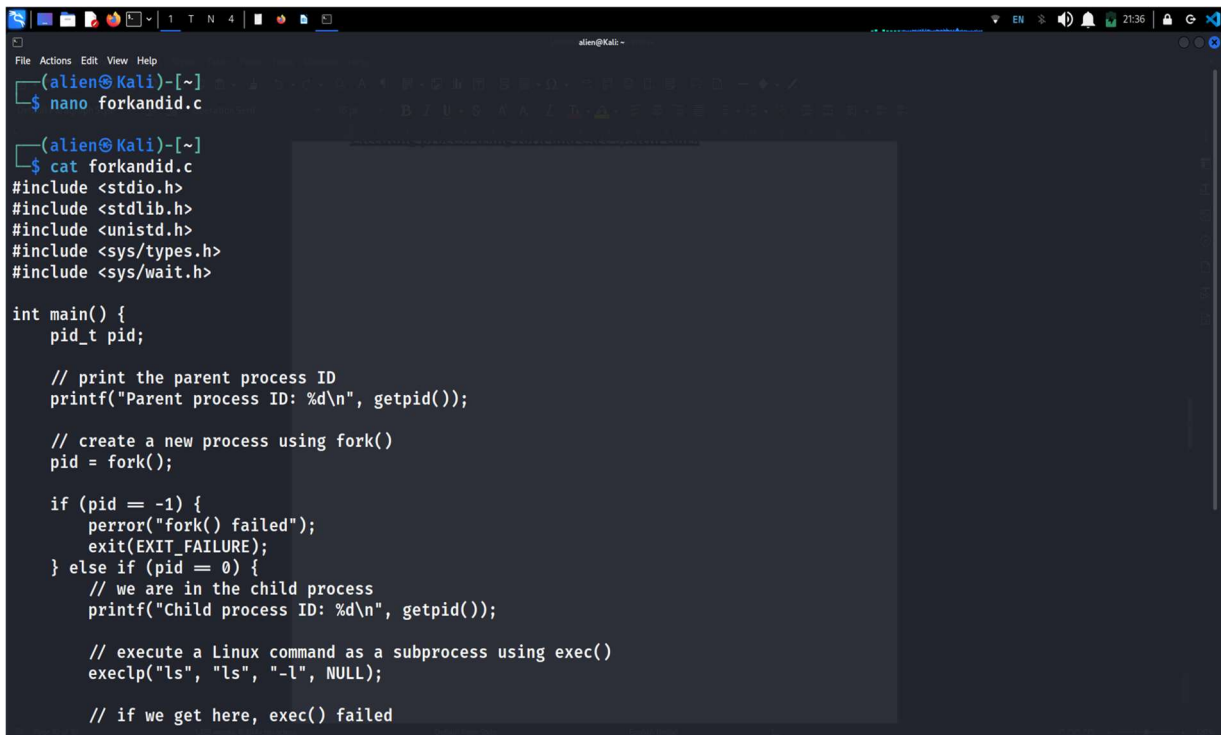
```
(alien@Kali)-[~]
$ nano print.c

(alien@Kali)-[~]
$ gcc print.c -o print.out

(alien@Kali)-[~]
$ ./print.out
hello
```

The screenshot shows a terminal window with a dark background. The prompt is (alien@Kali)-[~]. The user enters 'nano print.c' to create a file. Then, they enter 'gcc print.c -o print.out' to compile the program. Finally, they enter './print.out' to execute it, which outputs 'hello'. The terminal window has a menu bar with 'File', 'Actions', 'Edit', 'View', and 'Help'. There are also icons for various applications in the top bar.

3.) Implement Process concepts using C language by Printing process Id, Execute Linux command as sub process, Creating and executing process using fork and exec system calls.



```
File Actions Edit View Help
(alien@Kali)-[~]
$ nano forkandid.c

(alien@Kali)-[~]
$ cat forkandid.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
    pid_t pid;

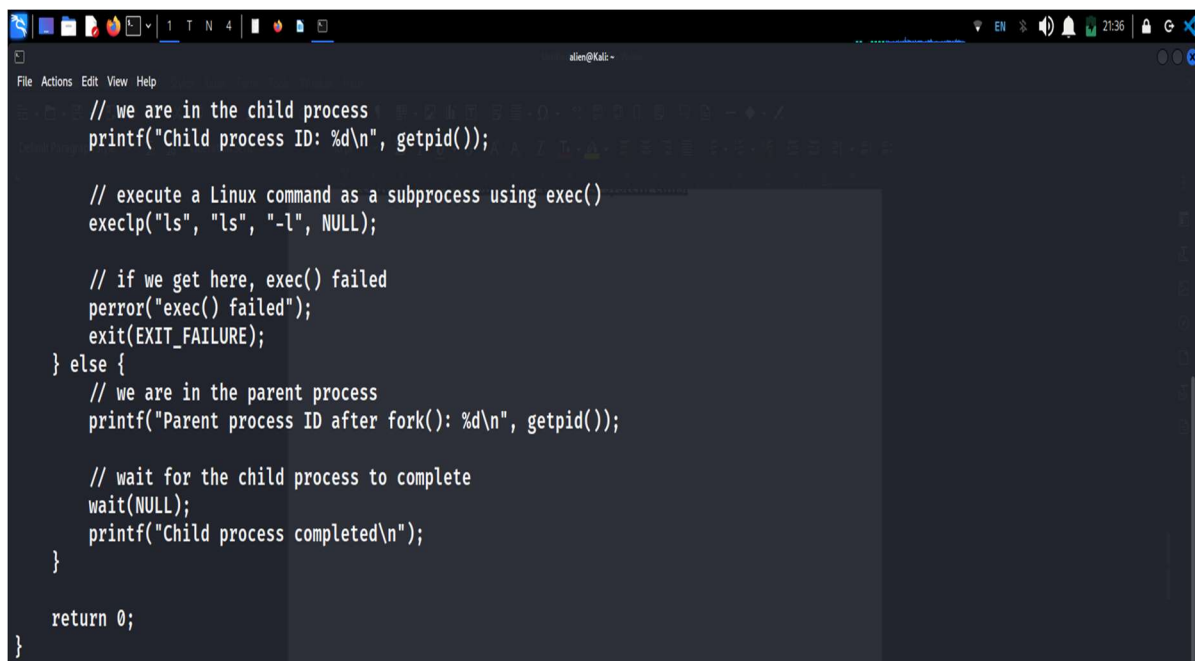
    // print the parent process ID
    printf("Parent process ID: %d\n", getpid());

    // create a new process using fork()
    pid = fork();

    if (pid == -1) {
        perror("fork() failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // we are in the child process
        printf("Child process ID: %d\n", getpid());

        // execute a Linux command as a subprocess using exec()
        execlp("ls", "ls", "-l", NULL);

        // if we get here, exec() failed
    }
```



```
File Actions Edit View Help
// we are in the child process
printf("Child process ID: %d\n", getpid());

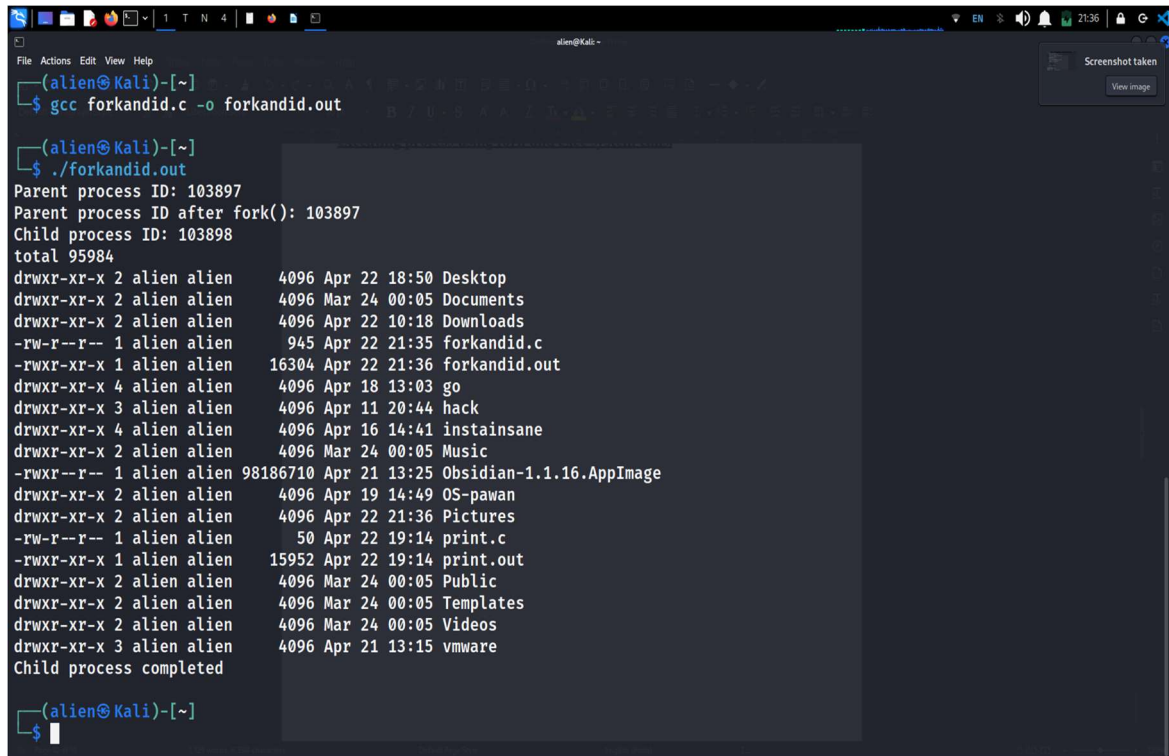
// execute a Linux command as a subprocess using exec()
execlp("ls", "ls", "-l", NULL);

// if we get here, exec() failed
perror("exec() failed");
exit(EXIT_FAILURE);
} else {
    // we are in the parent process
    printf("Parent process ID after fork(): %d\n", getpid());

    // wait for the child process to complete
    wait(NULL);
    printf("Child process completed\n");
}

return 0;
}
```

EXECUTION:



```
(alien@Kali)-[~]
$ gcc forkandid.c -o forkandid.out

(alien@Kali)-[~]
$ ./forkandid.out
Parent process ID: 103897
Parent process ID after fork(): 103897
Child process ID: 103898
total 95984
drwxr-xr-x 2 alien alien 4096 Apr 22 18:50 Desktop
drwxr-xr-x 2 alien alien 4096 Mar 24 00:05 Documents
drwxr-xr-x 2 alien alien 4096 Apr 22 10:18 Downloads
-rw-r--r-- 1 alien alien 945 Apr 22 21:35 forkandid.c
-rwxr-xr-x 1 alien alien 16304 Apr 22 21:36 forkandid.out
drwxr-xr-x 4 alien alien 4096 Apr 18 13:03 go
drwxr-xr-x 3 alien alien 4096 Apr 11 20:44 hack
drwxr-xr-x 4 alien alien 4096 Apr 16 14:41 instainsane
drwxr-xr-x 2 alien alien 4096 Mar 24 00:05 Music
-rwxr--r-- 1 alien alien 98186710 Apr 21 13:25 Obsidian-1.1.16.AppImage
drwxr-xr-x 2 alien alien 4096 Apr 19 14:49 OS-pawan
drwxr-xr-x 2 alien alien 4096 Apr 22 21:36 Pictures
-rw-r--r-- 1 alien alien 50 Apr 22 19:14 print.c
-rwxr-xr-x 1 alien alien 15952 Apr 22 19:14 print.out
drwxr-xr-x 2 alien alien 4096 Mar 24 00:05 Public
drwxr-xr-x 2 alien alien 4096 Mar 24 00:05 Templates
drwxr-xr-x 2 alien alien 4096 Mar 24 00:05 Videos
drwxr-xr-x 3 alien alien 4096 Apr 21 13:15 vmware
Child process completed

(alien@Kali)-[~]
$
```

4.) Implement FCFS, SJF, priority scheduling, and RR scheduling algorithms in C language.

FCFS- First Come First Serve CPU Scheduling Algorithm shortly known as FCFS is the first algorithm of CPU Process Scheduling Algorithm. In First Come First Serve Algorithm what we do is to allow the process to execute in linear manner. This means that whichever process enters process enters the ready queue first is executed first. This shows that First Come First Serve Algorithm follows First in First Out (FIFO) principle. The First Come First Serve Algorithm can be executed in Pre Emptive- and Non-Pre Emptive manner. Before, going into examples, let us understand what Pre Emptive is and Non-Pre Emptive-Approach in CPU Process Scheduling.

Characteristics of FCFS CPU Process Scheduling

The characteristics of FCFS CPU Process Scheduling are:

1. Implementation is simple.
2. Does not cause any causalities while using.
3. It adopts a non pre emptive and pre emptive strategy.
4. It runs each procedure in the order that they are received.
5. Arrival time is used as a selection criterion for procedures.

Advantages of FCFS CPU Process Scheduling

The advantages of FCFS CPU Process Scheduling are:

1. To allocate processes, it uses the First in First Out queue.
2. The FCFS CPU Scheduling Process is straight forward and easy to implement.
3. In the FCFS situation pre emptive scheduling, there is no chance of process starving.
4. As there is no consideration of process priority, it is an equitable algorithm.

Disadvantages of FCFS CPU Process Scheduling

The disadvantages of FCFS CPU Process Scheduling are:

- FCFS CPU Scheduling Algorithm has Long Waiting Time
- FCFS CPU Scheduling favors CPU over Input or Output operations
- In FCFS there is a chance of occurrence of Convoy Effect
- Because FCFS is so straight forward, it often isn't very effective. Extended waiting periods go hand in hand with this. All other orders are left idle if the CPU is busy processing one time-consuming order.

CODE EXECUTION:

```
alien@Kali: ~$ nano FCFS.c
alien@Kali: ~$ gcc FCFS.c -o FCFS.out
alien@Kali: ~$ cat FCFS.c
#include <stdio.h>
int main()
{
    int pid[15];
    int bt[15];
    int n;
    printf("Enter the number of processes: ");
    scanf("%d",&n);
    printf("Enter process id of all the processes: ");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&pid[i]);
    }
    printf("Enter burst time of all the processes: ");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&bt[i]);
    }
    int i, wt[n];
    wt[0]=0;
    for(i=1; i<n; i++)
    {
        wt[i]= bt[i-1]+ wt[i-1];
    }
}
```

```
alien@Kali: ~$ cat FCFS.c
#include <stdio.h>
int main()
{
    int i, wt[n];
    wt[0]=0;
    for(i=1; i<n; i++)
    {
        wt[i]= bt[i-1]+ wt[i-1];
    }
    printf("Process ID\tBurst Time\tWaiting Time\tTurnAround Time\n");
    float twt=0.0;
    float tat= 0.0;
    for(i=0; i<n; i++)
    {
        printf("%d\t\t", pid[i]);
        printf("%d\t\t", bt[i]);
        printf("%d\t\t", wt[i]);
        printf("%d\t\t", bt[i]+wt[i]);
        printf("\n");
        twt += wt[i];
        tat += (wt[i]+bt[i]);
    }
    float att,awt;
    awt = twt/n;
    att = tat/n;
    printf("Avg. waiting time= %f\n",awt);
    printf("Avg. turnaround time= %f",att);
}
```

```
alien@Kali: ~$ ./FCFS.out
Enter the number of processes: 5
Enter process id of all the processes: 1
2
3
4
5
Enter burst time of all the processes: 5 10 4 3 8
Process ID\tBurst Time\tWaiting Time\tTurnAround Time
1\t5\t0\t5
2\t10\t5\t15
3\t4\t15\t19
4\t3\t19\t22
5\t8\t22\t30
Avg. waiting time= 12.200000
Avg. turnaround time= 18.200001
```

SJE SCHEDULING: Shortest Job First (SJF) is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution. The full form of SJF is Shortest Job First.

There are basically two types of SJF methods:

- Non-Preemptive SJF
- Preemptive SJF

Characteristics of SJF Scheduling

- It is associated with each job as a unit of time to complete.
- This algorithm method is helpful for batch-type processing, where waiting for jobs to complete is not critical.
- It can improve process throughput by making sure that shorter jobs are executed first, hence possibly have a short turnaround time.
- It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

Non-Preemptive SJF

In non-preemptive scheduling, once the CPU cycle is allocated to process, the process holds it till it reaches a waiting state or terminated.

Preemptive SJF

In Preemptive SJF Scheduling, jobs are put into the ready queue as they come. A process with shortest burst time begins execution. If a process with even a shorter burst time arrives, the current process is removed or preempted from execution, and the shorter job is allocated CPU cycle.

Advantages of SJF

- SJF is frequently used for long term scheduling.
- It reduces the average waiting time over FIFO (First in First Out) algorithm.
- SJF method gives the lowest average waiting time for a specific set of processes.
- It is appropriate for the jobs running in batch, where run times are known in advance.

Disadvantages/Cons of SJF

- Job completion time must be known earlier, but it is hard to predict.
- It is often used in a batch system for long term scheduling.
- It is hard to know the length of the upcoming CPU request.

CODE EXECUTION:

```
File Edit Search View Document Help
*~J5fc - Mousepad

4 int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,totalT=0,pos,temp;
5 float avg_wt,avg_tat;
6 printf("Enter number of process:");
7 scanf("%d",&n);
8 printf("Enter Burst Time:\n");
9 for(i=0;i<n;i++)
10 {
11     printf("p%d:",i+1);
12     scanf("%d",&bt[i]);
13     p[i]=i+1;
14 }
15 //sorting of burst times
16 for(i=0;i<n;i++)
17 {
18     pos=i;
19     for(j=i+1;j<n;j++)
20     {
21         if(bt[j]<bt[pos])
22             pos=j;
23     }
24     temp=bt[i];
25     bt[i]=bt[pos];
26     bt[pos]=temp;
27     temp=p[i];
28     p[i]=p[pos];
29     p[pos]=temp;
30 }
31 wt[i]=0;
32 for(i=0;i<n;i++)
33 {
34     wt[i]=0;
35     for(j=i+1;j<n;j++)
36         wt[i]=bt[j];
37     //total waiting time
38     total+=wt[i];
39 }
40 //average waiting time
41 avg_wt=(float)total/n;
42 printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
43 for(i=0;i<n;i++)
44 {
45     //turnaround time of individual processes
46     tat[i]=bt[i]+wt[i];
47     //total turnaround time
48     totalT+=tat[i];
49     printf("\np%d\t\t %d\t\t %d\t\t %d\t\t %d",p[i],bt[i],wt[i],tat[i]);
50 }
51 //average turnaround time
52 avg_tat=(float)totalT/n;
53 printf("\n\nAverage Waiting Time=%f",avg_wt);
54 printf("\n\nAverage Turnaround Time=%f",avg_tat);
55 }
```

```
File Actions Edit View Help
[1] 119013
(alien@Kali)-[~]
$ mousepad Sjfc.c
(alien@Kali)-[~]
$ gcc Sjfc.c -o Sjfc.out
(alien@Kali)-[~]
$ ./Sjfc.out
Enter number of process:5
Enter Burst Time:
p1:5
p2:4
p3:9
p4:3
p5:7

Process Burst Time Waiting Time Turnaround Time
p4 3 0 3
p2 4 3 7
p1 5 7 12
p5 7 12 19
p3 9 19 28

Average Waiting Time=8.200000
Average Turnaround Time=13.800000
(alien@Kali)-[~]
$
```

SJF Scheduling Program in C

Problem Description:
Write a C++ scheduling program in C++ to determine the average waiting time and average turnaround time given a provided set of 5000 burst times.

SJF Scheduling Algorithm in C++
The C++ scheduling algorithm discussed for First SJF, involves the CPU to give a process according to the process with smallest execution time.

Advantages of SJF:

- It uses the minimum waiting time among all the scheduling algorithms.
- A process having longer burst time may get into starvation but the problem can be solved using concept of Aging.
- It is a greedy algorithm and provides optimal scheduling.

Process	Burst Time	Waiting Time	Turnaround Time
p4	3	0	3
p2	4	3	7
p1	5	7	12
p5	7	12	19
p3	9	19	28

Average Waiting Time=8.200000
Average Turnaround Time=13.800000

Sanfoundry Certification

Round Robin CPU Scheduling

Round Robin CPU Scheduling is the most important CPU Scheduling Algorithm which is ever used in the history of CPU Scheduling Algorithms. Round Robin CPU Scheduling uses Time Quantum (TQ). The Time Quantum is something which is removed from the Burst Time and lets the chunk of process to be completed. Time Sharing is the main emphasis of the algorithm. Each step of this algorithm is carried out cyclically. The system defines a specific time slice, known as a time quantum. First, the processes which are eligible to enter the ready queue enter the ready queue. After entering the first process in Ready Queue is executed for a Time Quantum chunk of time. After execution is complete, the process is removed from the ready queue. Even now the process requires some time to complete its execution, then the process is added to Ready Queue. The Ready Queue does not hold processes which already present in the Ready Queue. The Ready Queue is designed in such a manner that it does not hold non unique processes. By holding same processes Redundancy of the processes increases.

Advantages

The Advantages of Round Robin CPU Scheduling are:

1. A fair amount of CPU is allocated to each job.
2. Because it doesn't depend on the burst time, it can truly be implemented in the system.
3. It is not affected by the convoy effect, or the starvation problem as occurred in First Come First Serve CPU Scheduling Algorithm.

Disadvantages

The Disadvantages of Round Robin CPU Scheduling are:

1. Low Operating System slicing times will result in decreased CPU output.
2. Round Robin CPU Scheduling approach takes longer to swap contexts.
3. Time quantum has a significant impact on its performance.
4. The procedures cannot have priorities established.

CODE EXECUTION:

```
File Actions Edit View Help
#include<stdio.h>
// #include<conio.h>
void main()
{
    // initialize the variable name
    int i, NOP, sum=0, count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
    float avg_wt, avg_tat;
    printf("Total number of process in the system: ");
    scanf("%d", &NOP);
    y = NOP; // Assign the number of process to variable y
    // Use for loop to enter the details of the process like Arrival time and the Burst Time
    for(i=0; i<NOP; i++)
    {
        printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
        printf("Arrival time is: \t"); // Accept arrival time
        scanf("%d", &at[i]);
        printf("\nBurst time is: \t"); // Accept the Burst time
        scanf("%d", &bt[i]);
        temp[i] = bt[i]; // store the burst time in temp array
    }
    // Accept the Time quantum
    printf("Enter the Time Quantum for the process: \t");
    scanf("%d", &quant);
    // Display the process No, burst time, Turn Around Time and the waiting time
    printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
    for(sum=0, i = 0; y>0; )
    {
        if(temp[i] <= quant && temp[i] > 0) // define the conditions
        {
            sum = sum + temp[i];
            temp[i] = 0;
            count=1;
        }
        else if(temp[i] > 0)
        {
            temp[i] = temp[i] - quant;
            sum = sum + quant;
        }
    }
}
```

```
}
if(temp[i]==0 && count==1)
{
    y--; //decrement the process no.
    printf("\nProcess No[%d] \t\t %d\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);
    wt = wt+sum-at[i]-bt[i];
    tat = tat+sum-at[i];
    count = 0;
}
if(i==NOP-1)
{
    i=0;
}
else if(at[i+1]<=sum)
{
    i++;
}
else
{
    i=0;
}
}
// represents the average waiting time and Turn Around time
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
// getch();
}
```

```
(alien@Kali)-[~]
$ gcc rr.c -o rr.out

(alien@Kali)-[~]
$ ./rr.out
Total number of process in the system: 4

Enter the Arrival and Burst time of the Process[1]
Arrival time is: 2
Burst time is: 3

Enter the Arrival and Burst time of the Process[2]
Arrival time is: 3
Burst time is: 5

Enter the Arrival and Burst time of the Process[3]
Arrival time is: 4
Burst time is: 4

Enter the Arrival and Burst time of the Process[4]
Arrival time is: 3
Burst time is: 2
Enter the Time Quantum for the process: 3

Process No      Burst Time      TAT      Waiting Time
Process No[1]   3               1         -2
Process No[4]   2               8         6
Process No[2]   5               10        5
Process No[3]   4               10        6
Average Turn Around Time: 3.750000
Average Waiting Time: 7.250000
```

5.) Implement the basic and user status commands like: su, sudo, man, help, history, who, whoami, id, uname, uptime, free, tty, cal, date, hostname, reboot, clear

1. **su:** The su command stands for "switch user" and allows you to temporarily become another user, typically the root user. This command requires superuser privileges and is commonly used to perform administrative tasks that require elevated permissions.
2. **sudo:** The sudo command is like su, but it allows you to execute a single command with elevated permissions, rather than switching to a different user account. This command also requires superuser privileges and is commonly used to perform administrative tasks.
3. **man:** The man command stands for "manual" and is used to display the manual pages for a particular command or topic. For example, man ls would display the manual page for the ls command.
4. **help:** The help command is used to display help information for the shell or a particular command. This command is often built into the shell and provides a brief overview of the available commands and their usage.
5. **history:** The history command is used to display a list of the commands that have been executed in the current shell session. This can be useful for recalling previously executed commands or for debugging.
6. **who:** The who command is used to display a list of the users who are currently logged in to the system.
7. **whoami:** The whoami command is used to display the username of the current user.
8. **id:** The id command is used to display information about the current user's identity, including their username, user ID, and group ID.
9. **uname:** The uname command is used to display information about the current system, including the kernel version, machine architecture, and operating system name.

10. **uptime:** The `uptime` command is used to display the current system uptime, including the current time, the length of time the system has been running, and the number of users currently logged in.
11. **free:** The `free` command is used to display information about the system's memory usage, including the amount of free, used, and available memory.
12. **tty:** The `tty` command is used to display the name of the terminal device that the current shell is running on.
13. **cal:** The `cal` command is used to display a calendar for the current month or a specified month and year.
14. **date:** The `date` command is used to display the current date and time.
15. **hostname:** The `hostname` command is used to display the hostname of the current system.
16. **reboot:** The `reboot` command is used to reboot the system.
17. **clear:** The `clear` command is used to clear the terminal screen.

CODE EXECUTION:

```
(alien@Kali)-[~]
$ su
Password:
su: Authentication failure

(alien@Kali)-[~]
$ sudo
usage: sudo -h | -K | -k | -V
usage: sudo -v [-ABkNnS] [-g group] [-h host] [-p prompt] [-u user]
usage: sudo -l [-ABkNnS] [-g group] [-h host] [-p prompt] [-U user] [-u user] [command [arg ...]]
usage: sudo [-ABbEHknPS] [-r role] [-t type] [-C num] [-D directory] [-g group] [-h host] [-p prompt] [-R directory] [-T timeout] [-u user] [VAR=value] [-i|-s] [command [arg ...]]
usage: sudo -e [-ABkNnS] [-r role] [-t type] [-C num] [-D directory] [-g group] [-h host] [-p prompt] [-R directory] [-T timeout] [-u user] file ...

(alien@Kali)-[~]
$ man
What manual page do you want?
For example, try 'man man'.

(alien@Kali)-[~]
$ history
 6 cs Download/
 7 cd Downloads?
 9 sudo dpkg -i code_1.76.2-1678817801_amd64.deb
11 sudo apt-get -y install blueman
15 cd "/media/alien/AREA 51/study material/C/Basic C programs/" && gcc sum_of_2num.c -o sum_of_2num && "/media/alien/AREA 51/study material/C/Basic_C_pro
```

```
File Actions Edit View Help
2039 sudo
2040 su
2041 sudo
2042 man

(alien@Kali)-[~]
(alien@Kali)-[~]
$ who
alien    tty7      2023-04-22 18:26 (:0)

(alien@Kali)-[~]
$ whoami
alien

(alien@Kali)-[~]
$ id
uid=1000(alien) gid=1000(alien) groups=1000(alien),4(adm),20(dialout),21(fax),24(cdrom),25(floppy),26(tape),27(sudo),29(audio),30(dip),44(video),46(plugdev),100(users),106(netdev),119(wireshark),120(blueetooth),125(lpadm),132(scanner),145(kaboxer),994(sambashare)

(alien@Kali)-[~]
$ uname
Linux

(alien@Kali)-[~]
$ uptime
22:40:04 up 4:14, 1 user, load average: 1.81, 1.73, 1.41

(alien@Kali)-[~]
$ free
              total            used             free       shared  buff/cache   available
Mem:        7958372        5511384        953488      643016     2444640     2446988
Swap:      1000444         136448         863996

(alien@Kali)-[~]
$ tty
/dev/pts/0
```