**Week l**

 Operating System installation – Closed Source & Disk Fragmentation

Operating System installation – Open Source

**Week 2**

Implementation of Filter Commands

**Week 3**

Shell Scripting using Control Structures and Functions

**Week 4**

Implement producer consumer problem with counting Semaphores and Mutexs.

**Week 5&6**

Implement the following CPU Scheduling algorithms

a) FCFS b) SJF c) Priority d) Round Robin

**Week 7**

Simulate Bankers algorithm for deadlock avoidance

**Week 8&9**

Simulate the Following Page Replacement algorithms

a) FIFO b) LRU c) LFU d) OPTIMAL

**Week 10 & 11**

Implement Disk Scheduling Algorithms

1. SCAN b) C SCAN

**Week 12**

Demonstrate operations on Multi Coreoperating System

.

**Week 1**

INSTALLATION OF WINDOWS

A clean install is intended for users who want to freshly install Windows on their computer (by deleting all of the data on the hard disk and then installing Windows) or computers that do not have an operating system yet .

**1.Enter your computer's BIOS.**

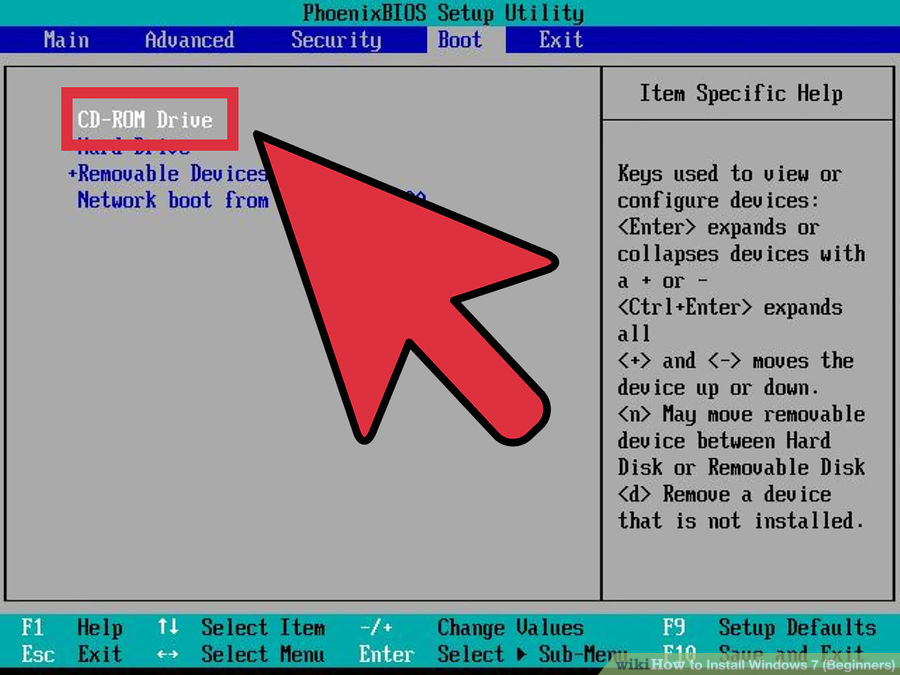
Turn off the computer that you want to install Windowson then turn it back on. When the BIOS screen appears or you are prompted to do so, press Del,Esc,F2,F10, orF9 (depending on your computer’s motherboard) to enter the system BIOS. The key to enter the BIOS is usually shown on the screen.



**2.Find your BIOS's boot options menu.**

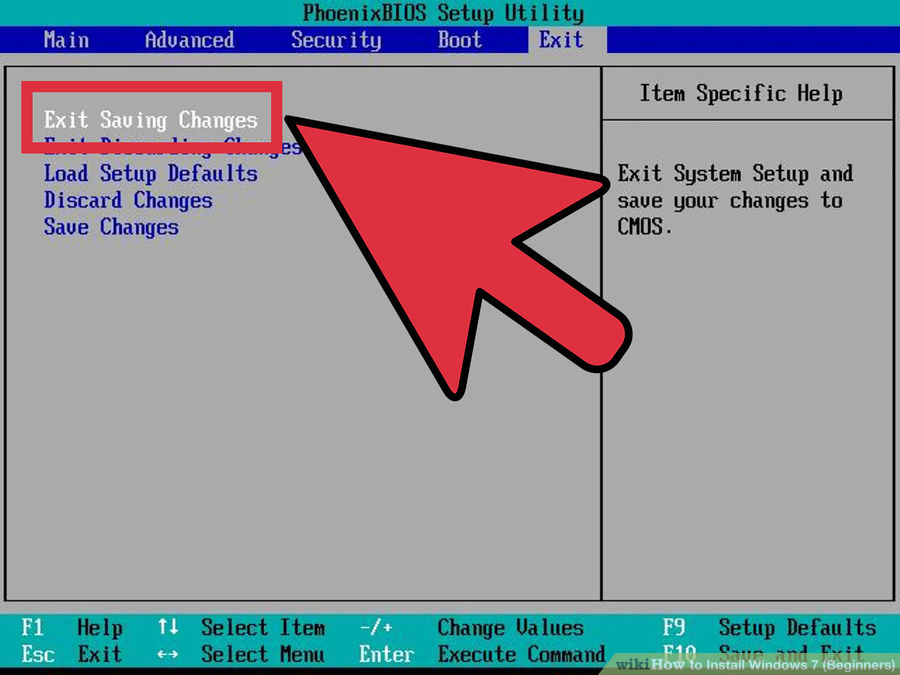
The boot options menu of your BIOS may vary in location or name from the illustration, but you may eventually find it if you search around.

* If you can't find the boot options menu, search the name of your BIOS (most likely located in the BIOS menu) online for help.



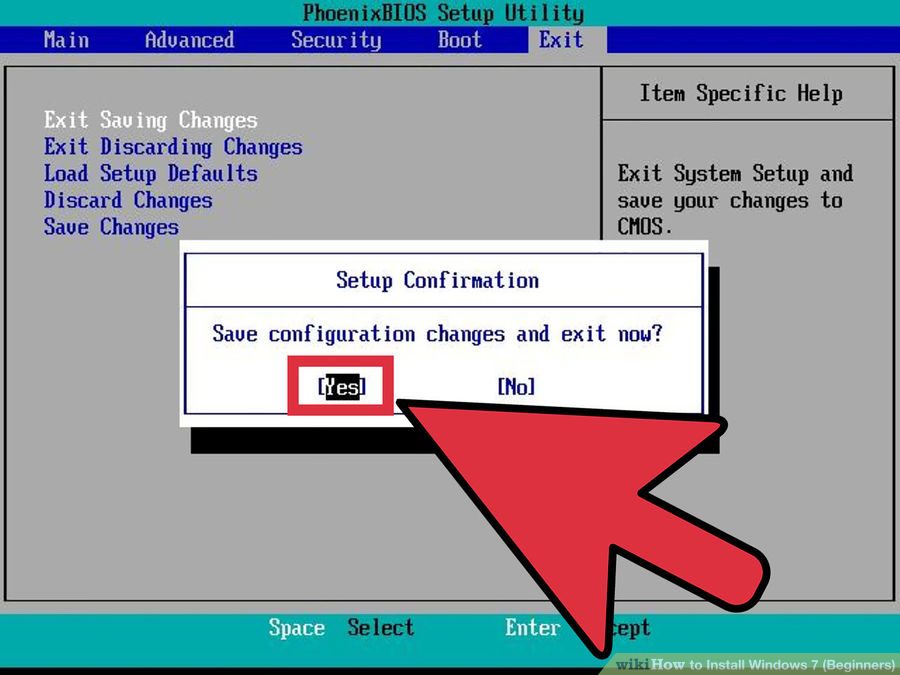
**3.Select the CD-ROM drive as the first boot device of your computer.**

* Although this method may vary among computers, the boot options menu is typically a menu of movable device names where you should set your CD-ROM drive as the first boot device. It can also be a list of devices that you can set the order of their boot on. Consult a manual or the internet for help if you're stuck.



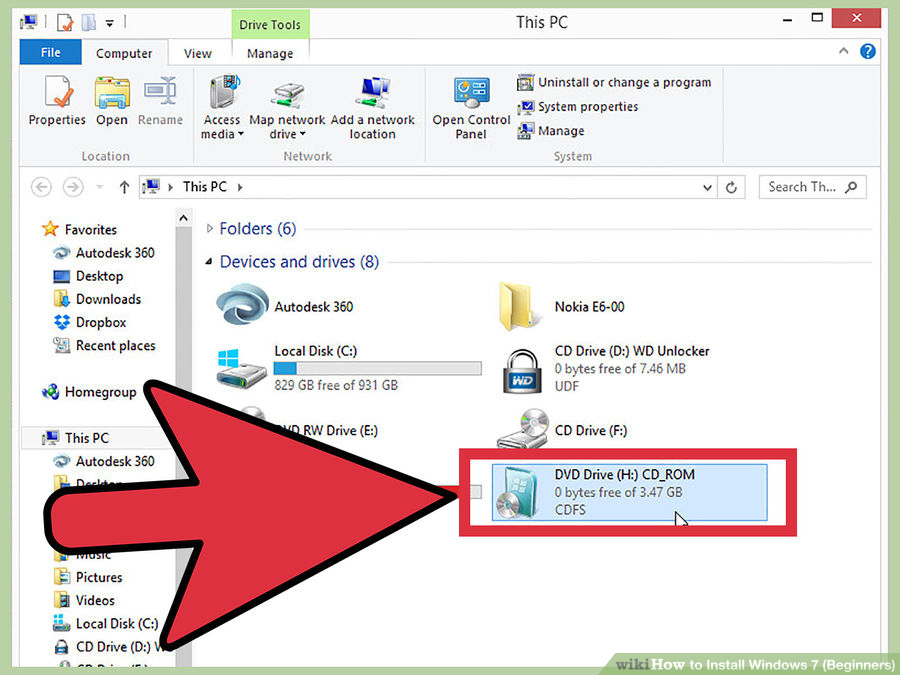
**4 Save the changes of the settings.**

Press the button indicated on the screen or select the save option from the BIOS menu to save your configuration.

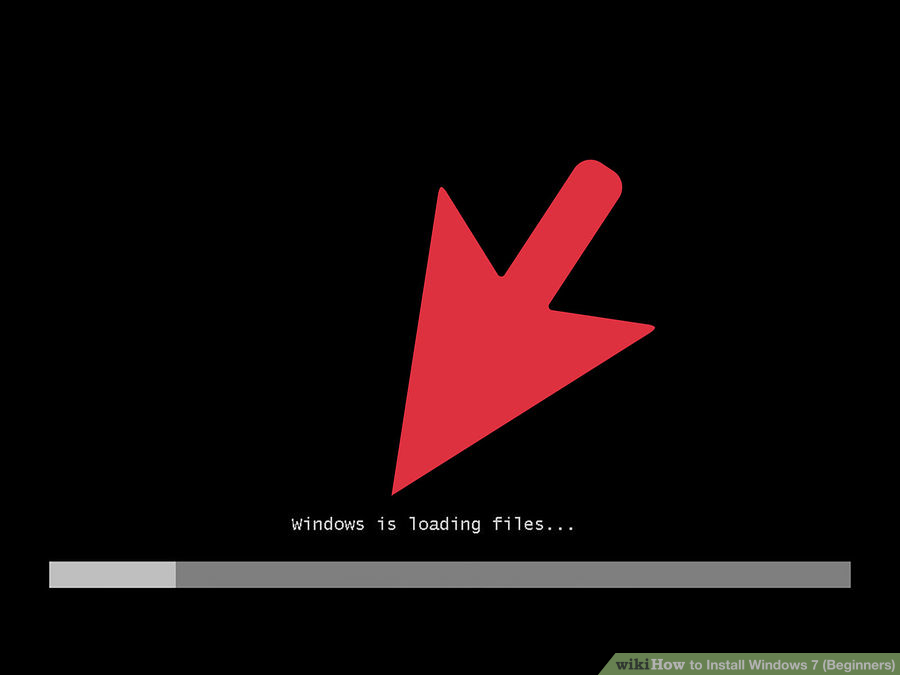


**5.Shut off your computer.**

Either turn off the computer by choosing the shut-down option in your current operating system, or hold the power button until the computer powers off.



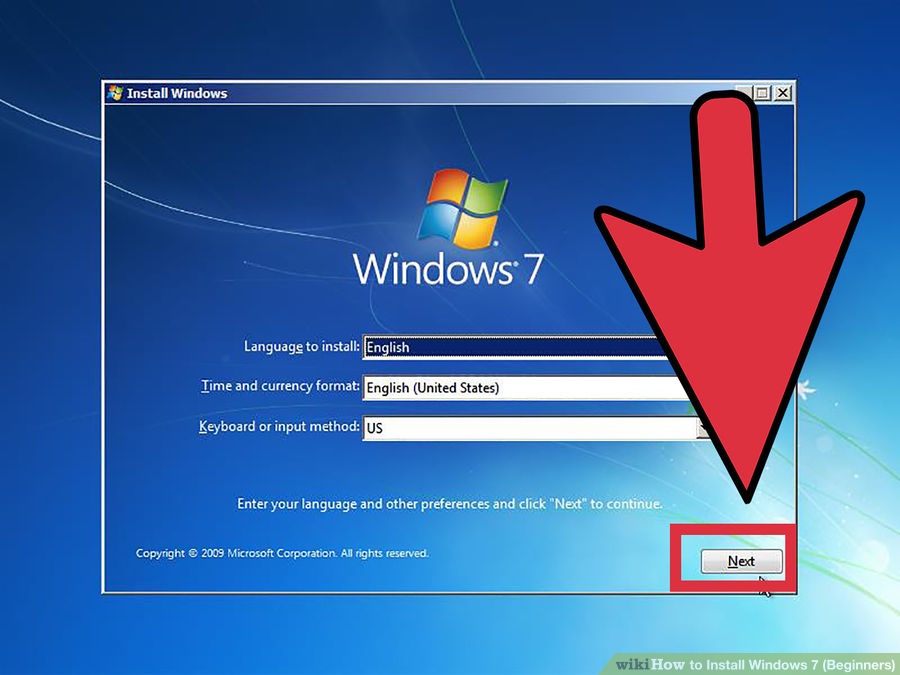
**6.Power on the PC and the insert the Windows 7 disc into your CD/DVD drive.**



**7 Start your computer from the disc**

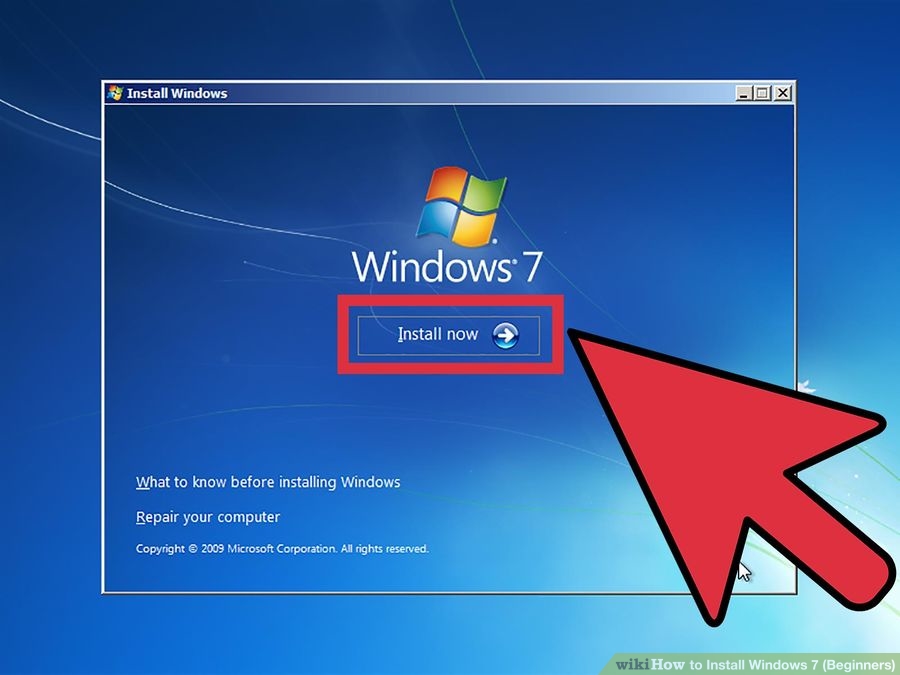
After you have placed the disc into the disc drive, start your computer. When the computer starts, press a key if you are asked if you would like to boot from the disc by pressing any key. After you choose to start from the disc, Windows Setup will begin loading.

* If you are not asked to boot from the disc, you may have done something wrong. Retry the previous steps to solve the problem.

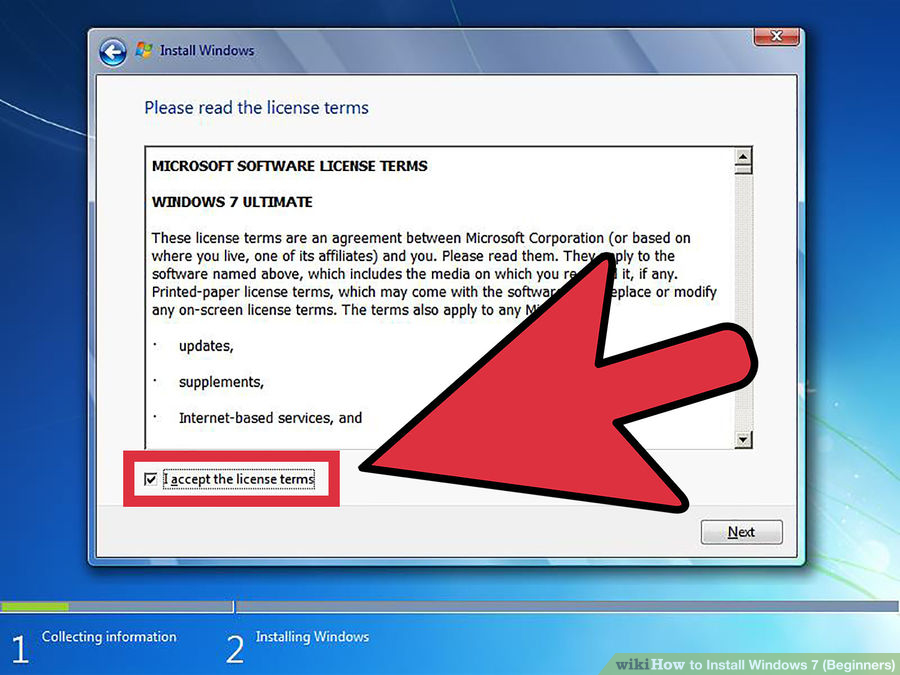


**8.Choose your Windows Setup options**

Once Windows Setup loads, you'll be presented with a window. Select your preferred language, keyboard type, and time/currency format, then click *Next*.

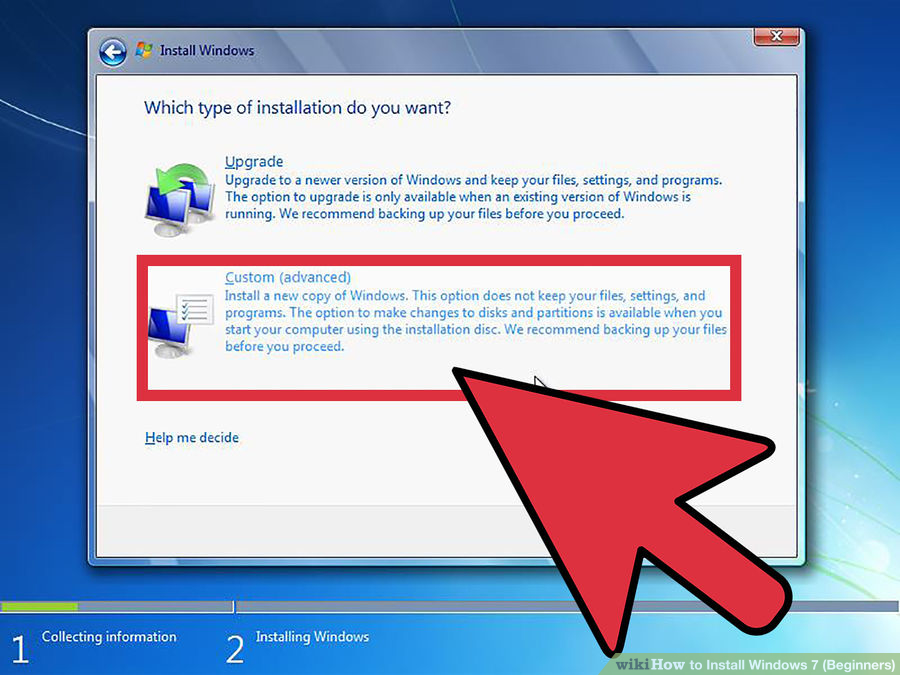


**9.Click the *Install Now* button.**



10.**Accept the License Terms.**

Read over the Microsoft Software License Terms, check.*I accept the license terms*, and click *Next*.



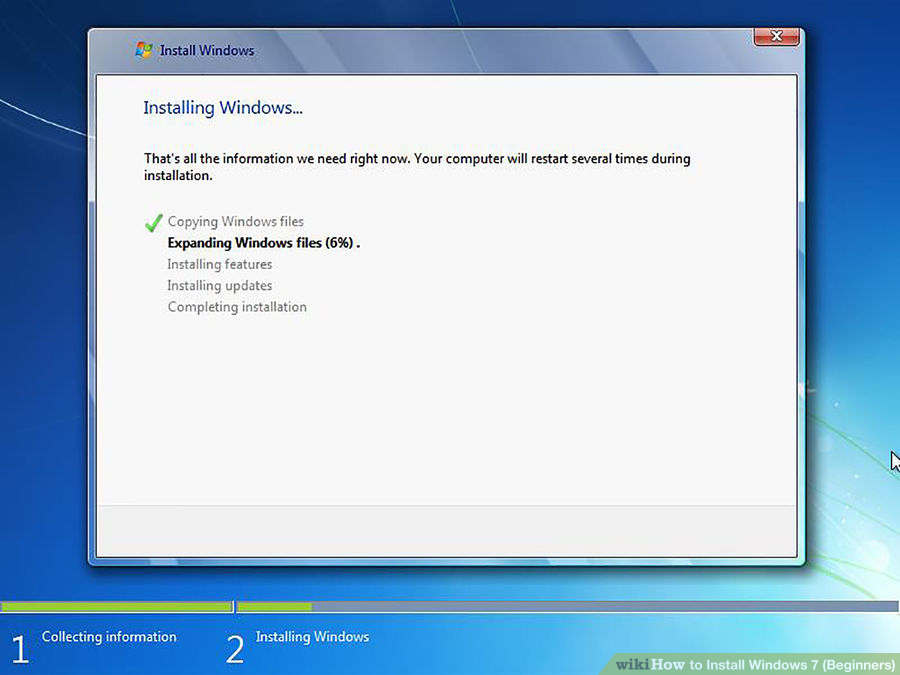
**11.Select the*Custom* installation.**



**12 Decide on which hard drive and partition you want to install Windows on.**

A hard drive is a physical part of your computer that stores data, and partitions "divide" hard drives into separate parts.

* If the hard drive has data on it, delete the data off of it, or **format** it.
  + Select the hard drive from the list of hard drives.
  + Click *Drive options (advanced)*.
  + Click *Format* from Drive options.
* If your computer doesn't have any partitions yet, create one to install Windows on it.
  + Select the hard drive from the list of hard drives.
  + Click *Drive options (advanced)*.
  + Select *New* from Drive options.
  + Select the size, and click *OK*.

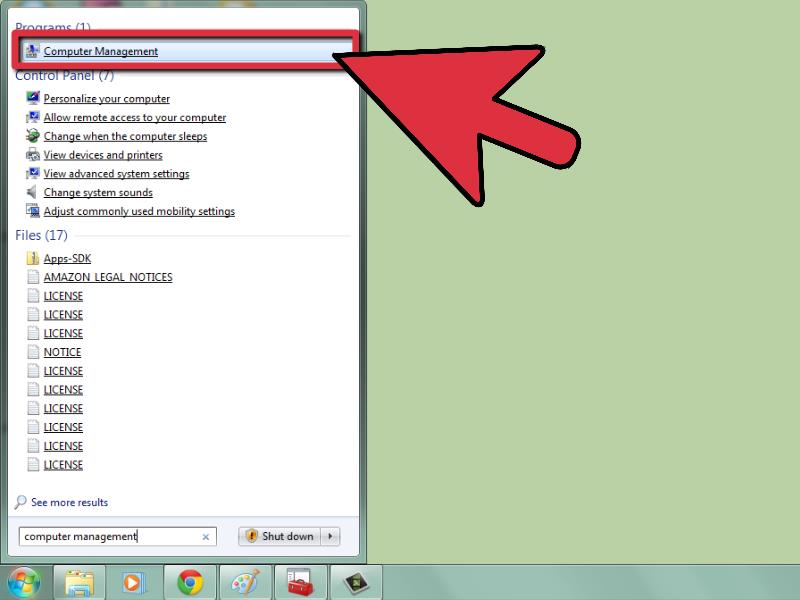


**13 Install Windows on your preferred hard drive and partition.**

Once you've decided on where to install Windows, select it and click*Next*. Windows will begin installing.

**DISK PARTITIONING(WINDOWS)**

Disk partitioning is to divide the hard drive into multiple logical units. People don’t often choose to partition their hard disks, but it has many benefits. Mainly, by partitioning your disk, you can separate your operating system from your data and thus reduce the chances of your data becoming corrupted.

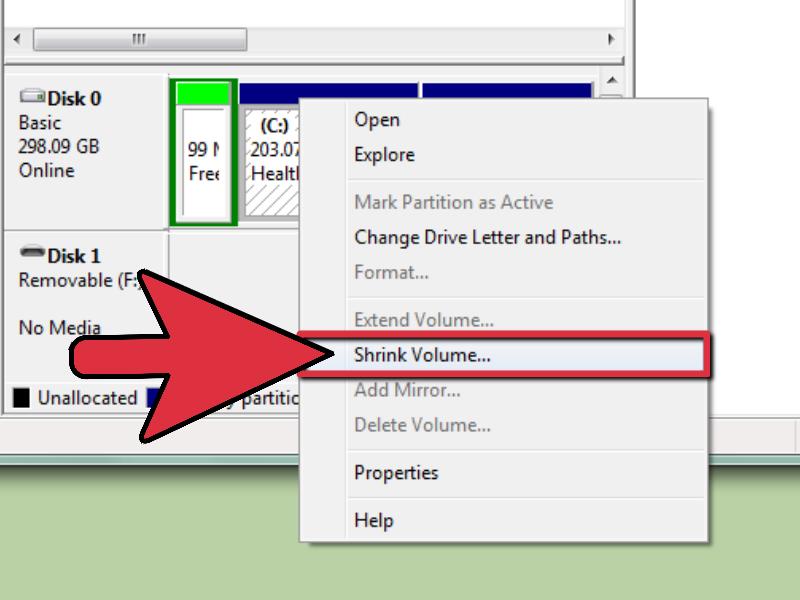


**1 Open the Computer Management tool.** Open the Start menu. Type “Computer Management” in the search bar of the Start menu and press enter.



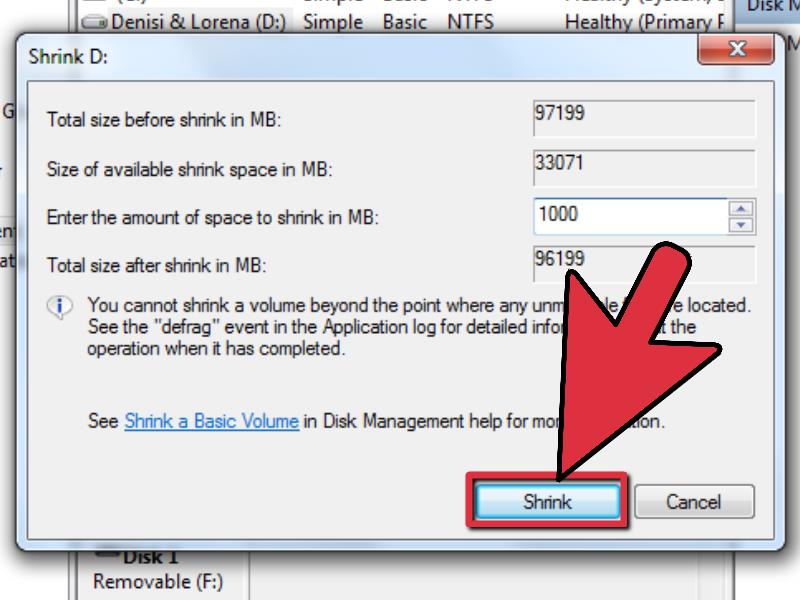
**2 Select the Disk Management tool.** Click on **Disk Management** on the left side of the window and you should see all of the disks and their partitions on your computer.

* In the example in the picture, there is 1 disk with two partitions.



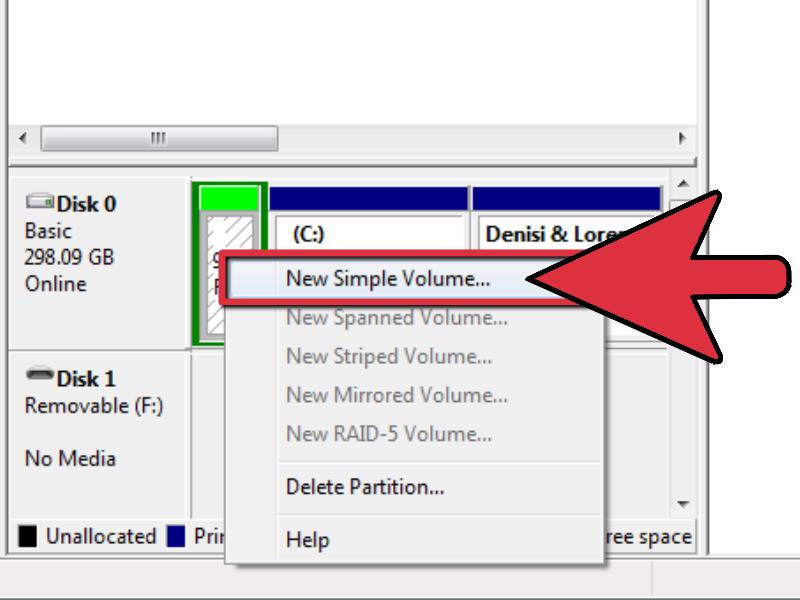
**3 Make some space for the new partition.** Right-click on the partition you wish to re-size and select the **Shrink Volume** option.

* In the example in the picture the *(C:)* drive is used.**Note:** There may be a partition named **System Reserved**. It is not recommended that you alter this partition at all.



**4 Shrink the drive.** Enter the size you wish to shrink your drive to in megabytes (1000 MB = 1GB). Then click on the **Shrink** button.

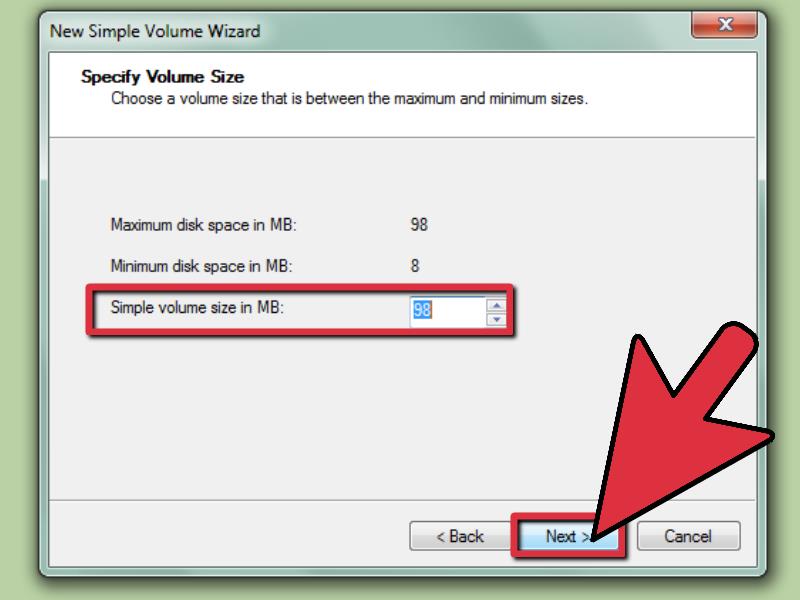
* In this example the drive is shrunk by 10000 MB or 10 GB.
* **Note:** You cannot shrink your volume greater than the amount indicated in the *Size of available shrink space in MB* section.



**5 Create a new volume.** You should now see a new unlocated partition in your *Disk Management* window. Right-click on the **unallocated** partition and select the **New Simple Volume** option.

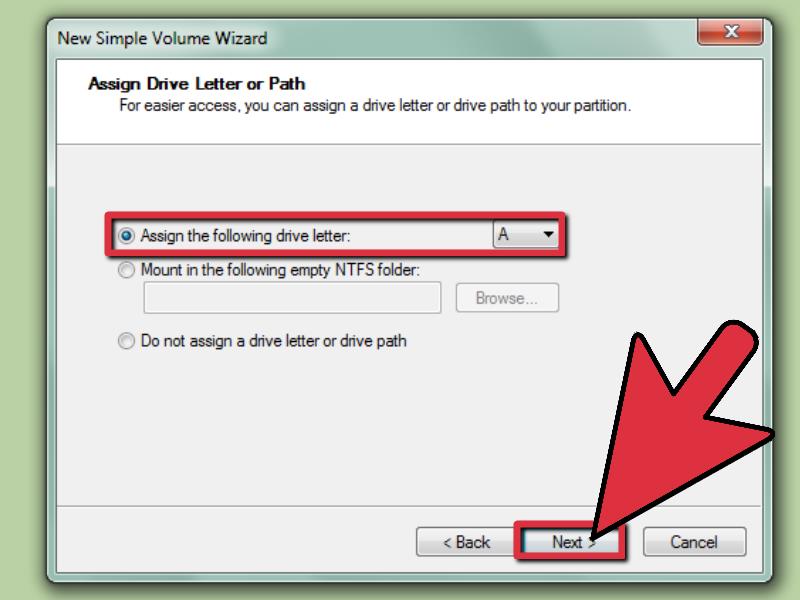


**6 The New Simple Volume Wizard.** The *New Simple Volume Wizard* should popup. Click on the **Next** button to continue.



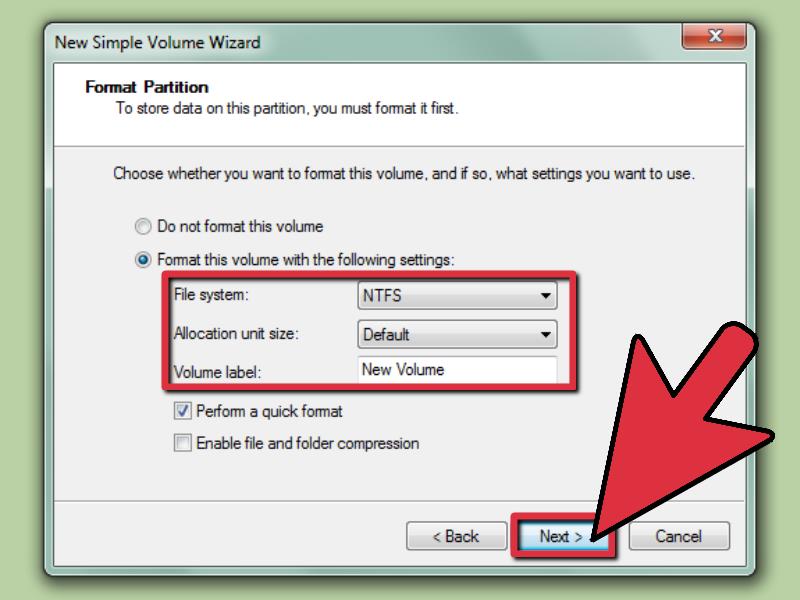
**7 Enter size of new partition.** Enter the amount of memory you wish to allocate for your new partition and click on the **Next** button.

* In the example in the picture, the maximum memory available is allocated to the new volume.
* **Note:** You cannot make your new volume larger than the maximum amount of memory available.



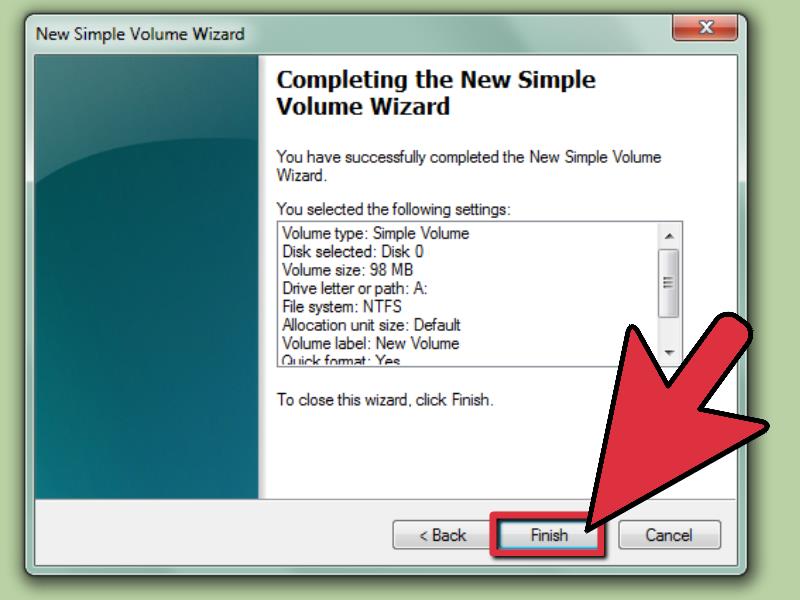
**8 Give the new volume a letter name or path.** Select from the menu, a letter name for your new partition and click on the “Next” button.

* The letter name chosen for the example in the picture is *(E:)*
* The letter name or path is used by Windows to identify and navigate to your new volume.

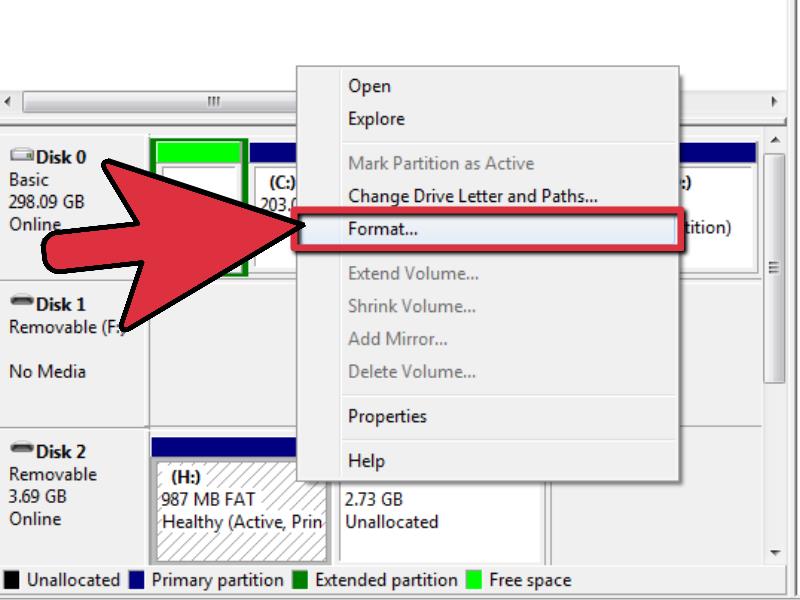


**9 Settings for the new volume.**

* Click on the *Format this volume with the following settings:*
* For **File System**, select **NTFS**
* For **Allocation unit size**, select **Default**
* For **Volume Label**, type the name you wish to give your new drive.
* Click on the *Perform a quick format*
* Then click on the **Next** button

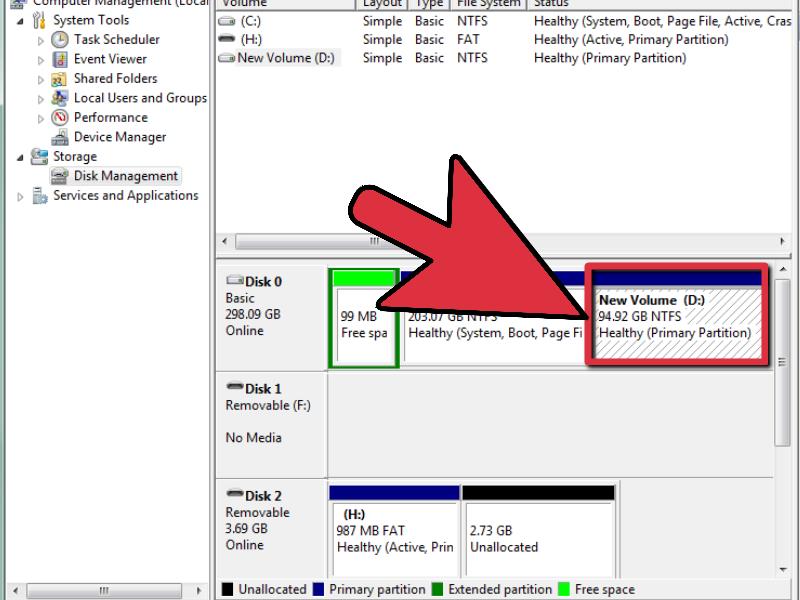


**10 Create the new volume.** Look over your settings and click on the **Finish** button



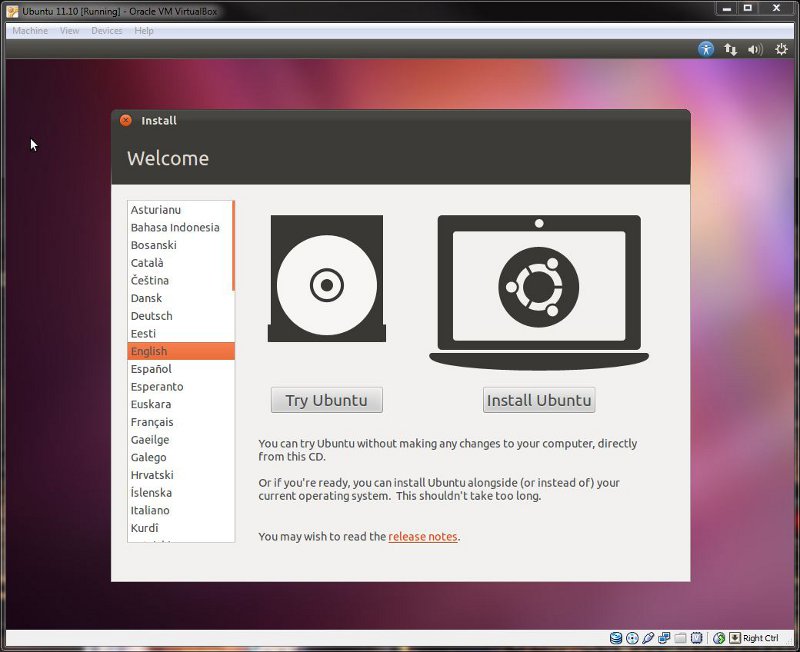
**11 Format the new volume.**

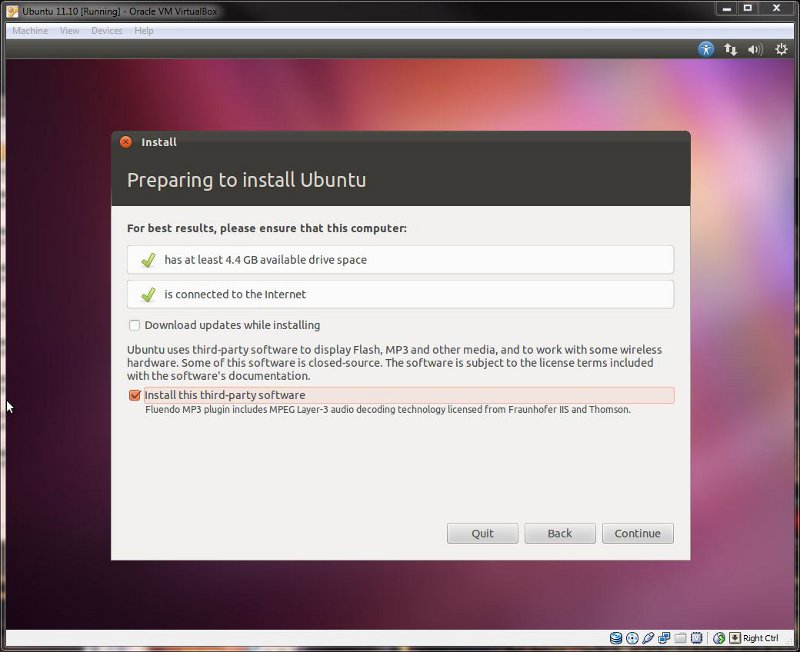
* You will get a popup asking you to partition your new drive. Click on the **Format disk** button.
* A new window will popup. Keep the settings and click on the **Start** button.
* A warning will popup. Click on the **OK** button.

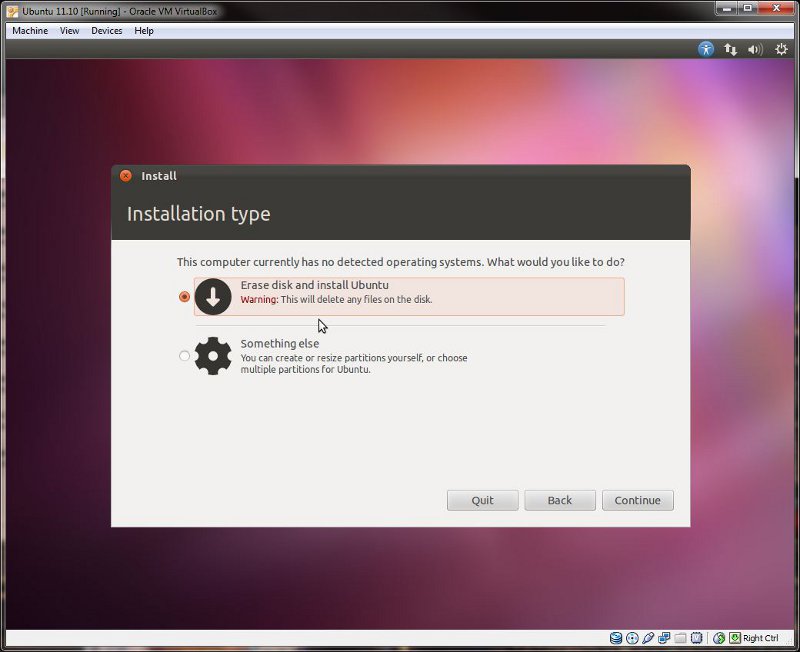
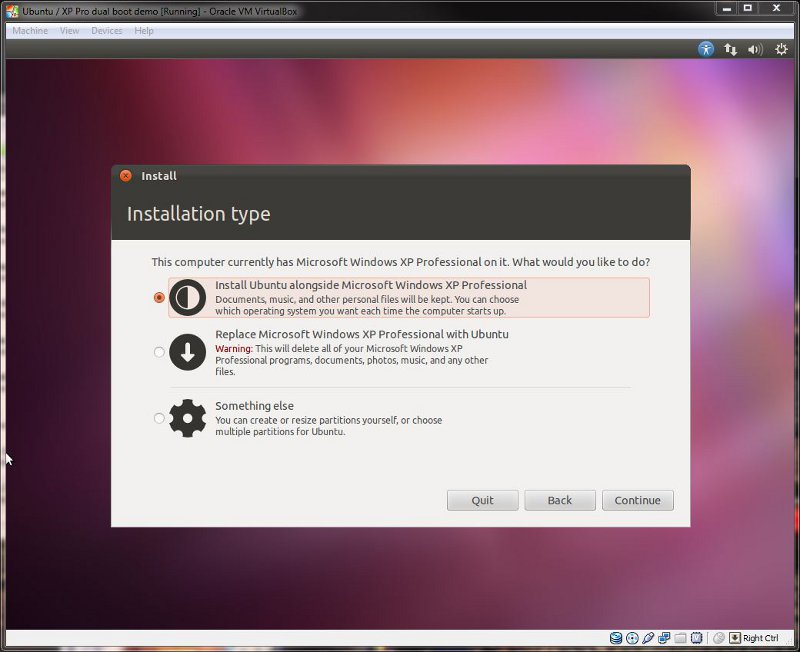
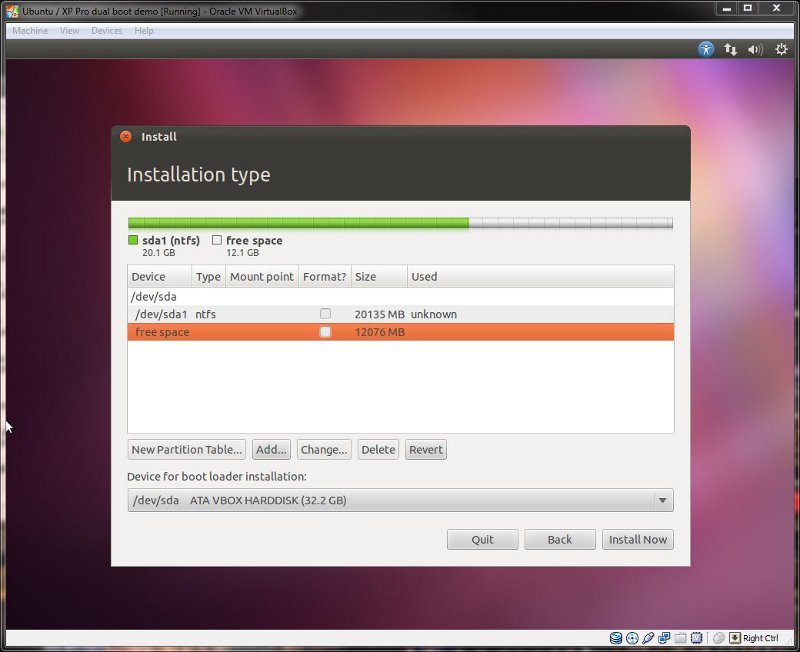
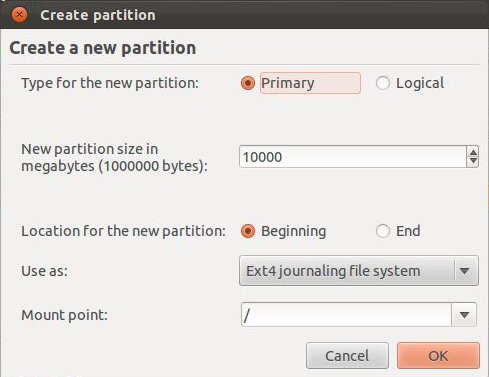


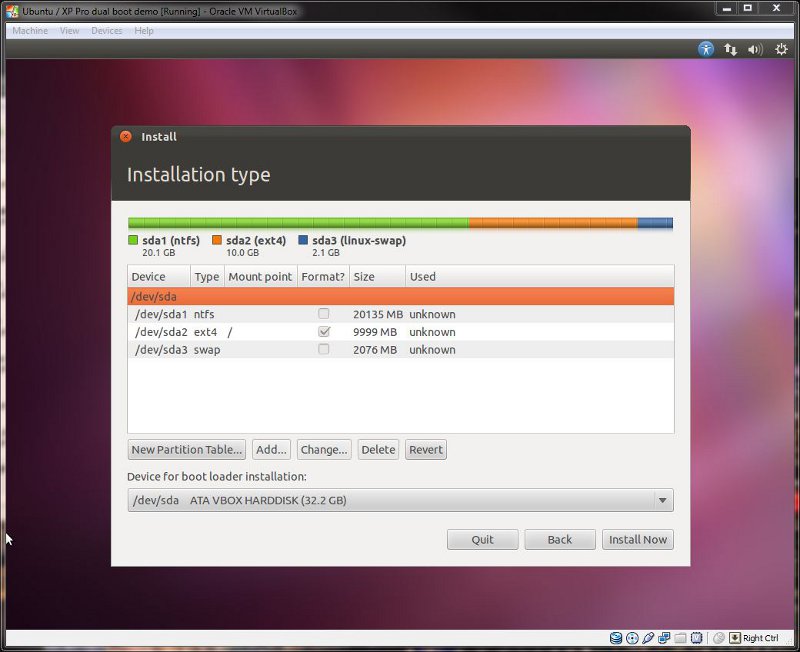
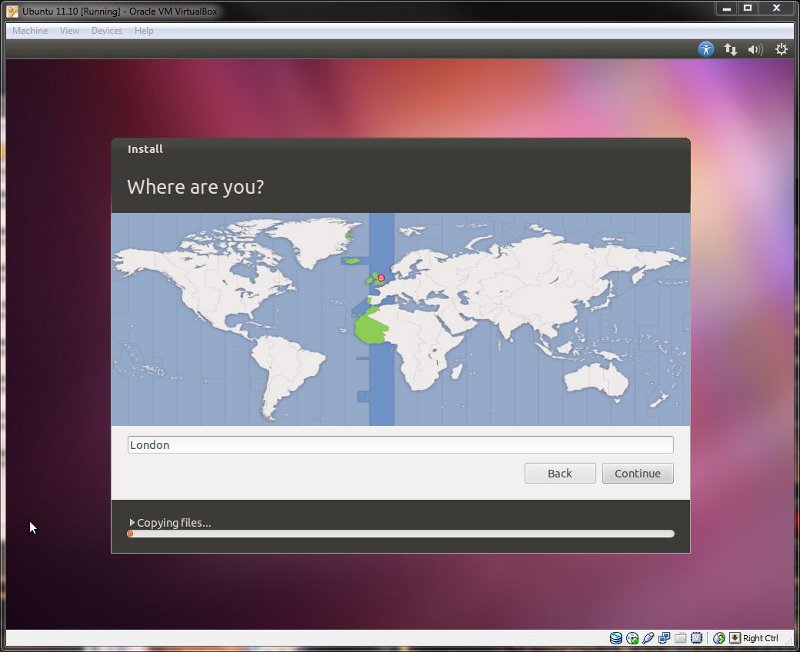
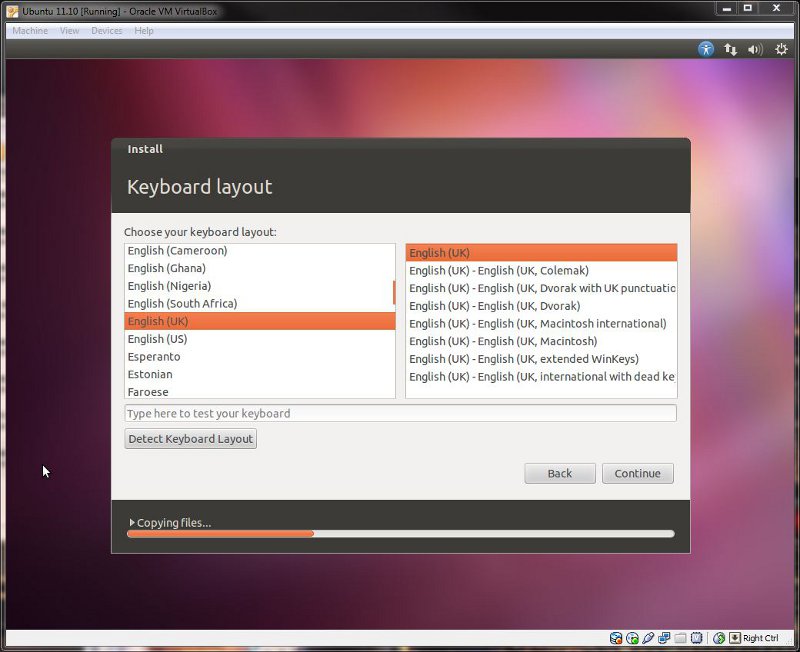
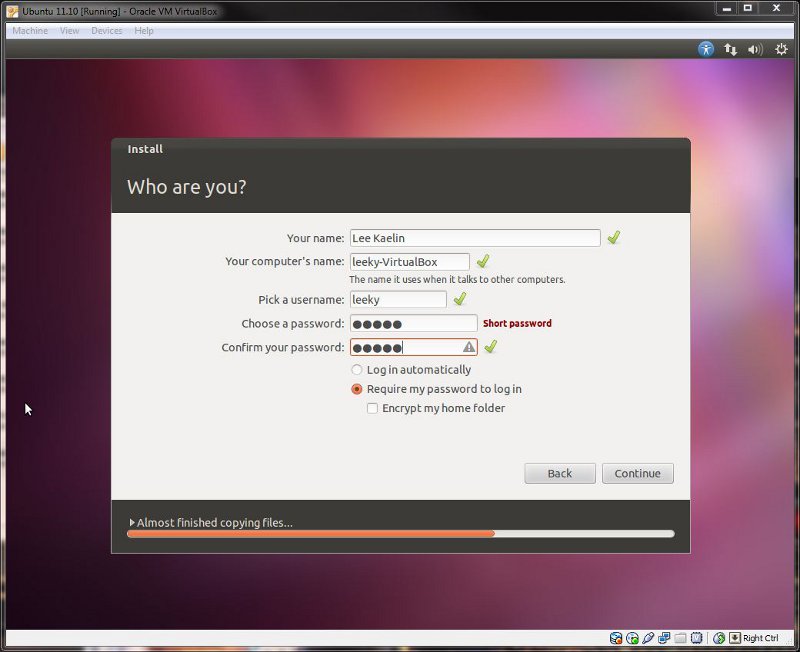
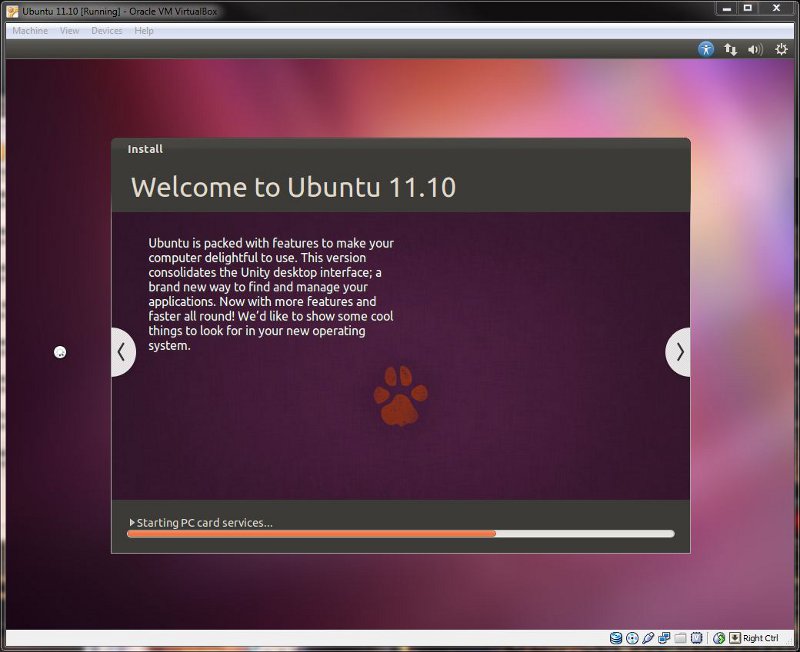
**12 Check new volume.** If everything was done correctly, you should now see your new drive in the *Disk Management* window.

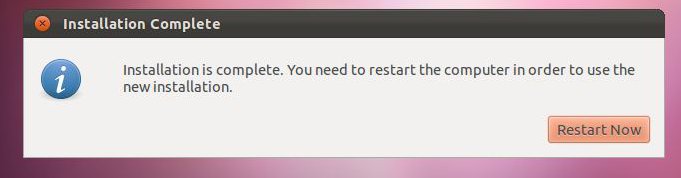
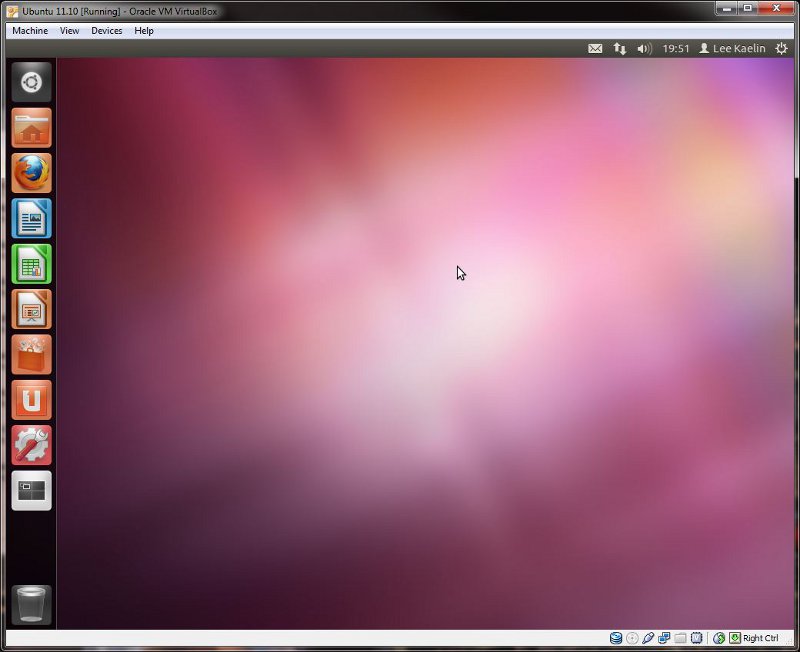
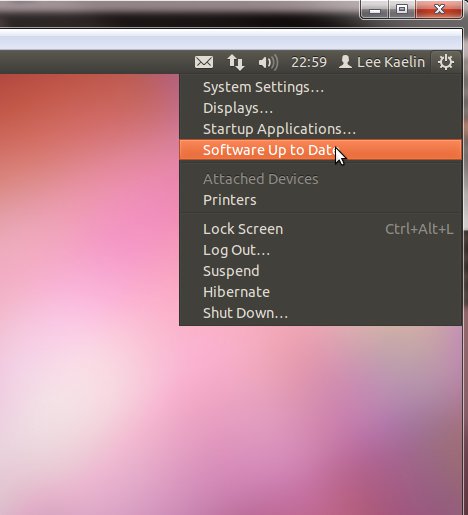
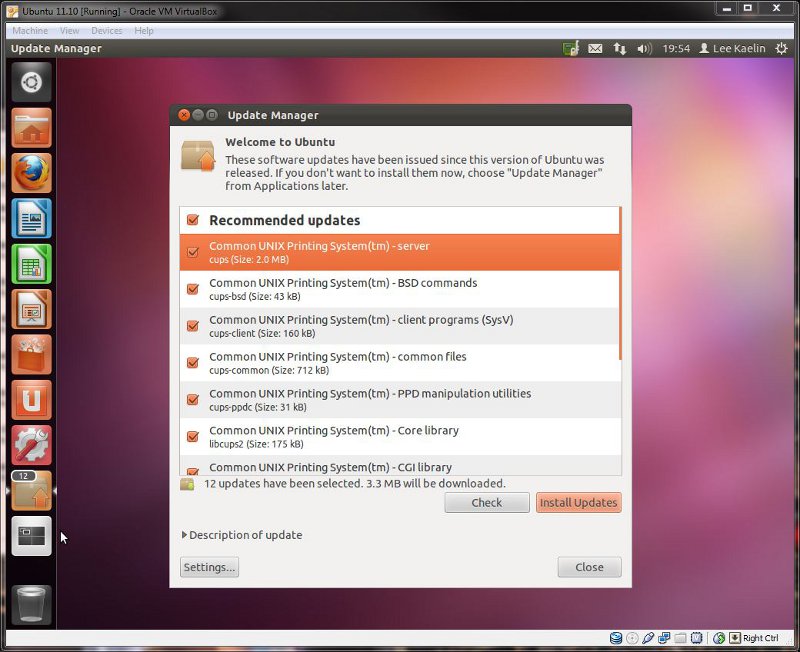
**Installation of Ubuntu Linux 11.10 from a LiveCD**

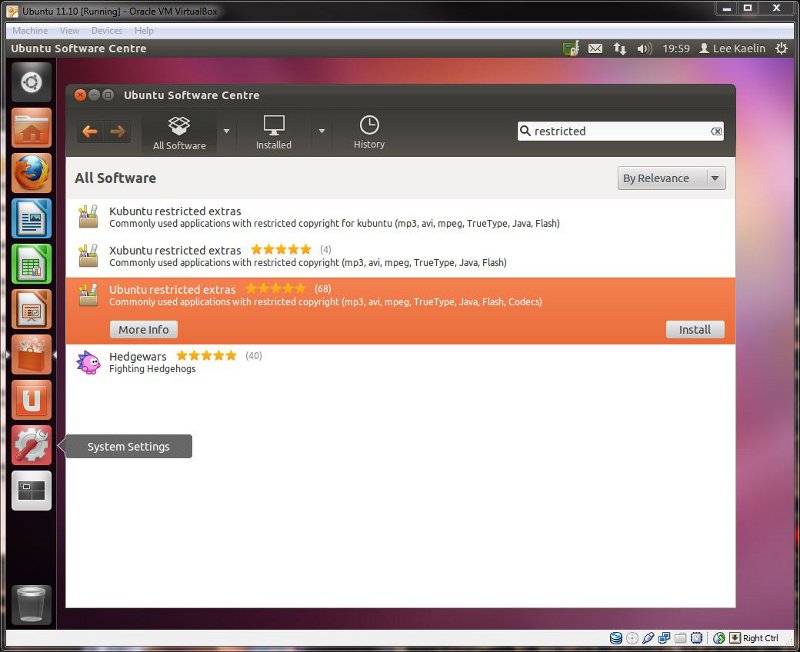
**Step 1:**  
The first thing you should do is head to <http://www.ubuntu.com/download/ubuntu/download> and download Ubuntu 11.10 LiveCD. Just click the big orange Start download box.  
  
**Step 2:**  
Using your disc burning software, burn the .iso you downloaded to a CD.   
  
**Step 3:**  
Before you go any further, ensure all important data is backed up in case of data loss on your drives. This guide assumes you have media backups of your Windows partitioned hard drive and you are safe to proceed.  
  
*\*\*\*Warning: Installing another operating system without first ensuring you have backups of your current files and operating system is a big risk. If you have no data to lose or you’ve backed up important data, you’re ready to proceed. YOU are responsible if you lose data.For those of you using Windows, and installing Linux for the first time I recommend you either use a separate hard disk that does not contain the Windows OS, or create a partition big enough for Linux within Windows using Disk Management in the Administrative Tools menu of the control panel. 30GB of hard disk space is absolutely plenty of space for you to begin exploring Ubuntu whilst at the same time having room to grow.*  
  
**Step 4:**  
Ensure you have a network cable connected, restart your computer, and boot from the CD drive.  
  
**Step 5:**  
The LiveCD will load up, and you'll be presented by the following box:  
  
  
  
For the purposes of this guide, we will assume you've already tried Ubuntu and want to proceed with an installation, so click "Install Ubuntu."  
  
**Step 6:**  
You'll be greeted by the "Preparing to install Ubuntu" screen, exactly as below:



I recommend you select "Install third-party software" as I have done in the screenshot above. I prefer to do system updates once up and running, but if you have the extra time you can also select "download updates while installing" as well. Then click continue.  
  
**Step 7:**  
The next screen you will see is "installation type," what you see will be dependent on whether you have an existing Windows installation or not.   
  
I'm going to split this into three different sub-steps, to make it as simple as possible.  
**Step 7-A:**  
For those installing in a virtual machine or to hard disks without an OS you will see the following screen:  
  
  
  
You have two choices:  
1. Erase the entire disk and use all of it for installation -- Ubuntu will automatically partition your disk and proceed with installation.  
2. Select "something else" and manually create your partitions (which is covered in detail in step 7-C).  
If you are choosing the first option, select the radio button and then click continue, proceeding to step 8.  
**Step 7-B:**  
Those of you that have current Windows installations or are going to dual-boot with another existing OS will be presented with a screen similar to below:  
  
  
  
You have three options available:  
1. You can choose the first option and install Ubuntu alongside your existing OS.   
2. You can opt to replace your Windows installation with Ubuntu, allowing the installer to format your current partitions and automatically create new ones for Linux.  
3. You can choose "something else" and create your own partition scheme and sizing (covered in detail in step 7-C).  
Once you have selected which route you wish to proceed with click continue and proceed to step 8.  
**Step 7-C:**  
Having selected the "something else" option you will be presented with the following window:  
  
  
  
Linux recognizes and assigns IDs to drive in a different manner to Windows. In the above image, you can clearly see my hard disk in the list. It is identified by "/dev/sda." Linux recognizes drives in the following way:   
• Sda = 1st drive  
• Sdb = 2nd drive  
• Sdc = 3rd drive and so on   
  
Partitions are also shown after the drive letters. So if I had 2 partitions on my first disk, they would be identified as:  
• Sda1 – 1st drive, 1st partition  
• Sda2 – 1st drive, 2nd partition  
  
You will not see the common Windows C: label in the disk menu in the above list. You do, however, have key things to help you recognize your Windows C: drive. Both of these can be used to identify which is your Windows disk.  
• The size of the disk is shown  
• The name of the drive is shown  
  
This is your current partition layout for your hard disks. If you have more than one disk, they will show up as /dev/sda, /dev/sdb etc.  
  
Firstly, identify your Windows installation. In my case, it's sda1 (which is my first hard disk, first partition). What you see depends on how you created the extra space. I just resized the Windows partition from within Windows, and left the free space ready to install Linux. I recommend using Windows or a free utility from within Windows to resize your partition as most beginners will understand it more.  
  
Therefore, we now need create a minimum of two partitions:  
  
Click "Add" and the following box will appear:  
  
  
  
You will notice I have already filled out the example above to create a 10GB root partition.  
  
You can have a maximum of 4 primary partitions, or 3 primary partitions and 1 logical (which allows for another 64 partitions)  
The size above is 10.00GB. e.g 1,000 = 1GB 10,000 = 10GB (Remember to leave enough free remaining space to create your SWAP partition!)  
Location for new partition: e.g. do you want it at the start or end of the free space. Select beginning.  
Use as: Ext4 is the recommended file system for Ubuntu, much the same as NTFS is Windows. SWAP is for SWAP space.  
Mount point: This is where you want the partition to mount. E.g. we need a root partition, which in Linux is denoted by a "/".  
  
Click OK once you have finished setting the partition information and you will return to your partition screen, now showing the root partition you just created. Using the same methods as before, create a SWAP partition.   
  
I recommend you set the size of your SWAP partition to at least the size of your available RAM. If you have plenty of hard disk capacity I would suggest you use double the size. So if you have 2GB of RAM, set it to either 2GB or 4GB. For best performance it is recommended you have your SWAP partition at the beginning or end of your drive.

Once you have done that, you should be looking at something like below.  
  
  
So to re-cap the above, (in my example) we have the following:  
• /dev/sda1 is your Windows partition.  
• /dev/sda2 is your new root partition (Windows equiv. of C:).  
• /dev/sda3 is your SWAP space.Once you are happy with the changes you have made, click install now and proceed to the next step.  
  
**Step 8:**  
As the installation starts to copy the required files to the hard disk, you will be presented with a screen to select your locale. It should automatically find where you are, as it has for me already:  
  
  
  
Just double check it is correct, and then select continue.  
  
**Step 9:**  
The next screen to appear will be keyboard layout:  
  
  
  
Ensure the correct option is selected, above you will see the correct (and default UK) selection has been automatically made for me.  
  
**Step 10:**  
You will now be greeted by the "who are you" screen, ready for you to fill out with your user details:  
  
  
  
The computers name and username will automatically populate when you type your full name. You can however edit them as you please. Fill in the details and then click continue.  
  
You can opt to have Ubuntu automatically log in for you -- even with a password set -- or you can choose the traditional option requiring a password to log in. You really don't need to choose the encrypt option unless you're installing on a laptop and are dealing with highly secure information.  
  
**Step 11:**  
The installation information screens will now appear as Ubuntu continues the installation:  
  


**Step 12:**  
Once installation has finished, you will be presented with the following box:  
  
  
  
Select "restart now" and when requested, remove your installation CD, then press enter to reboot.  
  
**Step 13:**  
For those of you that have Ubuntu as the only OS the computer will boot directly into Linux. If you're dual-booting, you will see the GRUB menu appear similar to below:  
  
Hit enter, to select the first option and load your newly installed Ubuntu OS.  
  
**Step 14:**  
For those that elected to automatically log into Ubuntu, you will go straight to the desktop in Step 15. For everyone else, you will be greeted with the new login manager for Ubuntu:  
Enter your password, and hit enter to login to your desktop.  
  
**Step 15:**  
Your desktop should look like this:  
  
  
  
  
**Step 16:**  
Before we proceed further, let's check for updates. Click on the power button on the top right corner of the screen and select "check for updates," or words to that effect.  
  
  
  
  
I'd already run updates on this install so the example above is displaying "software up to date," but the picture highlights where you need to select anyway.   
  
Upon selecting the update option, the update manager will appear, as below:  
  
  
  
  
If it comes up with no available updates, just select "check" again to verify that it is correct. Having done the same thing myself, I was presented with the updates you see above. For those that opted to install updates during installation it is unlikely there will be further updates required.  
  
Once complete select close, and restart Ubuntu.The power button is located on the top right corner of the screen. Click this and select shutdown.

**Step 17:**  
No install is complete without full support for mp3s, core MS fonts, DVD playback codecs, Flash and Java, so let'sgo ahead and sort this now.  
  
Click the black Ubuntu menu button at the top left corner of the screen and in the menu that appears, type "software centre" and select the Ubuntu Software Centre. Once open, click the search bar, type "restricted" and the following should appear:  
  
  
Select Ubuntu restricted extras, and then click on the install button.   
Authentication is much like UAC (user access control) in windows Vista and 7. It is required to elevate your user privileges to that of root (Linux administrator).   
  
Ubuntu restricted extras will now download, sort any dependencies and install. You can check its progress by viewing the progress bar above the install button. Once finished the In Progress tab will disappear -- restart Linux.It's not strictly necessary, but I always do it after installing this package so everything can start up properly.

**Week 2**

### **What is Linux?**

Just like Windows XP, Windows 7, Windows 8, and Mac OS X, Linux is an operating system. An operating system is software that manages all of the hardware resources associated with your desktop or laptop. To put it simply – the operating system manages the communication between your software and your hardware. Without the operating system , the software wouldn’t function. Linux is one of popular version of UNIX operating System. It is open source as its source code is freely available. It is free to use. Linux was designed considering UNIX compatibility. Its functionality list is quite similar to that of UNIX.

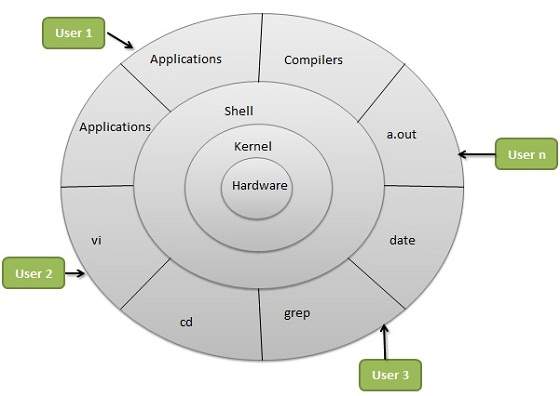
## Basic Features

Following are some of the important features of Linux Operating System.

* **Portable** − Portability means software can works on different types of hardware in same way. Linux kernel and application programs supports their installation on any kind of hardware platform.
* **Open Source** − Linux source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
* **Multi-User** − Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
* **Multiprogramming** − Linux is a multiprogramming system means multiple applications can run at same time.
* **Hierarchical File System** − Linux provides a standard file structure in which system files/ user files are arranged.
* **Shell** − Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.
* **Security** − Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

## Architecture

The following illustration shows the architecture of a Linux system −



The architecture of a Linux System consists of the following layers −

* **Hardware layer** − Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).
* **Kernel** − It is the core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.
* **Shell** − An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes kernel's functions.
* **Utilities** − Utility programs that provide the user most of the functionalities of an operating systems.

**Advantages**

* **Cost**
* **Security**
* **Choice (Freedom)**
* **Software**
* **Hardware**

**Disadvantages of Linux:**

* **Understanding**
* **Compatibility**
* **Alternative Programs**

The OS is comprised of a number of pieces:

* **The Bootloader,The kernel,Daemons,The Shell,Graphical Server,Desktop Environment,Applications**

# **Login to your Account**

## login: You type your ID and RETURN.

## Password: You type your password and RETURN. It does not appear.

## $ The UNIX prompt (or similar).

## You can now enter commands.

## Changing your Password

## The command is:

### **passwd**

## It will ask you for the new password twice

# Logout from your Account

## logout or ^D Press CONTROL and D together or exit

# **Unix Commands**

# **Working with Directories**

**mkdir** This command will make a new subdirectory.

Use this command to create a directory.

$ **mkdir kalyan**

To create a subdirectory named morestuff in the existing directory named /tmp, enter:

mkdir /tmp/morestuff

# **Moving between Directories**

**Cd** change directory

cd kalyan will change directory from current directory to kalyan directory.

Use pwd to check your current directory and ls to see if kalyan directory is there or not.

You can then use cd kalyan to change the directory to this new directory.

## s3910120’s home directory:

## If you are in directory s3910120 how do you move to directory proj1?

### **cd proj1**

## You are now in proj1. This is called the current working directory.

## pwd Print name of current working directory

## Move back to directory s3910120 (the parent directory):

### **cd ..**

## When in proj1, move to proj2 with one command:

### **cd ../proj2 ../proj2 is a *relative* pathname**

**rmdir** **This command will remove a subdirectory**

Use this command to remove a directory. A directory must be empty before it can be removed. To empty a directory, use [rm.](http://www.math.utah.edu/lab/unix/unix-commands.html#rm#rm)

To remove a subdirectory named oldstuff, enter: rmdir oldstuff

**Note:** The directory you specify for removal must be empty. To clean it out, switch to the directory and use the ls and rm commands to inspect and delete files.

### rm **Use rm to remove files from your directory.**

This command will remove (destroy) a file. You should enter this command with the  -i  option, so that you'll be asked to confirm each file deletion.

To remove a file named junk, enter:

rm -i junk remove junk? **y**

**Note:** Using rm will remove a file permanently, so be sure you really want to delete a file before you use rm.

To remove a non-empty subdirectory, rm accepts the  -r  option. On most systems this will prompt you to confirm the removal of each file. This behavior can be prevented by adding the  -f  option.

To remove an entire subdirectory named oldstuff and all of its contents, enter:

rm -rf oldstuff

**ls** list the contents of a directory   
ls [options] [directories]  
the current working directory used if no directories specified  
A few options:

* + -a    list all entries includeing hidden files (starting with .)
  + -i    print inode numbers
  + -l    long list (mode, links, owner, group, size, timeof last modification,

and name

* + -t    sort by modification time
  + -x    multi-column list, sorted across each row

# Some System Directories

## / root directory

## /bin commands

## /etc system data files (e.g. /etc/passwd)

## /dev files representing I/O devices

/home files representing users home directories

# ***Working with Files***

**cat > filename :** To create a new file

ex: $ cat > file1

This is my first file.

^D

note: To save the file after entering data press CONTROL+D.

**cat** >> filename : To append the contents to a file.

Ex:

**$ cat** >> file1

This is addition of some content to already existed file.

**cat filename** : To view the contents of a file

ex:$ cat file1

This is my first file.

This is addition of some content to already existed file.

$ ***cat files*** *read file(s)****cat > file*** *create file (reads form terminal; terminate input with ^D)****cat >> file*** *append to file (reads form terminal; terminate input with ^D)****cat file2 >> file1*** *appends contents of file2 to file1*

**cat** file1 file2 > file3 to combine two files.

# ***Filename Conventions***

## Many files have a name and an extension:

### **file.c A C program**

### **file.cpp A C++ program**

### **file.txt A text file**

## However, you can call a file anything. It doesn’t have to have an extension.

# ***Other Commands on files***

**Cp**  Use cp to copy files or directories.

This command copies a file, preserving the original and creating an identical copy. If you already have a file with the new name, cp will overwrite and destroy the duplicate. For this reason, it's safest to always add  -i  after the cp command, to force the system to ask for your approval before it destroys any files.

The general syntax for cp is: cp -i oldfile newfile

cp foo foo.2 This makes a copy of the file foo.

cp ~/poems/jabber .

This copies the file jabber in the directory poems to the current directory. The symbol "." stands for the current directory. The symbol "~" stands for the home directory.

The  ~  character (tilde) is interpreted as the path of your home directory.

**Mv** Use this command to change the name of file and directories.

This command will move a file. You can use mv not only to change the directory location of a file, but also to rename files. Unlike the cp command, mv will not preserve the original file.

**Note:** As with the cp command, you should always use  -i  to make sure you do not overwrite an existing file.

To rename a file named oldname in the current directory to the new name newname, enter:

mv -i oldname newname

% mv foo foobar

The file that was named foo is now named foobar

To move a file named hw1 from a subdirectory named newhw to another subdirectory named oldhw (both subdirectories of the current directory), enter:

mv -i newhw/hw1 oldhw

## rm file Delete file rm -i file Double-check first

* **cal** display a calendar   
  cal [month] year
  + month   number between 1 and 12
  + year      number between 1 and 9999

**Examples:**  
**cal 1996**      print calendar for year 1996  
**cal 1 1997**   print calendar for January 1997

* **cancel** send/cancel requests to an LP print service
* **clear** clear the terminal screen
* **date** print and set the date

## useful control characters

### **^C terminate command**

### **^S suspend output**

### **^Q resume output**

# ***Communicating with People***

# ***Information on Others :***

## users Who else is logged on?

## who Information on current users

## ps What are people doing?

## 4. who am i display the effective current username

### **ps -el**

## w What are people doing?

**Pipes and Filter**

You can connect two commands together so that the output from one program becomes the input of the next program. Two or more commands connected in this way form a pipe.

To make a pipe, put a vertical bar (|) on the command line between two commands.

When a program takes its input from another program, performs some operation on that input, and writes the result to the standard output, it is referred to as a *filter*.

## The grep Command

The grep program searches a file or files for lines that have a certain pattern. The syntax is −

$grep pattern file(s)

The name "grep" derives from the ed (a UNIX line editor) command g/re/p which means "globally search for a regular expression and print all lines containing it."

A regular expression is either some plain text (a word, for example) and/or special characters used for pattern matching.

The simplest use of grep is to look for a pattern consisting of a single word. It can be used in a pipe so that only those lines of the input files containing a given string are sent to the standard output. If you don't give grep a filename to read, it reads its standard input; that's the way all filter programs work −

$ls -l | grep "Aug"

-rw-rw-rw- 1 john doc 11008 Aug 6 14:10 ch02

-rw-rw-rw- 1 john doc 8515 Aug 6 15:30 ch07

-rw-rw-r-- 1 john doc 2488 Aug 15 10:51 intro

-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros

$

There are various options which you can use along with grep command −

|  |  |
| --- | --- |
| **Option** | **Description** |
| **-v** | Print all lines that do not match pattern. |
| **-n** | Print the matched line and its line number. |
| **-l** | Print only the names of files with matching lines (letter "l") |
| **-c** | Print only the count of matching lines. |
| **-i** | Match either upper- or lowercase. |

Use a regular expression that tells grep to find lines with "carol", followed by zero or more other characters abbreviated in a regular expression as ".\*"), then followed by "Aug".

Here we are using *-i* option to have case insensitive search −

$ls -l | grep -i "carol.\*aug"

-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros

$

## The sort Command

The **sort** command arranges lines of text alphabetically or numerically. The example below sorts the lines in the food file −

$sort food

Afghani Cuisine

Bangkok Wok

Big Apple Deli

Isle of Java

Mandalay

Sushi and Sashimi

Sweet Tooth

Tio Pepe's Peppers

$

The **sort** command arranges lines of text alphabetically by default. There are many options that control the sorting −

|  |  |
| --- | --- |
| **Option** | **Description** |
| **-n** | Sort numerically (example: 10 will sort after 2), ignore blanks and tabs. |
| **-r** | Reverse the order of sort. |
| **-f** | Sort upper- and lowercase together. |
| **+x** | Ignore first x fields when sorting. |

More than two commands may be linked up into a pipe. Taking a previous pipe example using **grep**, we can further sort the files modified in August by order of size.

The following pipe consists of the commands **ls, grep,** and **sort** −

$ls -l | grep "Aug" | sort +4n

-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros

-rw-rw-r-- 1 john doc 2488 Aug 15 10:51 intro

-rw-rw-rw- 1 john doc 8515 Aug 6 15:30 ch07

-rw-rw-rw- 1 john doc 11008 Aug 6 14:10 ch02

$

This pipe sorts all files in your directory modified in August by order of size, and prints them to the terminal screen. The sort option +4n skips four fields (fields are separated by blanks) then sorts the lines in numeric order.

## The pg and more Commands

A long output would normally zip by you on the screen, but if you run text through more or pg as a filter, the display stops after each screenful of text.

Let's assume that you have a long directory listing. To make it easier to read the sorted listing, pipe the output through **more** as follows −

$ls -l | grep "Aug" | sort +4n | more

-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros

-rw-rw-r-- 1 john doc 2488 Aug 15 10:51 intro

-rw-rw-rw- 1 john doc 8515 Aug 6 15:30 ch07

-rw-rw-r-- 1 john doc 14827 Aug 9 12:40 ch03

.

.

.

-rw-rw-rw- 1 john doc 16867 Aug 6 15:56 ch05

--More--(74%)

The screen will fill up with one screenful of text consisting of lines sorted by order of file size. At the bottom of the screen is the **more** prompt where you can type a command to move through the sorted text.

# **Sed Command**

Sed is a Stream Editor used for modifying the files in unix (or linux). Whenever you want to make changes to the file automatically, sed comes in handy to do this. Most people never learn its power; they just simply use sed to replace text. You can do many things apart from replacing text with sed. Here I will describe the features of sed with examples.  
  
Consider the below text file as an input.

>cat file.txt

unix is great os. unix is opensource. unix is free os.

learn operating system.

unixlinux which one you choose.

Sed Command Examples

**1.** Replacing or substituting string  
  
Sed command is mostly used to replace the text in a file. The below simple sed command replaces the word "unix" with "linux" in the file.

>sed 's/unix/linux/' file.txt

linux is great os. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you choose.

Here the "s" specifies the substitution operation. The "/" are delimiters. The "unix" is the search pattern and the "linux" is the replacement string.  
  
By default, the sed command replaces the first occurrence of the pattern in each line and it won't replace the second, third...occurrence in the line.  
  
**2.** Replacing the nth occurrence of a pattern in a line.  
  
Use the /1, /2 etc flags to replace the first, second occurrence of a pattern in a line. The below command replaces the second occurrence of the word "unix" with "linux" in a line.  
  
>sed 's/unix/linux/2' file.txt

unix is great os. linux is opensource. unix is free os.

learn operating system.

unixlinux which one you choose.

**3.** Replacing all the occurrence of the pattern in a line.  
  
The substitute flag /g (global replacement) specifies the sed command to replace all the occurrences of the string in the line.

>sed 's/unix/linux/g' file.txt

linux is great os. linux is opensource. linux is free os.

learn operating system.

linuxlinux which one you choose.

**4.** Replacing from nth occurrence to all occurrences in a line.  
  
Use the combination of /1, /2 etc and /g to replace all the patterns from the nth occurrence of a pattern in a line. The following sed command replaces the third, fourth, fifth... "unix" word with "linux" word in a line.

>sed 's/unix/linux/3g' file.txt

unix is great os. unix is opensource. linux is free os.

learn operating system.

unixlinux which one you choose.

**5.** Changing the slash (/) delimiter  
  
You can use any delimiter other than the slash. As an example if you want to change the web url to another url as  
  
>sed 's/http:\/\//www/' file.txt

In this case the url consists the delimiter character which we used. In that case you have to escape the slash with backslash character, otherwise the substitution won't work.  
  
Using too many backslashes makes the sed command look awkward. In this case we can change the delimiter to another character as shown in the below example.

>sed 's\_http://\_www\_' file.txt

>sed 's|http://|www|' file.txt

**6.** Printing only the replaced lines  
  
Use the -n option along with the /p print flag to display only the replaced lines. Here the -n option suppresses the duplicate rows generated by the /p flag and prints the replaced lines only one time.

>sed -n 's/unix/linux/p' file.txt

linux is great os. unix is opensource. unix is free os.

linuxlinux which one you choose.

**WC Command**

The **wc** (**word count**) command in Unix/Linux operating systems is used to find out number of **newline count**, **word count**, **byte and characters** count in a files specified by the file arguments. The syntax of **wc** command as shown below.

**wc -l** : Prints the number of lines in a file.

**wc -w** : prints the number of words in a file.

**wc -c** : Displays the count of bytes in a file.

Cat tecmint.txt

Red Hat

CentOS

Fedora

Debian

Scientific Linux

OpenSuse

Ubuntu

Xubuntu

Linux Mint

Pearl Linux

Slackware

Mandriva

### **Example of WC Command**

The ‘**w** without passing any parameter will display a basic result of ”**tecmint.txt**‘ file. The three numbers elow are **12** (**number of lines**), **16** (**number of words**) and **112** (**number of bytes**) of the file.

$ wc tecmint.txt

**12** **16** **112** tecmint.txt

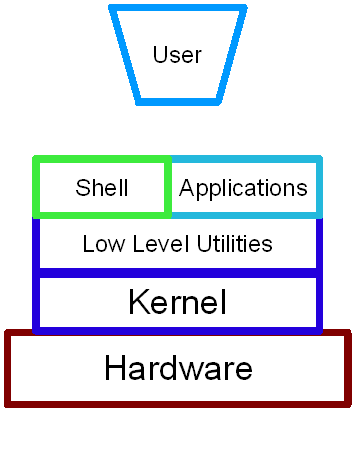
**Week 3**

**Shell**

The shell is the command prompt within Linux where we can type commands. If we have logged into a machine over a network (using ssh or telnet) then the commands you entered were run by the shell. If you are logged in using a graphical interface then you will may need to open a terminal client to see the shell. There are several different terminal clients available such as xterm, konsole and lxterm, or it may be just named Terminal Emulator.

Windows users may be familiar with the concept of a command prompt, or DOS prompt, which looks similar to a UNIX shell. The UNIX shell has more features and is practically an entire programming language, although don't let that put you off as you can use the shell without any programming ability.

To take a basic view of how Linux is built up see the diagram below:

  
The different layers of the Linux operating system

The kernel is the heart of the operating system. This is the bit that is actually *Linux*. The kernel is a process that runs continuously managing the computer.

The shell is an application that allows users to communicate with the computer. It is a text based application that allows programs to be started and tasks to be run. The shell is within a collections of utilities known as GNU. Without the kernel the computer cannot run and without the GNU utilities it can't do anything useful which is why the operating system is sometimes called GNU/Linux;

**Different types of Shells**

|  |  |  |
| --- | --- | --- |
| **Name of shell** | **Command name** | **Description** |
| Bourne shell | Sh | The most basic shell available on all UNIX systems |
| Korn Shell | ksh / pdksh | Based on the Bourne shell with enhancements |
| C Shell | Csh | Similar to the C programming language in syntax |
| Bash Shell | Bash | Bourne Again Shell combines the advantages of the Korn Shell and the C Shell. The default on most Linux distributions. |
| Tcsh | Tcsh | Similar to the C Shell |

When you login to a Linux machine (or open a shell window) you will normally be in the bash shell.

You can change shell temporarily by running the appropriate shell command. To change your shell for future logins then you can use the chsh command. This is normally setup to only allow you to change to one of the approved shells listed in the /etc/shells file. If you change your shell for future sessions this is stored in the /etc/passwd file.

The shell is more than just a way of typing commands. It can be used to stop, start, suspend programs and by writing script files it becomes a programming language in itself.

**The Shell Prompt**

When logged into the shell you will normal see one of the following prompts: $, % or #. This is an indication that the shell is waiting for an input from the user.

The Bourne, Korn, and Bash shells all use a similar syntax. If however you are in the C or tcsh shells this uses a completely different syntax and can require commands to be entered differently. To make it a little easier these have two different prompts depending upon the shell.

The default prompts are: $ - Bourne, Korn and Bash Shells  
% - C Shell

**The vi Editor**

## Vi standard UNIX "visual" editor

## Syntax: $ vi file ...

## Parameters:

### **if *file* doesn’t exist, vi will create it**

### **Derived from command-drive editors (ed and ex).**

### **Uses temporary storage called an editing buffer**

### **vi is always in one of two modes**

#### 1.Command Mode: Typed characters are interpreted as commands

#### 2.Input Mode:Typed characters are interpreted as characters to be entered into the buffer.

**Vi Commands**

**Esc** Go to command mode

**Vi Scope**

**^G** Show the line number on the mode line

**n key-stroke(s)** Repeat the key-stroke(s) n number of times

**u** Undo last change **U** Undoes all changes on the current line

**e** Forward to the end of the current word **b** Backward to the beginning of the current word

**$** Forward to the end of the line **0** Backward to the beginning of the line

**)** Forward to the beginning of the next sentence **(** Backward to the beginning of the current sentence

**}** Forward to the beginning of the next paragraph **{** Backward to the beginning of the current paragraph

**Moving the Cursor**

**k** Up one line **j** Down one line

**l** Forward a character **h** Back one character **b** Back one word

**o** Beginning of the line **$** End of the line

**^D** Half a page down **^U** Half a page up

**^F** Full page down **^B** Full page up

**G** End of buffer **nG** Go to line n **1G** Beginning of the buffer

**Entering Insert mode**

**a** Insert after cursor **A** Insert at the end of the line

**i** Insert before the cursor **o** Insert a blank line

**r** Replace a single character **R** Enter text in overwrite mode

**Killing, copying and pasting**

**x** Delete character after cursor **dd** Kill current line

**d scope** Kill the scope **j** Close up lines

**yy** Yank a line (copy) **y scope** Yank the scope

**p** Paste the kill buffer after the cursor **p** Paste the kill buffer before the cursor

**Rearranging Text :**

To Rearrange text using vi:– Yank or delete the text.

Move the cursor to where the text should go.

Place the yanked or deleted text there.

**Examples What it does**

**yw** Yanks a word **y$** Yanks to the end of the line

**yy or Y** Yanks the current line **j** Close up lines

**yy** Yank a line (copy) **3yw** Yanks three words

**3Y** Yanks three lines, starting with the current line

**Searching and Replacing**

**/textstring** Searches for textstring **/** Searches again for the same textstring

**?textstring** Searches backwards for textstring **n** Searches again (forwards or backwards)

**N** Reverses the direction of the search

**Saving and Exiting**

**:w** Save buffer to file **:w newfile** Writes buffer to newfile (save as)

**:e newfile** loads newfile into buffer **:q!** Quit vi and don’t save buffer

**:wq** Save buffer to file and quit vi

When we create a file, by default it is created with read and write permission turned on and execute permission turned off. A file can be made executable using chmod.

Step 1 : create the script using vi

For example, the script file show has the following lines

echo Here is the date and time

date

Step 2 : Make the file executable

* **chgrp** change the group ownership of a file
* **chmod** change the permissions mode of a file
* **chown** change owner of file

$ chmod u+x filename

$ chmod u+x show

Step 3 : To run the script, just type the filename

$ ./show

Here is the date and time

Sat jun 03 13:40:15 PST 2006

**How to run C programs**

Step 1 : Use an editor, such as vi, ex, or ed to write the program. The name of the file containing the program should end in .c.

For example, the file show.c contains the following lines :

main()

{ printf(“ welcome to VRSEC “); }

Step 2 : Submit the file to CC ( the C Compiler )

$ cc show.c

If the program is okay, the compiled version is placed in a file called a.out

Step 3 : To run the program, type a.out

$ ./a.out

Welcome to VRSEC

Checking whethter a given number is Armstrong number or Not.

#include<stdio.h>

int main()

{

    int num1, temp, sum = 0, rem;

    printf("\nEnter a Number:\t");

    scanf("%d", &num1);

    temp = num1;

    while(num1 != 0)

    {

        rem = num1%10;

        sum = sum + (rem\*rem\*rem);

        num1 = num1/10;

    }

    if(sum == temp)

    {

        printf("\n%d is an Armstrong Number\n", temp);

    }

    else

    {

        printf("\n%d is not an Armstrong Number\n", temp);

    }

    return 0;

}

2.Write a C Program to print Fibonacci series for a given number range.

#include <stdio.h>

int main()

{

int i, n, t1 = 0, t2 = 1, nextTerm = 0;

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

scanf("%d",&n);

// displays the first two terms which is always 0 and 1

printf("Fibonacci Series: %d, %d, ", t1, t2);

// i = 3 because the first two terms are already dislpayed

for (i=3; i <= n; ++i)

{

nextTerm = t1 + t2;

t1 = t2;

t2 = nextTerm;

printf("%d, ",nextTerm);

}

return 0;

}

**Decision Making in Shell Programs:**

**The if-then Statement** The Bourne shell also has an if-then construct. The syntax of the construct is as follows:

*if command\_1*

*then command\_2*

*command\_3*

*fi*

*command\_4*

**program - 1 :Shell program with error checking.**

# syntax – copy filename filename

if cp $1 $2

then

echo file copied successfully

fi

In the program 1, the echo command is executed only if the cp command is successful.

**The if-then-else Statement** executes one set of commands if a condition is true and a different set of commands if the condition is false.

*if command\_1*

*then*

*command\_2*

*command\_3*

*else*

*command\_4*

*command\_5*

*fi*

In this construct, *command\_1* is always executed. If *command\_1* succeeds, the *command\_2* and *command\_3* are executed; if it fails, *command\_4* and *command\_5* are executed.

**Program – 2 : Copying files**

if cp $1 $2

then

echo file copied successfully

else

echo failed to copy the file

fi

The **test** command examines some condition and returns a zero exit status if the condition is true and a nonzero exit status if the condition is false.

The conditions that can be tested fall into four categories: 1) String operators that test the condition or relationship of character strings; 2) Integer relationships that test the numerical relationship of two integers; 3) File operators that test for the existence or state of a file; 4) Logical operators that allow for and/or combinations of the other conditions.

**Testing Character Data**

|  |  |
| --- | --- |
| str1 = str2 | True if str1 is the same length and contains the same characters as str2 |
| str1 != str2 | True if str1 is not the same as str2 |
| -n str1 | True if the length of str1 is greater than 0 (is not null) |
| -z str1 | True if str1 is null (has a length of 0) |
| *str1* | True if *str1* is not null |

**Testing Numeric Data**

|  |  |
| --- | --- |
| *int1 -eq int2* | True if *int1* is numerically equal to *int2* |
| *int1 -ne int2* | True if *int1* is not equal to *int2* |
| *int1 -gt int2* | True if *int1* is greater than *int2* |
| *int1 -ge int2* | True if *int1* is greater than or equal to *int2* |
| *int1 -lt int2* | True if *int1* is less than *int2* |
| int1 -le int2 | True if int1 is less than or equal to int2 |

**Program – 3 :** Demonstrates the use of test command

# if basic is less than 1500, then HRA=10% DA=90%

# if his salary is either equal to or above 1500 then HRA = 500 DA 98%

# Find the gross salary of the employee

echo -e "Enter basic salary : \c"

read sal

if [ $sal -lt 1500 ]

then

hra = $sal \\* 10/100 | bc

da = $sal \\* 90/100 | bc

echo hra

echo da

fi

**Nested if Statements and the elif Construct**

You can do so by nesting if-else statements, as in the following syntax:

*if command*

*then*

*command*

*else*

*if command*

*then*

*command*

*else*

*if command*

*then*

*command*

*fi*

*fi*

*fi*

**Program – 4 on nested if-else**

echo Enter either 1 or 2

read i

if [ $i -eq 1 ]

then

echo You would go to heaven!

else

if [ $i -eq 2 ]

then

echo Hello was created with you in mind

else

echo How about mother earth!

fi

fi

**The case Statement** The case statement is cleaner because it does away with the elifs and the thens. It is more powerful because it allows pattern matching, much as the command-line interpreter does. The case statement allows a value to be named, which is almost always a variable, and a series of patterns to be used to match against the value, and a series of commands to executed if the value matches the pattern. The general syntax of case is as follows:

*case value in*

*pattern1)*

*command*

*command ;;*

*...*

*patternn)*

*command;*

*esac*

The case statement executes only one set of commands. If the value matches more than one of the patterns, only the first set of commands specified is executed. The double semicolons after a command act as the delimiter of the commands to be executed for a particular pattern match.

**Program – 5 prints the given digit in words.**

echo -e "Enter a number from 1 to 8 : \c"

read num

echo -e "\nEntered number is : \c"

case $num in

1) echo One ;;

2) echo Two ;;

3) echo Three ;;

4) echo Four ;;

5) echo Five ;;

6) echo Six ;;

7) echo Seven ;;

8) echo Eight ;;

esac

**Building Repetitions into a Program (Looping) :**

The Bourne shell has three different looping constructs built into the language.The three types of loops are the while loop, the until loop, and the for loop;

**while Loop** The while construct enables you to specify commands that will be executed while some condition is true.The general format:

*while command*

*do*

*command*

*command*

*...*

*command*

*done*

**Program - 6** prints the square of integers in succession

int=1

while [ $int -lt 5 ]

do

sq='expr $int \\* $int'

echo $sq

int='expr $int + 1'

done

echo "Job Complete"

**until Loop** The while construct causes the program to loop as long as some condition is true. The until construct is the complement to while; it causes the program to loop until a condition is true. The general format of the until construct is as follows:

*until command*

*do*

*command*

*command*

*...*

*command*

*done*

**Processing an Arbitrary Number of Parameters with shift** Before considering the for loop, it would be helpful to look at the shift command, since the for loop is really a shorthand use of shift.

In the examples presented so far, the number of positional parameters passed on to a command as a whole using the $\* variable. If a program needs to process each of the command-line arguments individually, and the number of arguments is not known, you can process the arguments one by one by using the shift command in your program. The shift command shifts the position of positional parameters by one; $2 becomes $1, $3 becomes $2, and so on. The parameter that was $1 before the shift command is not available after shift. The following simple program illustrates this concept:

**Program – 7** shifter

until [ $# -eq 0 ]

do

echo "Argument is $1 and 'expr $# - 1' argument(s) remain"

shift

done

**Program – 8** An integer summing program.

# sumints - a program to sum a series of integers

if [ $# -eq 0 ]

then

echo "Usage: sumints integer list"

exit 1

fi

sum=0

until [ $# -eq 0 ]

do

sum=´´expr $sum + $1´´

shift

done

echo $sum

**for Loop** for each word in the argument list it has been supplied. For each iteration of the loop, a variable name supplied on the for command line assumes the value of the next word in the argument list. The general syntax of the for loop is as follows:

*for variable in arg1 arg2 ... argn*

*do*

*command*

*...*

*command*

*done*

The following simple example illustrates the construct:

$ for LETTER in a b c d; do echo $LETTER; done

a

b

c

d

$

**Program - 9** Modified integer summing program.

if [ $# -eq 0 ]

then

echo "Usage: sumints integer list"

exit 1

fi

sum=0

for INT in $\*

do

sum=´´expr $sum + $INT´´

done

echo $sum

**Program** **11**:

AIM:**Write a shellscript to print the multiplication table for the given number.**

echo “enter the number”

read n

for i in 1 2 3 4 5 6 7 8 9 10

do

x= `expr $n \\* $i`

echo “ $n \* $i = $x”

done

Input:

enter the number6

Output:

6\*1=6

6\*2=12

6\*3=18

6\*4=24

6\*5=30

6\*6=36

6\*7=42

6\*8=48

6\*9=54

6\*10=60

**Program 12:**

AIM:**Write a shell script to perform simple calculator.**

echo ‘enter the value for a’

read a

echo ‘enter the value for b’

read b

echo ‘enter operator’

read op

case $op in

+) c =`expr $a + $b`;;

-) c = `expr $a -$b`;;

\\*) c = `expr $a \\* $b`;;

/) c = `expr $a / $b ;;

esac echo $c

Input:

enter the value for a

3

enter the value for b

6

enter the operator \* Output : 18

**Week 4**

**Program to implement Producer Consumer Problem using Semaphores**

**Description** : The problem illustrates that the producer produces the items which are consumed by the consumer and these two processes share a common pool consisting of n buffers , each capable of holding one item. The mutex semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1. The empty and full semaphores count the number of empty and full buffers, respectively. The producer produces full buffers to the consumer and the consumer produces empty buffers for the producer.

#include<conio.h>

void main()

{

int ch,n,c1=0,c2=0,produce[23],consume[23];

clrscr();

printf("\n\n\n\n\n\t\n\n\t\t\tEnter Stack Size : ",n);

scanf("%d",&n);

while(1)

{

clrscr();

printf("\t\tProducer Stack (Stack Size : %d

)\n\t\t~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~",n);

display(c1,produce);

printf("\n\n\t\tConsumer Stack (Stack Size : %d

)\n\t\t~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~",n);

display(c2,consume);

printf("\n\t\tCHOICES\n\t\t~~~~~~~\n\t1.Producer\n\t2.Consumer\n\t3.

Exit\nEnter your choice : ");

scanf("%d",&ch);

switch(ch)

{

case 1:

if(c1==n)

printf("Produer stack is FULL.So Producer goes to SLEEP\n");

else

{

c1++;

printf("\t\tEnter PRODUCE item is :");

scanf("%d",&produce[c1]);

}

break;

case 2:

if(c2==n)

printf("Consumer Stack is FULL.So it goes to SLEEP!..........\n\tReset the Cosumer

Stack\n",c2=0);

else if(c1==0)

printf("\tProducer stack is EMPTY\n");

Department of Information Technology, VR Siddhartha Engineering College, Vijayawada-7

28

else

{

c2++;

consume[c2]=produce[c1];

printf("\t\tCONSUME one item");

c1--;

}

break;

case 3:

exit(0);

default:

printf("\tIt is Wrong choice,Please enter correct

choice!............\n");

}

getch();

}

}

display(intc,int stack[])

{

int i;

printf("\n--------------------------------------------------------------------------

-----\n");

if(c==0)

printf("\tStack is EMPTY\n\t\t(Now It is sleeping)");

else

for(i=1;i<=c;i++)

printf("\t%d",stack[i]);

printf("\n--------------------------------------------------------------------------

-----\n");

}

**SAMPLE INPUT AND OUTPUT:**

CHOICES

~~~~~~~

1.Producer

2.Consumer

3.Exit

Enter your choice

Enter Stack Size : 4

Producer Stack (Stack Size : 4)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

-------------------------------------------------------------------------------

Stack is EMPTY

(Now It is sleeping)

------------------------------------------------------------------------- -----

Consumer Stack (Stack Size : 4)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

-------------------------------------------------------------------------------

Stack is EMPTY

(Now It is sleeping)

------------------------------------------------------------------------- -----

CHOICES

~~~~~~~

1.Producer

2.Consumer

3.Exit

Enter your choice : 2

Producer stack is EMPTY

Producer Stack (Stack Size : 4)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

-------------------------------------------------------------------------------

10

------------------------------------------------------------------------- -----

Consumer Stack (Stack Size : 4)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

-------------------------------------------------------------------------------

Stack is EMPTY

(Now It is sleeping)

------------------------------------------------------------------------- -----

CHOICES

~~~~~~~

1.Producer

2.Consumer

3.Exit

Enter your choice : 1

Enter PRODUCE item is :30

Producer Stack (Stack Size : 4)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

-------------------------------------------------------------------------------

10 30

------------------------------------------------------------------------- -----

Consumer Stack (Stack Size : 4)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

-------------------------------------------------------------------------------

Stack is EMPTY

(Now It is sleeping)

**Week 5 & 6**

**1. Simulate the following CPU scheduling algorithms**

**a) Round Robin**

**b) SJF**

**c) FCFS**

**d) Priority**

**Theory:**

Scheduling is a fundamental operating system function.

CPU scheduling is the basis of multi programming operating system. CPU scheduling algorithm determines how the CPU will be allocated to the process. These are of two types.

**1.** Primitive scheduling algorithms

**2.** Non-Primitive scheduling algorithms

1) **Primitive Scheduling algorithms**: In this, the CPU can release the process even in the middle of execution. For example: the cpu executes the process p1, in the middle of execution the cpu received a request signal from process p2, then the OS compares the priorities of p1&p2. If

the priority p1 is higher than the p2 then the cpu continue the execution of process p1. Otherwise the cpu preempt the process p1 and assigned to process p2.

2) **Non-Primitive Scheduling algorithm**: In this, once the cpu assigned to a process the

processor do not release until the completion of that process. The cou will assign to some other job only after the previous job has finished.

**Scheduling methodology:**

**Though put:** It means how many jobs are completed by the CPU with in a time period.

**Turn around time:** The time interval between the submission of the process and the time of the completion is the turn around time.

**Turn around time=Finished time – arrival time**

**Waiting time:** it is the sum of the periods spent waiting by a process in the ready queue

**Waiting time=Starting time- arrival time**

**Response time**: it is the time duration between the submission and first response

**Response time=First response-arrival time**

**CPU Utilization:** This is the percentage of time that the processor is busy. CPU utilization may range from 0 to 100%

**First-come, first-serve scheduling(FCFS):** In this, which process enter the ready queue first is served first. The OS maintains DS that is ready queue. It is the simplest CPU scheduling algorithm. If a process request the CPU then it is loaded into the ready queue, which process is the head of the ready queue, connect the CPU to that process.

**Shortest job First:** The criteria of this algorithm are which process having the smallest CPU burst, CPU is assigned to that next process. If two process having the same CPU burst time FCFS is used to break the tie.

**Priority Scheduling**: These are of two types. One is internal priority, second is external priority. The cpu is allocated to the process with the highest priority. Equal priority processes are scheduled in the FCFS order. Priorities are generally some fixed range of numbers such as 0 to 409. The low numbers represent high priority

**Round Robin**: It is a primitive scheduling algorithm it is designed especially for time sharing systems. In this, the CPU switches between the processes. When the time quantum expired, the CPU switches to another job. A small unit of time called a quantum or time slice. A time quantum is generally is a circular queue new processes are added to the tail of the ready queue.

If the process may have a CPU burst of less than one time slice then the process release the CPU voluntarily. The scheduler will then process to next process ready queue otherwise; the process will be put at the tail of the ready queue.

FCFS

#include<stdio.h>

#include<conio.h>

#define max 100

int main()

{

int ct[max],bt[max],wt[max],tat[max],a[max],s,i;

float awt=0.0;

float ata=0.0;

char p[max];

clrscr();

printf("\n Enter the number of process: ");

scanf("%d",&s);

printf("\n Enter the arrival times in order from 0: ");

for(i=0;i<s;i++)

scanf("%d",&a[i]);

printf("\n Enter burst times: ");

for(i=0;i<s;i++)

scanf("%d",&bt[i]);

for(i=0;i<s;i++){

if(i==0)

ct[i]=bt[i];

else

ct[i]=ct[i-1]+bt[i];

tat[i]=ct[i]-a[i];

wt[i]=tat[i]-bt[i];

awt+=wt[i];

ata+=tat[i];

}

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

for(i=0;i<s;i++)

printf(" P%d ||",i);

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("0 ");

for(i=0;i<s;i++)

printf(" %d",ct[i]);

printf("\n Process AT BT CT TAT WT ");

for(i=0;i<s;i++)

printf("\n P%d \t%d %d %d %d %d",i, a[i],bt[i],ct[i],tat[i],wt[i]);

awt=awt/s;

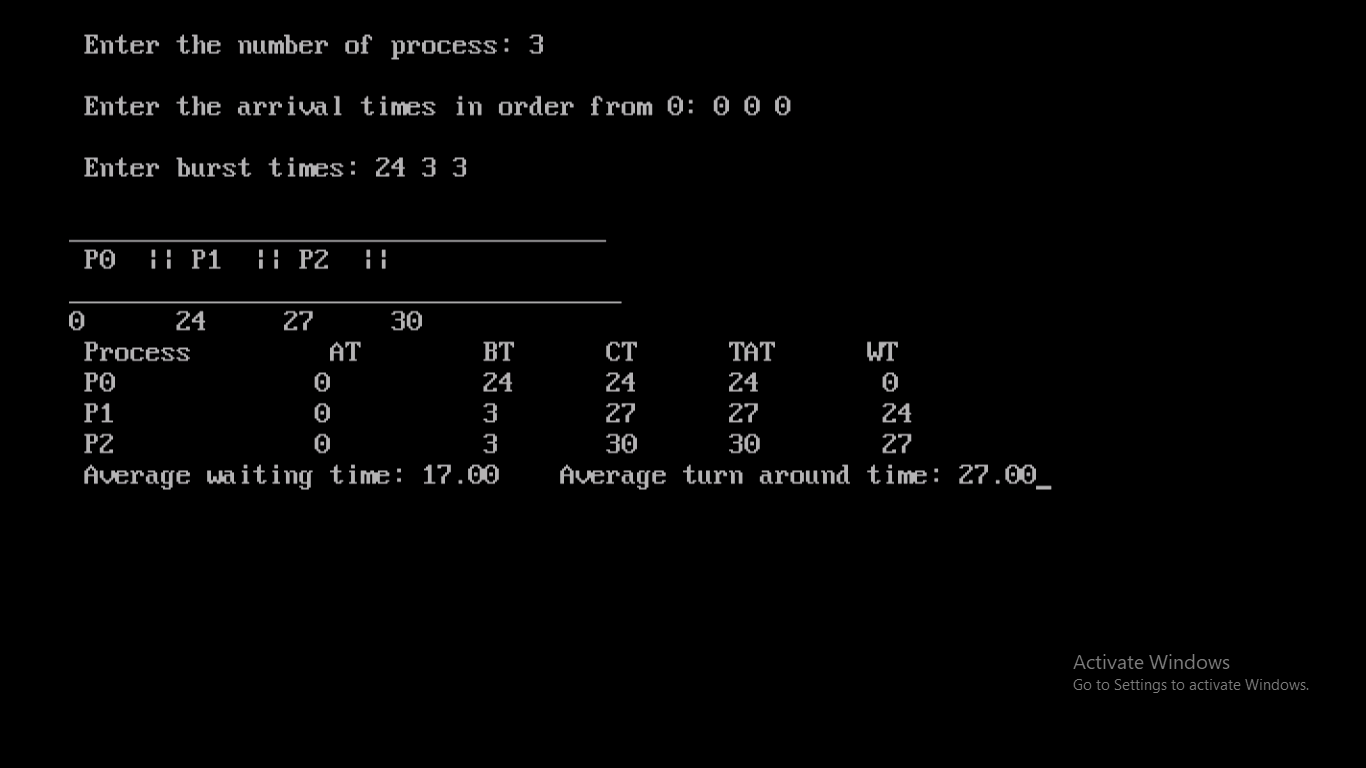
ata=ata/s;

printf("\n Average waiting time: %.2f\tAverage turn around time: %.2f",awt,ata);

getch();

return 0;

}



SJF

#include<stdio.h>

#include<conio.h>

#define pf printf

struct Process

{

int pno,at,bt,ct,tat,wt;

struct Process \*np;

};

typedef struct Process P;

P\* Crt\_New\_Process();

P\* Add\_New\_Process(P\*);

void Clac\_All\_Times(P\*);

void Display\_Table(P\*);

void Display\_Gantt\_Chart(P\*);

int i;

float awt=0,atat=0;

int main()

{

P \*p1=NULL;

char ch;

i=0;

clrscr();

pf("\n\n\n\n\t\t Enter all details with arrival times in order. ");

getch();

do

{

clrscr();

pf("\n\a Enter the details for Process No: %d",i+1);

if(i==0)

p1= Crt\_New\_Process();

else

p1= Add\_New\_Process(p1);

pf("\n\n\n Anymore Processes??");

ch=getche();

i++;

}while(ch=='Y'||ch=='y');

Clac\_All\_Times(p1);

Display\_Table(p1);

Display\_Gantt\_Chart(p1);

getch();

free(p1);

return 0;

}

P\* Crt\_New\_Process()

{

P\* nn;

nn=(P\*)malloc(sizeof(P));

(nn->pno)=(i+1);

pf("\n\n\n\t AT: "); scanf("%d",&(nn->at));

pf("\n\t BT: "); scanf("%d",&(nn->bt));

nn->np=NULL;

return nn;

}

P\* Add\_New\_Process(P\* p1)

{

P \*nn=Crt\_New\_Process(),\*p=(P\*)malloc(sizeof(P));

p=p1;

if((nn->bt)<(p->bt))

{

nn->np=p1;

p1=nn;

}

else

{

while((nn->bt)>((p->np)->bt))

{

if((p->np)!=NULL)

p=p->np;

else

break;

}

(nn->np)=(p->np);

p->np=nn;

}

return p1;

}

void Clac\_All\_Times(P\* p1)

{

P \*p = (P\*)malloc(sizeof(P)),\*n = (P\*)malloc(sizeof(P));

p=p1;

while(p!=NULL)

{

if(p==p1)

(p->ct) = (p->bt);

else

(p->ct) = (n->ct) + (p->bt);

(p->tat) = (p->ct) - (p->at);

(p->wt) = (p->tat) - (p->bt);

awt += p->wt;

atat += p->tat;

n=p;

p=p->np;

}

n=p;

awt/=i;

atat/=i;

free(p);

free(n);

}

void Display\_Table(P\* p1)

{

P \*p;

clrscr();

p=(P\*)malloc(sizeof(P));

p=p1;

pf("\n\n\t\tPROCESS\_NO\tAT\tBT\tCT\tTAT\tWT\n\n");

while(p!=NULL)

{

pf("\n\t\t %d\t\t%d\t%d\t%d\t%d\t%d",p->pno,p->at,p->bt,p->ct,p->tat,p->wt);

p=p->np;

}

pf("\n\n\n\n\t\t\tAWT: %.2f\t\tATAT: %.2f",awt,atat);

free(p);

}

void Display\_Gantt\_Chart(P\* p1)

{

P\* p=(P\*)malloc(sizeof(P));

p=p1;

pf("\n\n\n\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\t");

while(p!=NULL)

{

pf("| P%d ",p->pno);

p=p->np;

}

pf("|\n\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

p=p1;

pf("\t0");

while(p!=NULL)

{

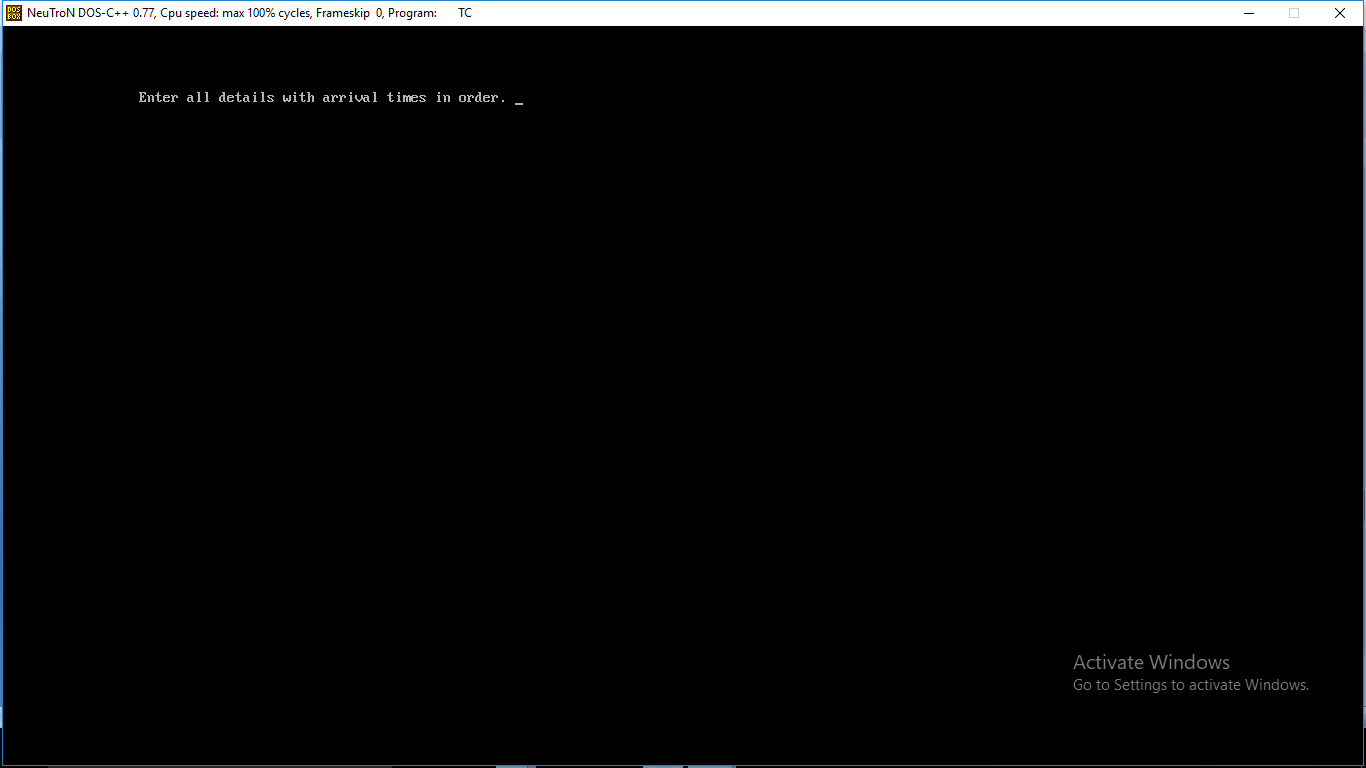
pf("%6d",p->ct);

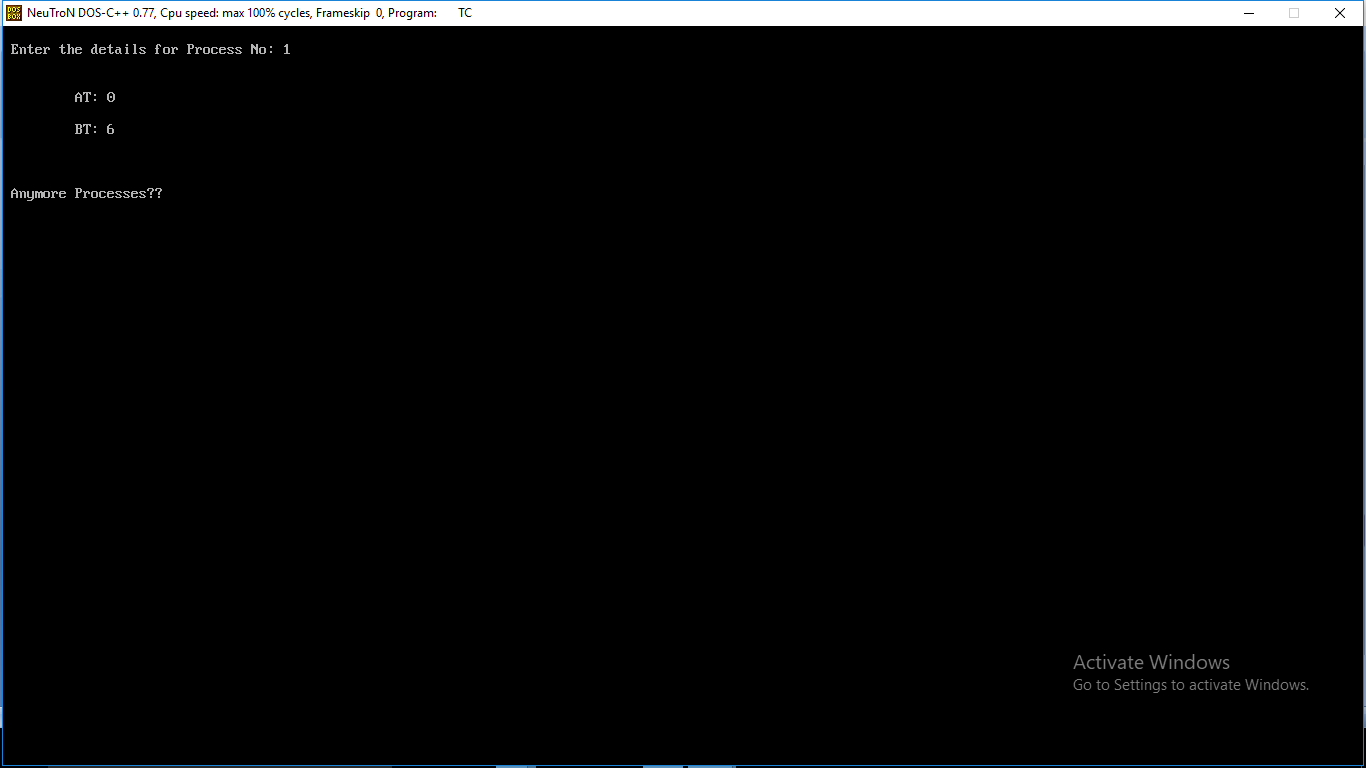
p=p->np;

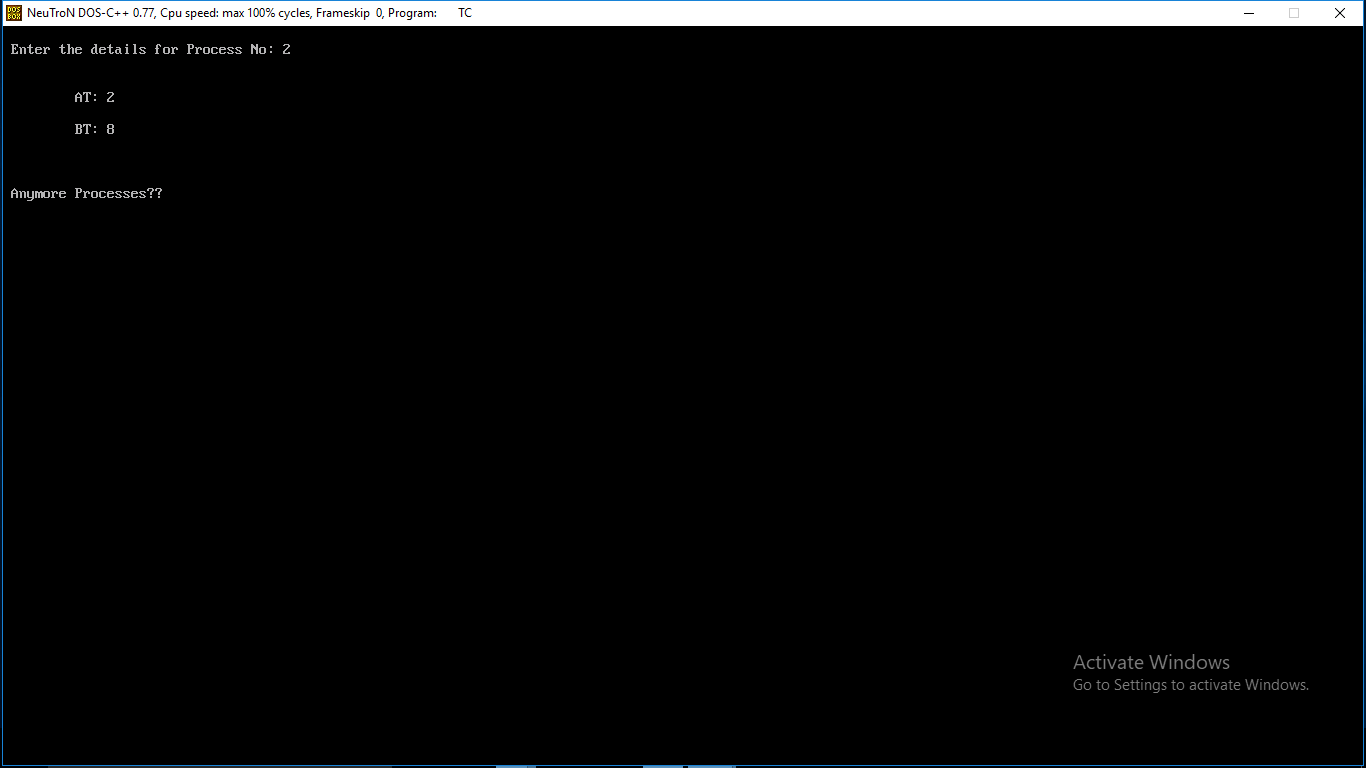
}

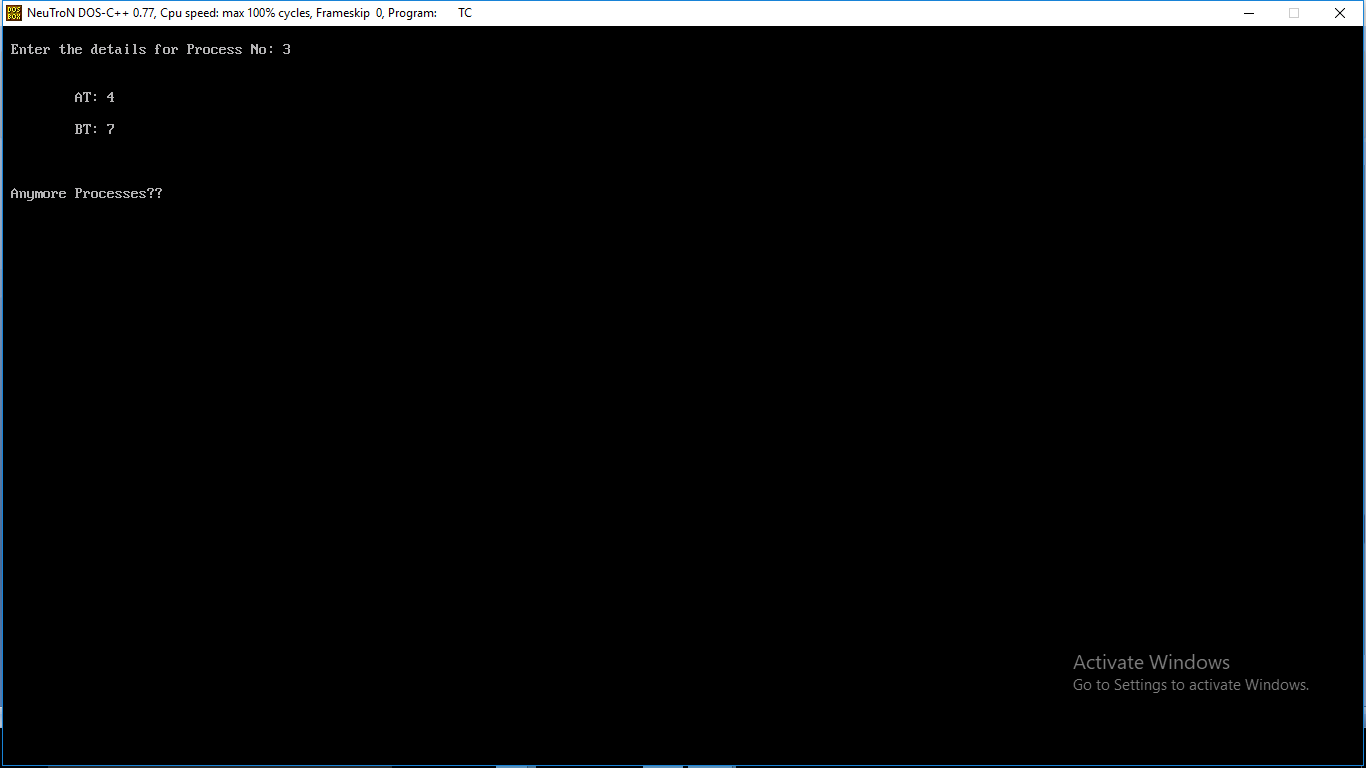
free(p);

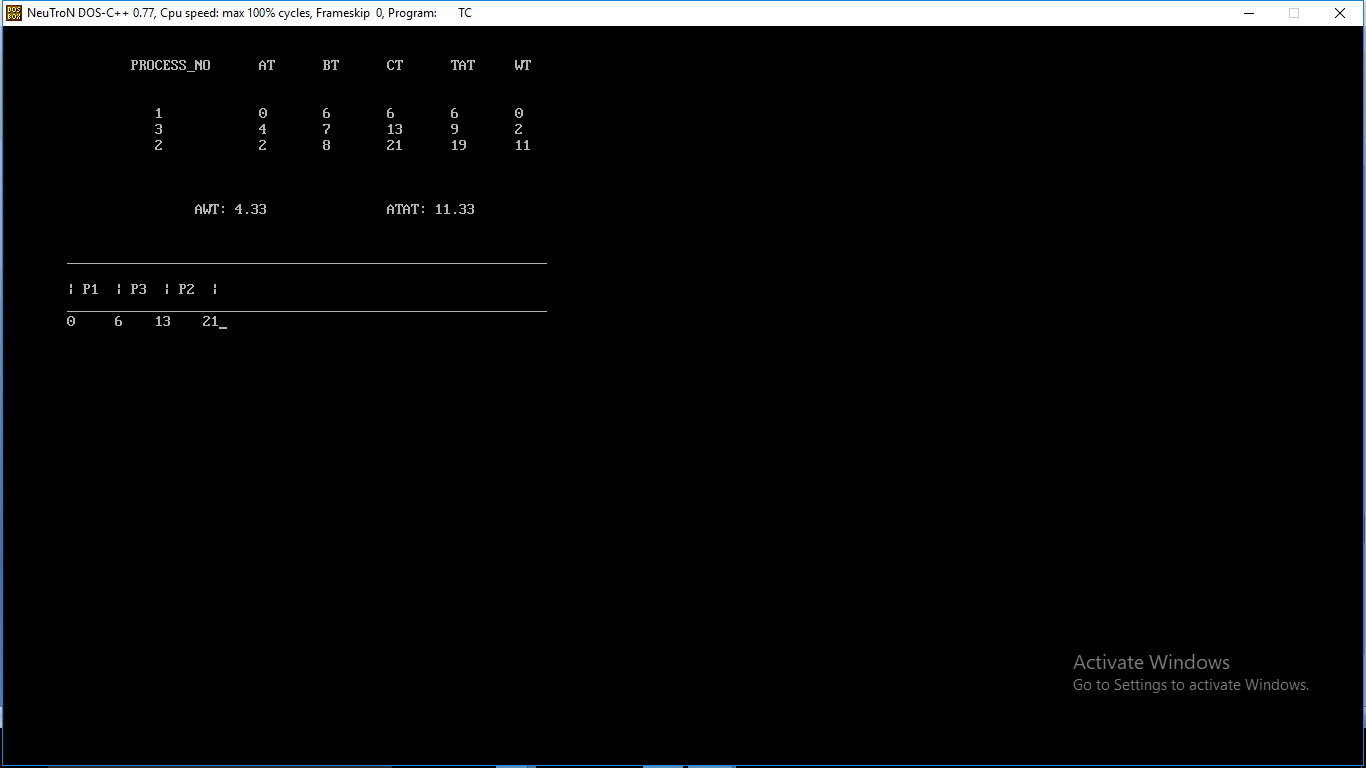
}











PRIORITY

#include<stdio.h>

#include<conio.h>

#define pf printf

struct Process

{

int pno,at,bt,ct,tat,wt,pr;

struct Process \*np;

};

typedef struct Process P;

P\* Crt\_New\_Process();

P\* Add\_New\_Process(P\*);

void Clac\_All\_Times(P\*);

void Display\_Table(P\*);

void Display\_Gantt\_Chart(P\*);

int i;

float awt=0,atat=0;

int main()

{

P \*p1=NULL;

char ch;

i=0;

clrscr();

pf("\n\n\n\n\t\t Enter all details with arrival times in order. ");

getch();

do

{

clrscr();

pf("\n\a Enter the details for Process No: %d",i+1);

if(i==0)

p1= Crt\_New\_Process();

else

p1= Add\_New\_Process(p1);

pf("\n\n\n Anymore Processes??");

ch=getche();

i++;

}while(ch=='Y'||ch=='y');

Clac\_All\_Times(p1);

Display\_Table(p1);

Display\_Gantt\_Chart(p1);

getch();

free(p1);

return 0;

}

P\* Crt\_New\_Process()

{

P\* nn;

nn=(P\*)malloc(sizeof(P));

(nn->pno)=(i+1);

pf("\n\n\n\t AT: "); scanf("%d",&(nn->at));

pf("\n\t BT: "); scanf("%d",&(nn->bt));

pf("\n\t Priority: "); scanf("%d",&(nn->pr));

nn->np=NULL;

return nn;

}

P\* Add\_New\_Process(P\* p1)

{

P \*nn=Crt\_New\_Process(),\*p=(P\*)malloc(sizeof(P));

p=p1;

if((nn->pr)<(p->pr))

{

nn->np=p1;

p1=nn;

}

else

{

while((nn->pr)>((p->np)->pr))

{

if((p->np)!=NULL)

p=p->np;

else

break;

}

(nn->np)=(p->np);

p->np=nn;

}

return p1;

}

void Clac\_All\_Times(P\* p1)

{

P \*p = (P\*)malloc(sizeof(P)),\*n = (P\*)malloc(sizeof(P));

p=p1;

while(p!=NULL)

{

if(p==p1)

(p->ct) = (p->bt);

else

(p->ct) = (n->ct) + (p->bt);

(p->tat) = (p->ct) - (p->at);

(p->wt) = (p->tat) - (p->bt);

awt += p->wt;

atat += p->tat;

n=p;

p=p->np;

}

n=p;

awt/=i;

atat/=i;

free(p);

free(n);

}

void Display\_Table(P\* p1)

{

P \*p;

clrscr();

p=(P\*)malloc(sizeof(P));

p=p1;

pf("\n\n\tPRIORITY PROCESS\_NO \tAT\tBT\tCT\tTAT\tWT\n\n");

while(p!=NULL)

{

pf("\n\t %d\t\t%d\t\t%d\t%d\t%d\t%d\t%d",p->pr,p->pno,p->at,p->bt,p->ct,p->tat,p->wt);

p=p->np;

}

pf("\n\n\n\n\t\t\tAWT: %.2f\t\tATAT: %.2f",awt,atat);

free(p);

}

void Display\_Gantt\_Chart(P\* p1)

{

P\* p=(P\*)malloc(sizeof(P));

p=p1;

pf("\n\n\n\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\t");

while(p!=NULL)

{

pf("| P%d ",p->pno);

p=p->np;

}

pf("|\n\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

p=p1;

pf("\t0");

while(p!=NULL)

{

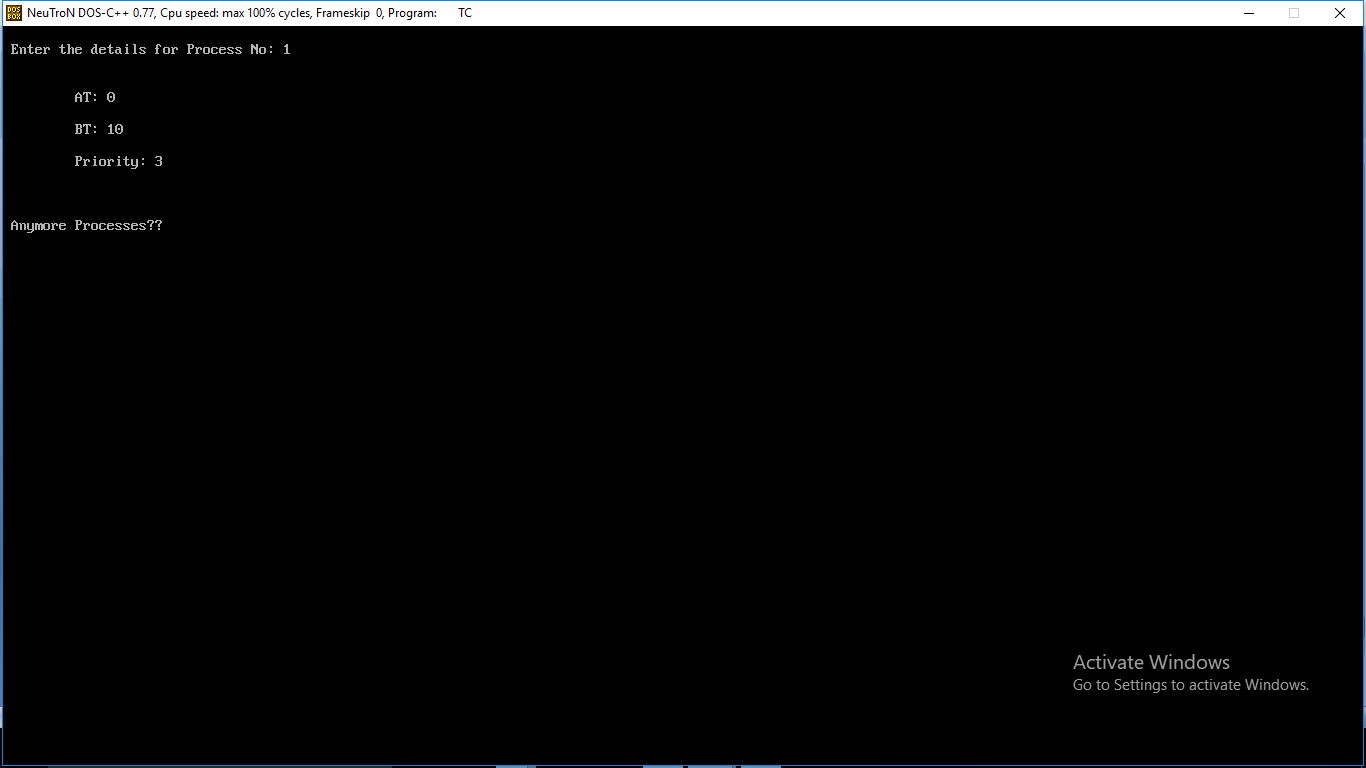
pf("%6d",p->ct);

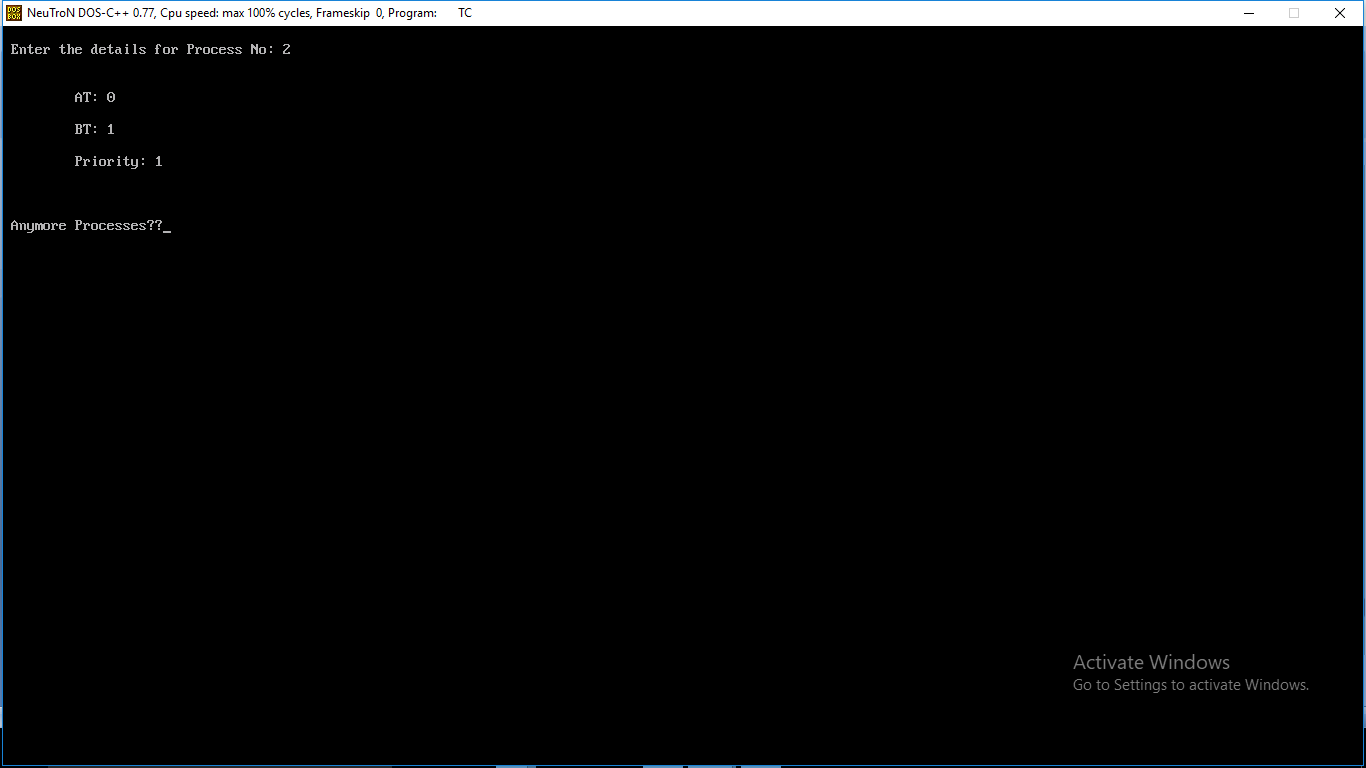
p=p->np;

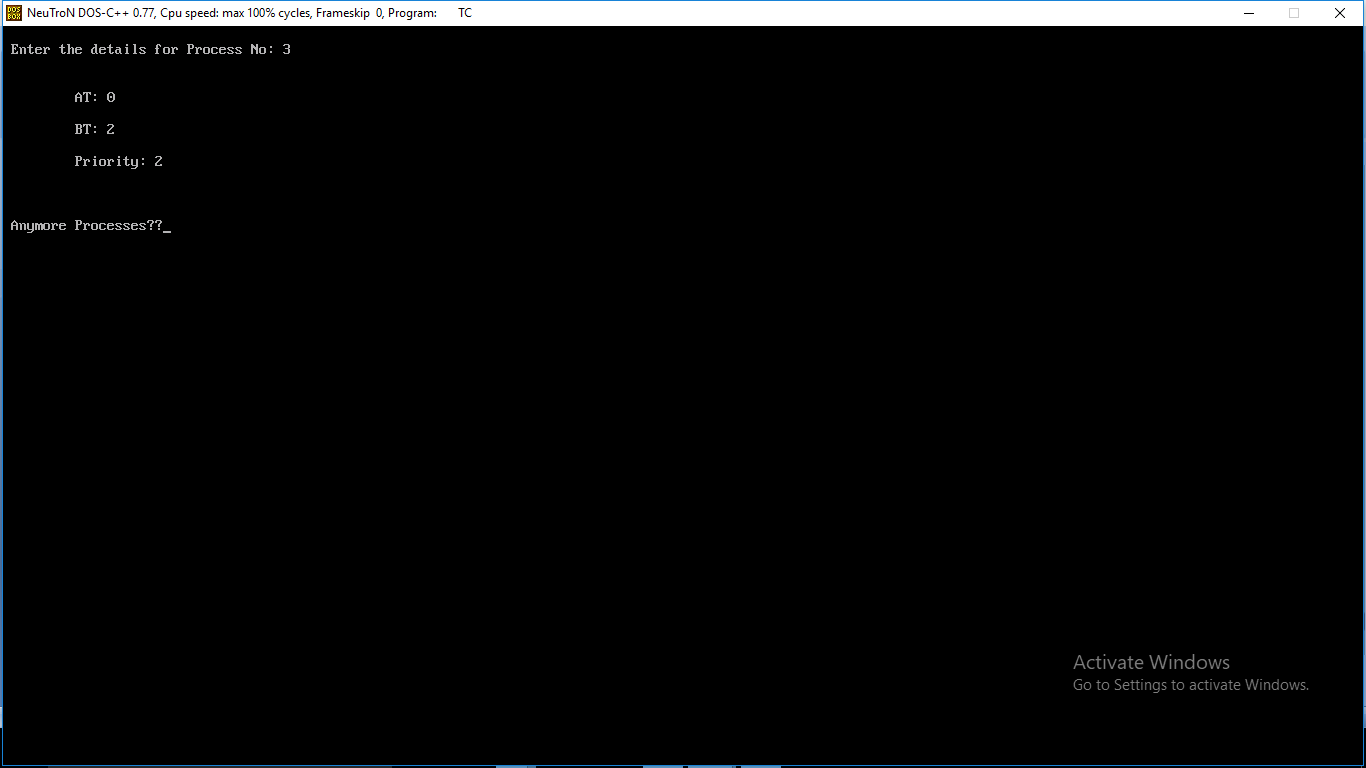
}

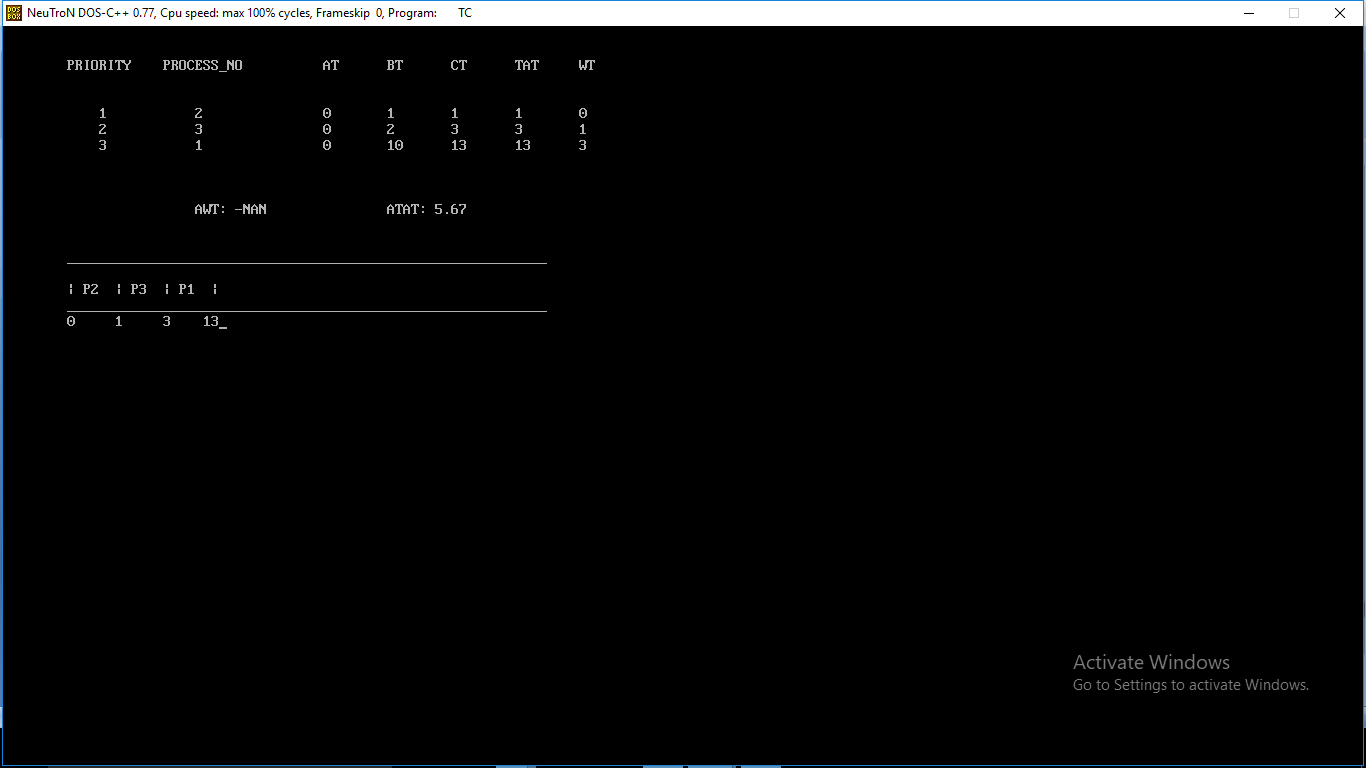
free(p);

}









Rr

#include<stdio.h>

#include<conio.h>

#define pf printf

struct Process

{

int pno,at,bt,ct,tat,wt;

struct Process \*np;

};

typedef struct Process P;

P\* Crt\_New\_Process();

P\* Add\_New\_Process(P\*);

P\* Clac\_All\_Times(P\*);

void Display\_Table(P\*);

void Display\_Gantt\_Chart();

int i,t;

int pn[100]={0},tv[100]={0},a=0;

float awt=0,atat=0;

int main()

{

P \*p1=NULL;

char ch;

i=0;

clrscr();

pf("\n\n\n\n\t\t Enter all details with arrival times in order. ");

getch();

pf("\n\n Enter the TIME QUNATUM: ");

scanf("%d",&t);

do

{

clrscr();

pf("\n\a Enter the details for Process No: %d",i+1);

if(i==0)

p1= Crt\_New\_Process();

else

p1= Add\_New\_Process(p1);

pf("\n\n\n Anymore Processes??");

ch=getche();

i++;

}while(ch=='Y'||ch=='y');

p1=Clac\_All\_Times(p1);

Display\_Table(p1);

Display\_Gantt\_Chart();

getch();

free(p1);

return 0;

}

P\* Crt\_New\_Process()

{

P\* nn;

nn=(P\*)malloc(sizeof(P));

(nn->pno)=(i+1);

pf("\n\n\n\t AT: "); scanf("%d",&(nn->at));

pf("\n\t BT: "); scanf("%d",&(nn->bt));

nn->np=nn;

return nn;

}

P\* Add\_New\_Process(P\* p1)

{

P \*nn=Crt\_New\_Process(),\*p=(P\*)malloc(sizeof(P));

p=p1;

while((p->np)!=p1)

p=p->np;

p->np=nn;

nn->np=p1;

return p1;

}

P\* Clac\_All\_Times(P\* p1)

{

P \*p = (P\*)malloc(sizeof(P)),\*n = (P\*)malloc(sizeof(P));

int \*b= (int\*)malloc(i\*sizeof(int)),j=0,k=0,f=0;

p=p1;

do

{

\*(b+j)=p->bt;

p=p->np;

j++;

}while(p!=p1);

p=p1;

p->ct=0;

if(\*(b+k)>t)

{(p->ct) = t;

\*(b+k) -= t;}

else

{(p->ct) = \*(b+k);

\*(b+k) = 0;}

pn[a]=(p->pno);

tv[a]=(p->ct);

n=p;

p=p->np;

do

{

k++;

if(\*(b+k))

{

if(\*(b+k)>t)

{

(p->ct) = t + (n->ct);

\*(b+k) -= t;

}

else

{

(p->ct) = \*(b+k) + (n->ct);

\*(b+k) = 0;

}

a++;

pn[a] = p->pno;

tv[a] = p->ct;

n=p;

}

p=p->np;

if(p==p1)

k=-1;

for(j=0;(!\*(b+j))&&(j<=i);j++)

if(j==i)

f=-2;

}while(f!=-2);

p=p1;

do

{

p->tat = (p->ct) - (p->at);

p->wt = (p->tat) - (p->bt);

awt+=(p->wt);

atat+=(p->tat);

p=p->np;

}while(p!=p1);

awt/=i;

atat/=i;

free(b);

return p1;

}

void Display\_Table(P\* p1)

{

P \*p;

clrscr();

p=(P\*)malloc(sizeof(P));

p=p1;

pf("\n\n\t\tPROCESS\_NO\tAT\tBT\tCT\tTAT\tWT\n\n");

do

{

pf("\n\t\t %d\t\t%d\t%d\t%d\t%d\t%d",p->pno,p->at,p->bt,p->ct,p->tat,p->wt);

p=p->np;

}while(p!=p1);

pf("\n\n\n\n\t\t\t\tTIME QUANTUM: %d",t);

pf("\n\n\t\t\tAWT: %.2f\t\tATAT: %.2f",awt,atat);

free(p);

}

void Display\_Gantt\_Chart()

{

int j=0;

pf("\n\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");

while(j<=a)

{

pf("|P%d ",pn[j]);

j++;

}

pf("|\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

j=0;

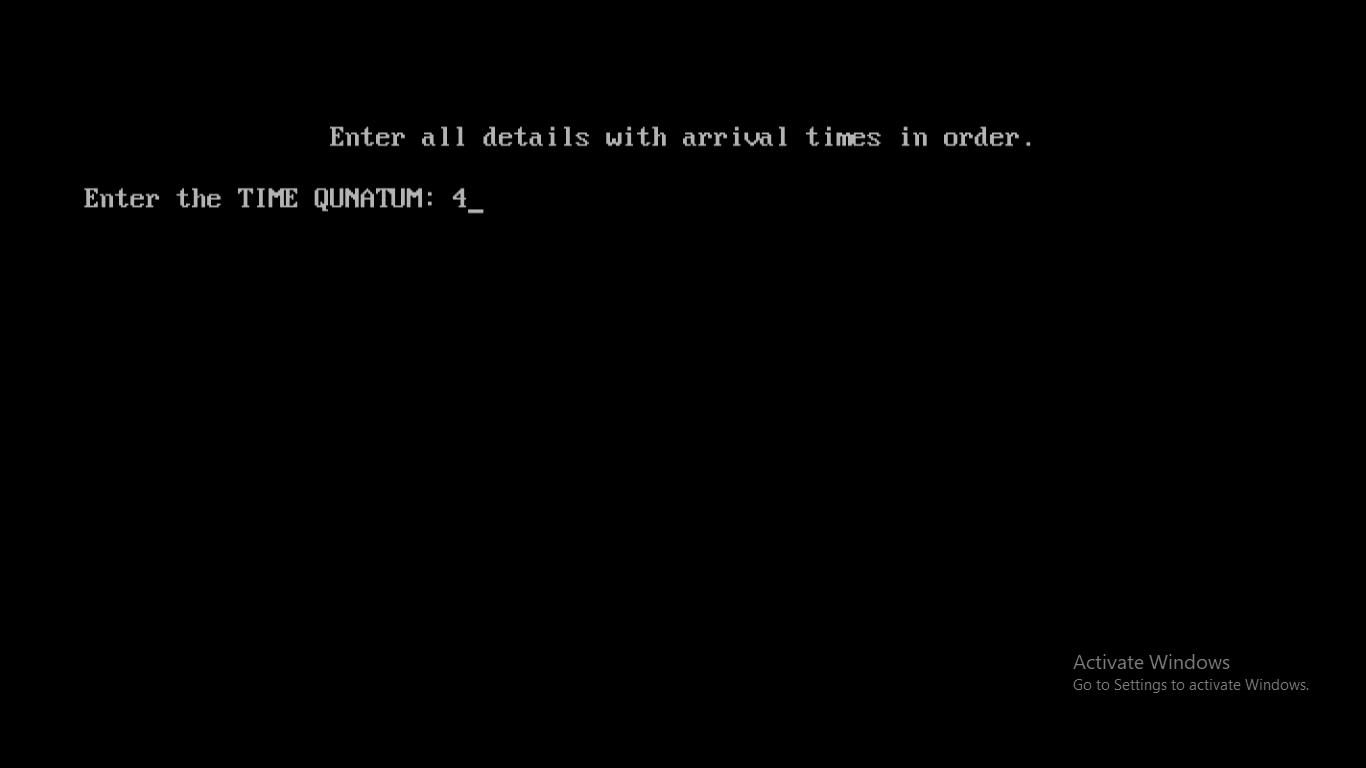
pf("0");

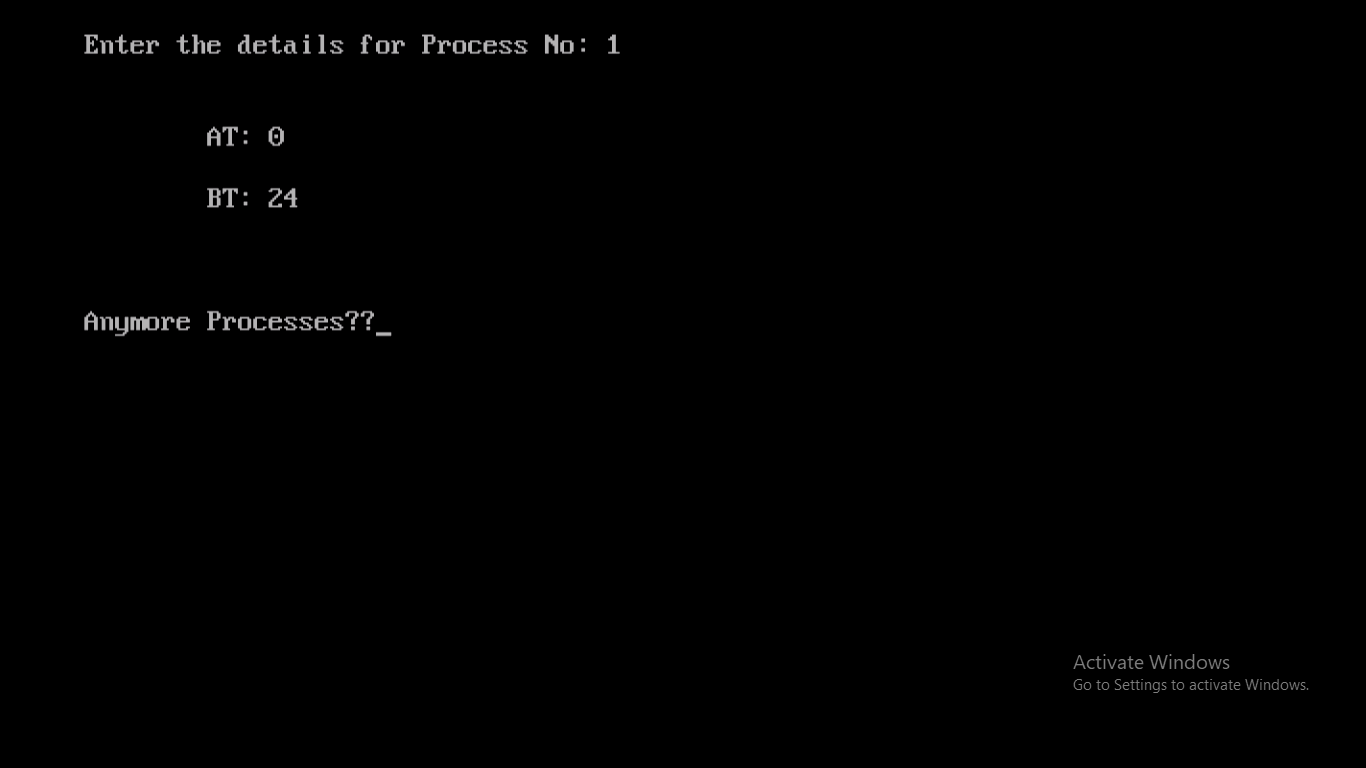
while(j<=a)

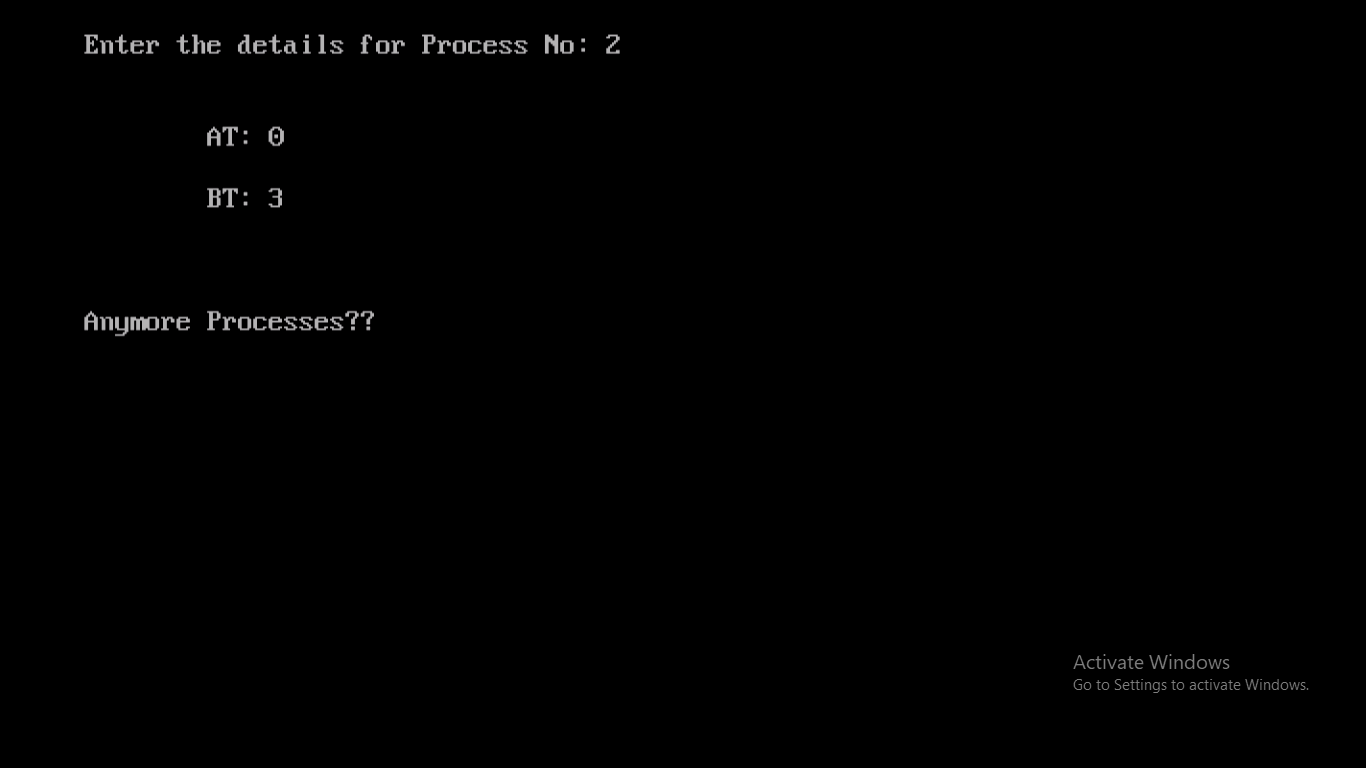
{

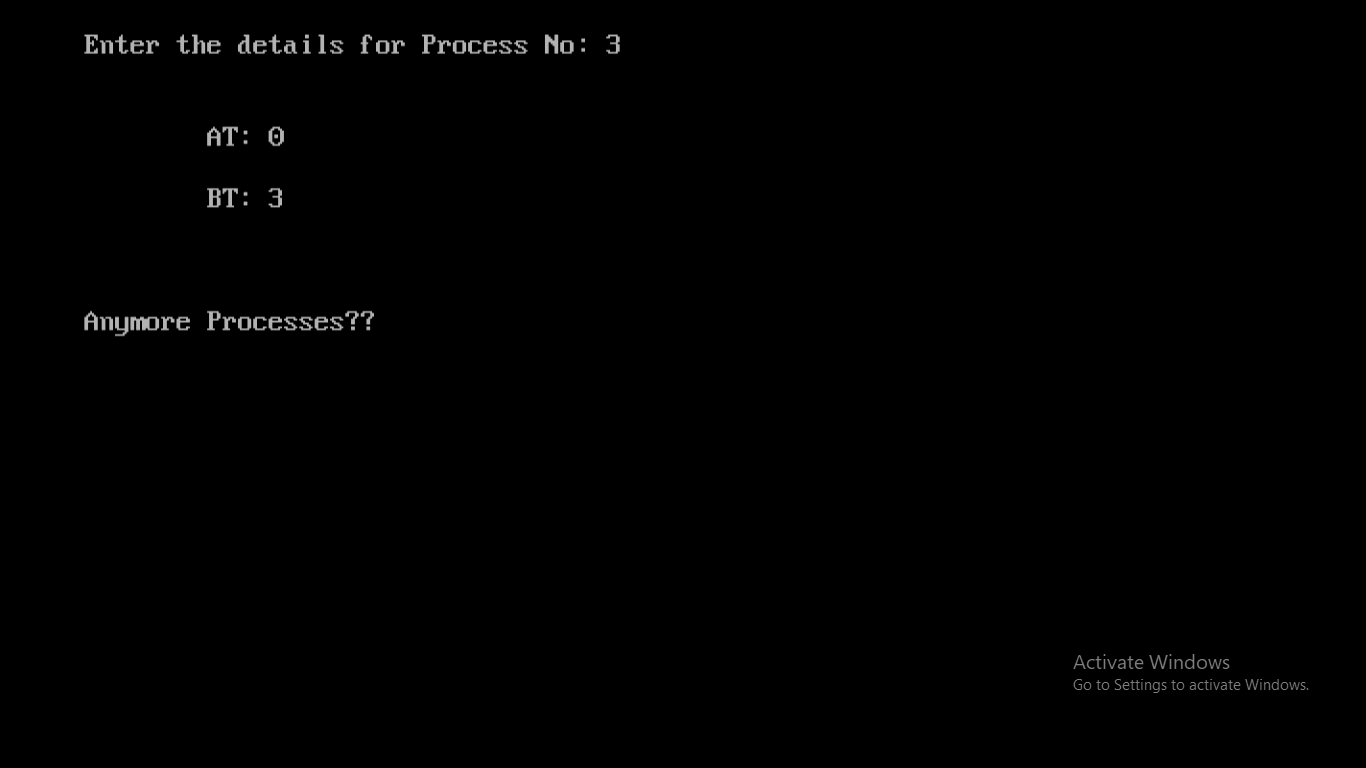
pf("%4d",tv[j]);

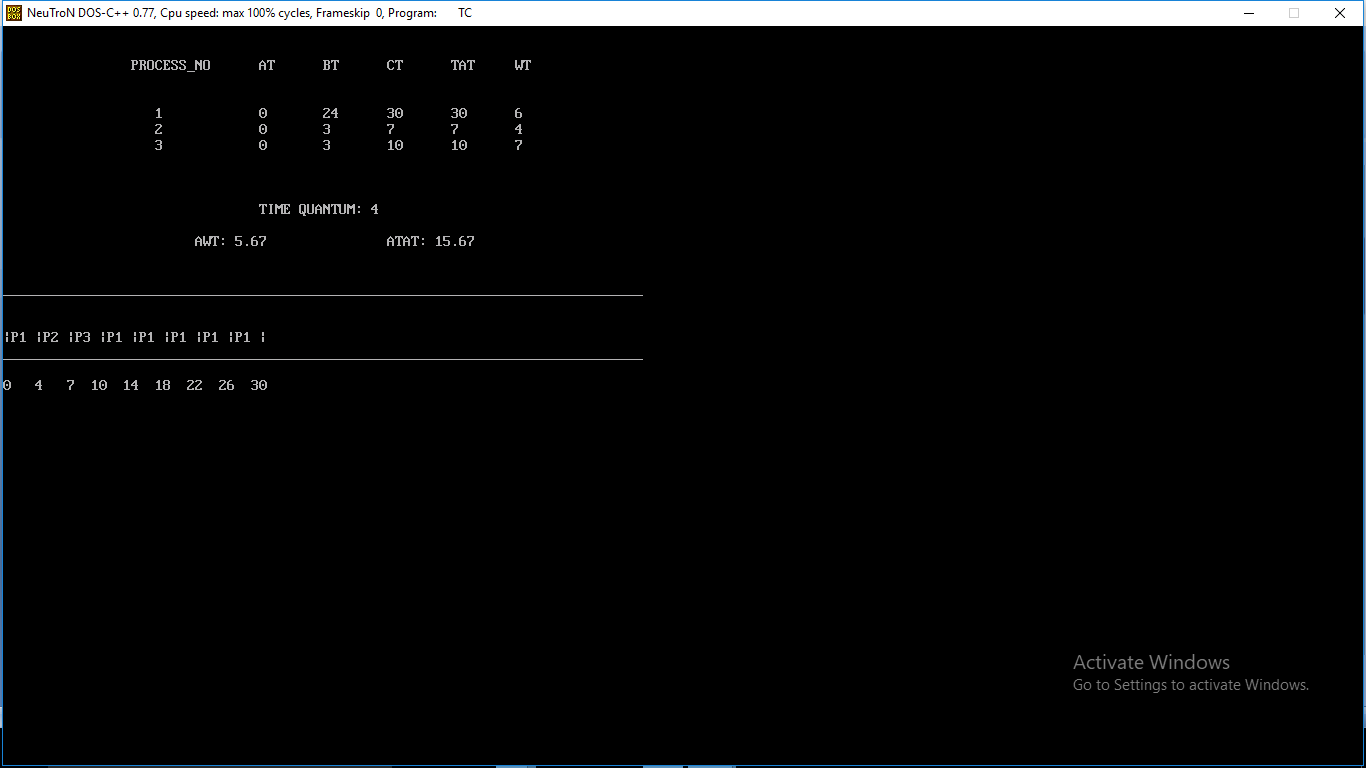
j++; } }











**Week 7**

**PROGRAM TO IMPLEMENT BANKER'S ALGORITHM**

**Simulate Bankers Algorithm for Deadlock Avoidance. Theory:**

**Deadlock:** A process request the resources, the resources are not available at that time, so the process enter into the waiting state. The requesting resources are held by another waiting process, both are in waiting state, this situation is said to be Deadlock.

A deadlocked system must satisfied the following 4 conditions. These are:

**(i) Mutual Exclusion:** Mutual Exclusion means resources are in non-sharable mode only, it means only one process at a time can use a process.

**(ii) Hold and Wait:** Each and every process is the deadlock state, must holding at least one resource and is waiting for additional resources, that are currently being held by another process.

P P P

**(iii) No Preemption:** No Preemption means resources are not released in the middle of the work, they released only after the process has completed its task.

**(iv) Circular Wait:** If process P1 is waiting for a resource R1, it is held by P2, process P2 is waiting for

R2, R2 held by P3, P3 is waiting for R4, R4 is held by P2, P2 waiting for resource R3, it is held by P1.

P1 R1 P2 R2 P3 R4 P2 R3

**Deadlock Avoidance:** It is one of the method of dynamically escaping from the deadlocks. In this scheme, if a process request for resources, the avoidance algorithm checks before the allocation of resources about the state of system. If the state is safe, the system allocate the resources to the requesting process otherwise (unsafe) do not allocate the resources. So taking care before the allocation said to be deadlock avoidance.

**Banker’s Algorithm:** It is the deadlock avoidance algorithm, the name was chosen because the bank never allocates more than the available cash.

**Available:** A vector of length ‘m’ indicates the number of available resources of each type. If available[j]=k, there are ‘k’ instances of resource types Rj available.

**Allocation:** An nxm matrix defines the number of resources of each type currently allocated to each process. If allocation[i,j]=k, then process Pi is currently allocated ‘k’ instances of resources type Rj. **Max:** An nxm matrix defines the maximum demand of each process. If max[i,j]=k, then Pi may request at most ‘k’ instances of resource type Rj.

**Need:** An nxm matrix indicates the remaining resources need of each process. If need[I,j]=k, then Pi may need ‘k’ more instances of resource type Rj to complete this task. There fore, Need[i,j]=Max[i,j]- Allocation[I,j]

BANKERS ALGORITHM

#include<stdio.h>

#include<conio.h>

#define pf printf

struct avail

{

int a[100];

};

struct resource

{

int inst;

int ava;

};

struct res

{

int max;

int alcd;

int need;

};

struct process

{

struct res r[100];

int state;

};

typedef struct resource R;

typedef struct process P;

typedef struct res RA;

typedef struct avail A1;

int main()

{

R r1[100];

P p1[100];

A1 av;

int n\_r,n\_p,i,j,f=1,x;

clrscr();

pf("\n Enter the number of resources: ");

scanf("%d",&n\_r);

for(i=1;i<=n\_r;i++)

{

pf("\n Enter the number of instances for Resource %c: ",i+64);

scanf("%d",&(r1[i].inst));

av.a[i]=r1[i].inst;

}

pf("\n Enter the number of Processes: ");

scanf("%d",&n\_p);

for(i=1;i<=n\_p;i++)

{

clrscr();

p1[i].state=1;

pf("\n Enter Maximum number of Resources for Process %d: \n\n",i);

for(j=1;j<=n\_r;j++)

{

pf("\n Resource %c: ",j+64);

flushall();

scanf("%d",&(p1[i].r[j].max));

}

pf("\n Enter number of Resources Allocated for Process %d: \n\n",i);

for(j=1;j<=n\_r;j++)

{

flushall();

pf("\n Resource %c: ",j+64);

scanf("%d",&(p1[i].r[j].alcd));

flushall();

(p1[i].r[j].need) = (p1[i].r[j].max) - (p1[i].r[j].alcd);

}

}

for(i=1;i<=n\_r;i++)

for(j=1;j<=n\_p;j++)

av.a[i]-=p1[j].r[i].alcd;

pf("\n Anailable Matrix ");

for(i=1;i<=n\_r;i++)

pf("\t%5d",av.a[i]);

clrscr();

pf("\n\n\t\t\t\t Need Matrix");

for(i=1;i<=n\_p;i++)

{

pf("\n\n\tP%d ||",i);

for(j=1;j<=n\_r;j++)

pf(" %5d ",p1[i].r[j].need);

}

pf("\n\n\t\t\t\t SAFE SEQUENCE: \n\n\n\t\t");

do

{

for(i=1;i<=n\_p;i++)

{

if(p1[i].state==1)

{

x=0;

for(j=1;j<=n\_r;j++)

{

if((av.a[j])>=(p1[i].r[j].need))

x++;

}

if(x==n\_r)

{

for(j=1;j<=n\_r;j++)

(av.a[j])+=(p1[i].r[j].alcd);

pf("\tP%d",i);

p1[i].state=0;

}

}

}

f=0;

for(i=1;i<=n\_p;i++)

if(p1[i].state)

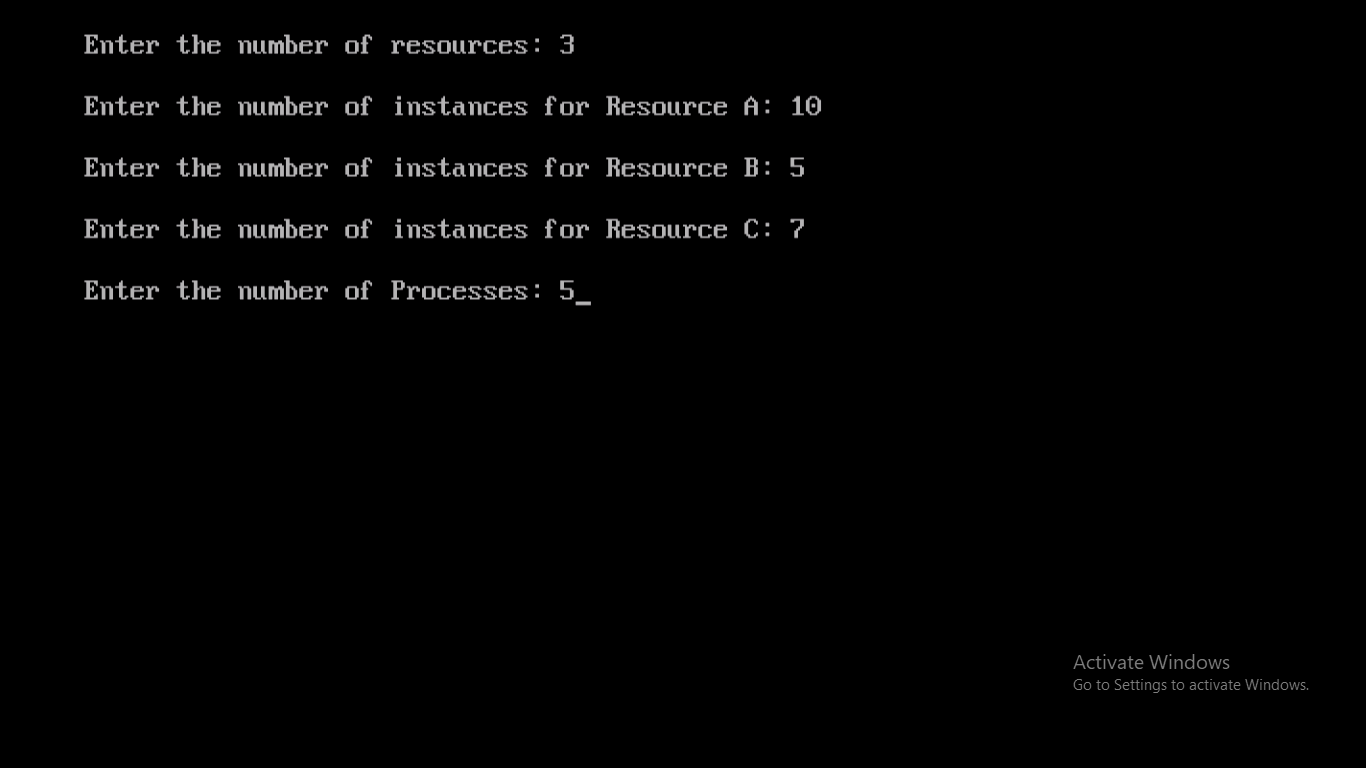
f=1;

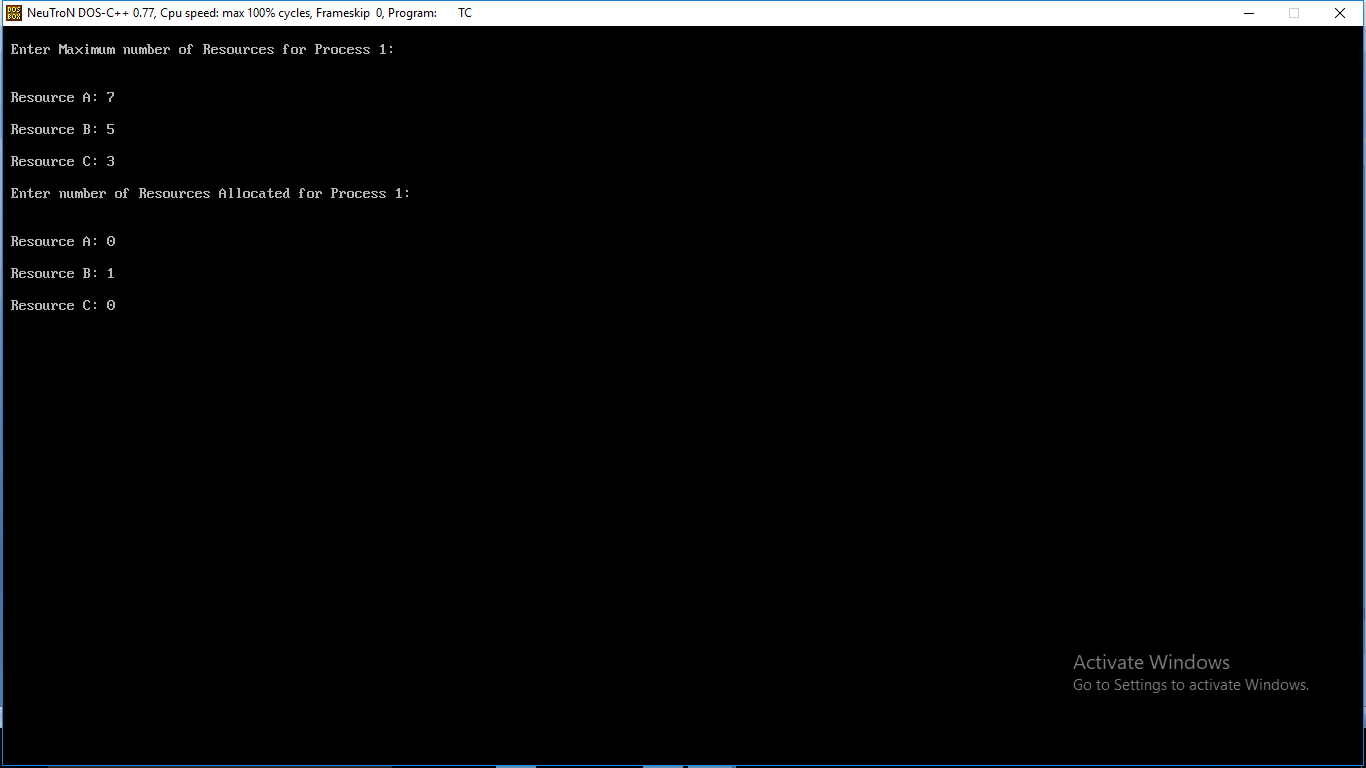
}while(f);

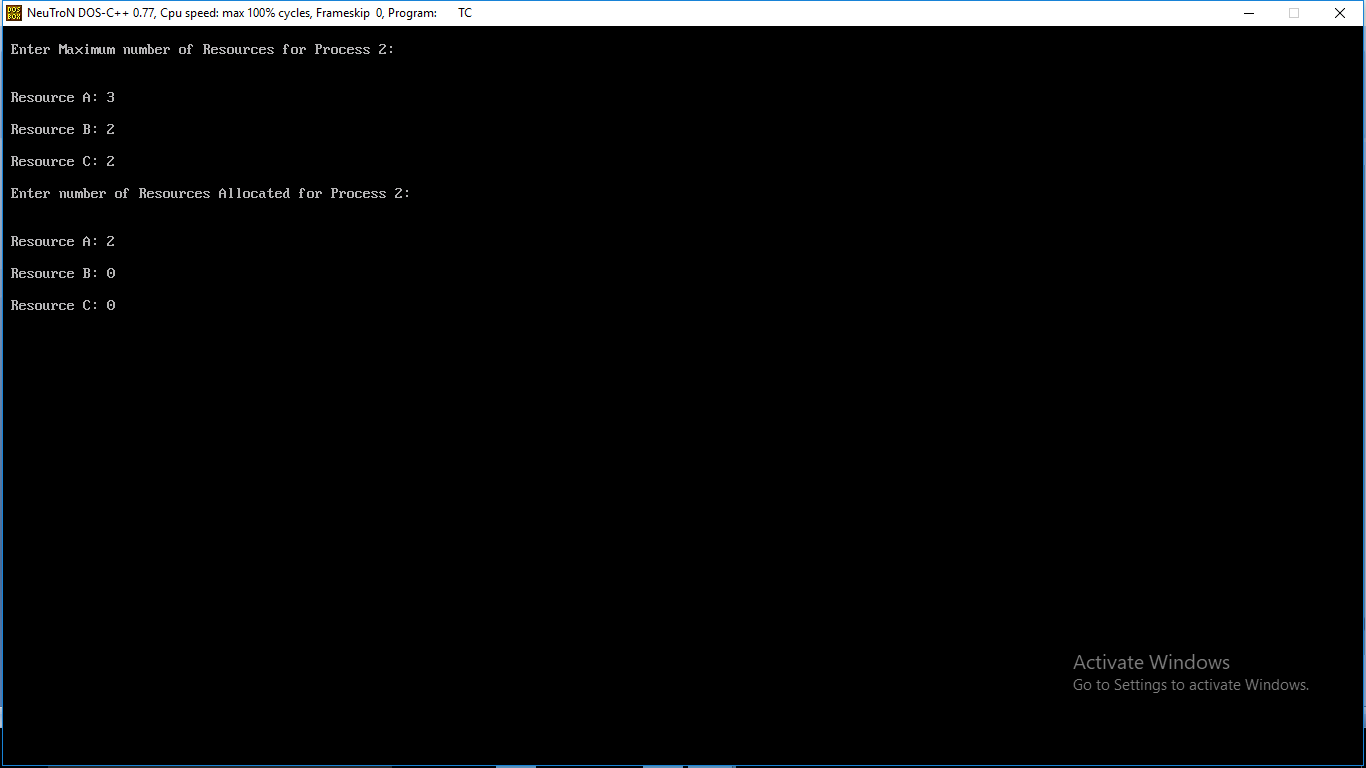
getch();

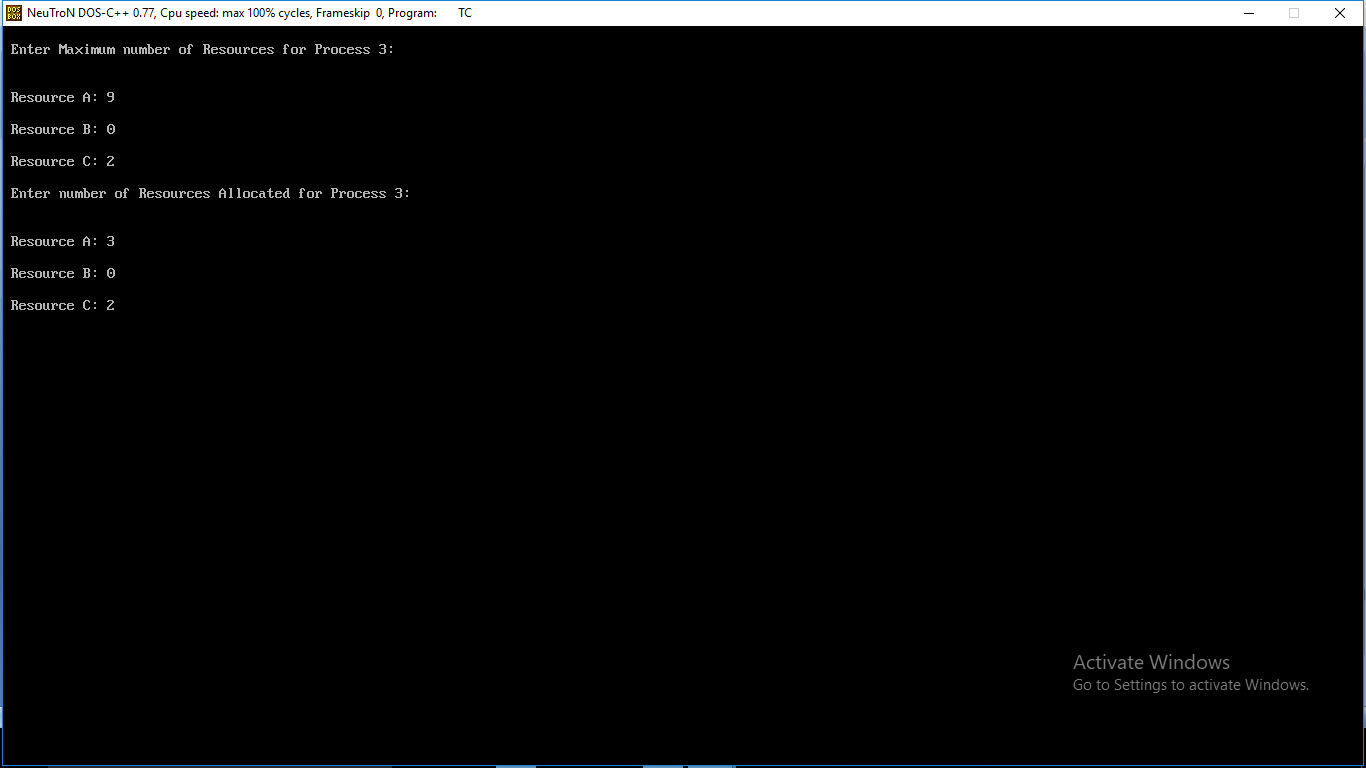
return 0;

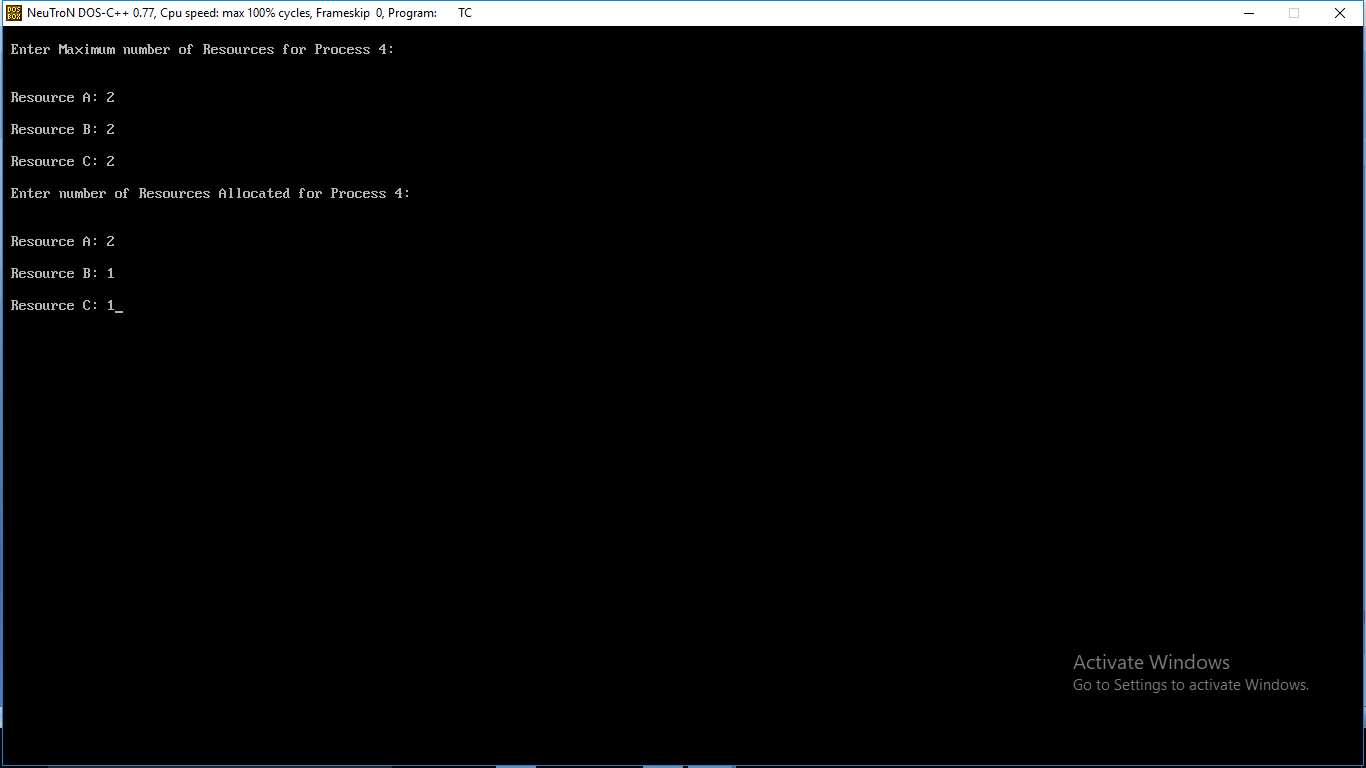
}

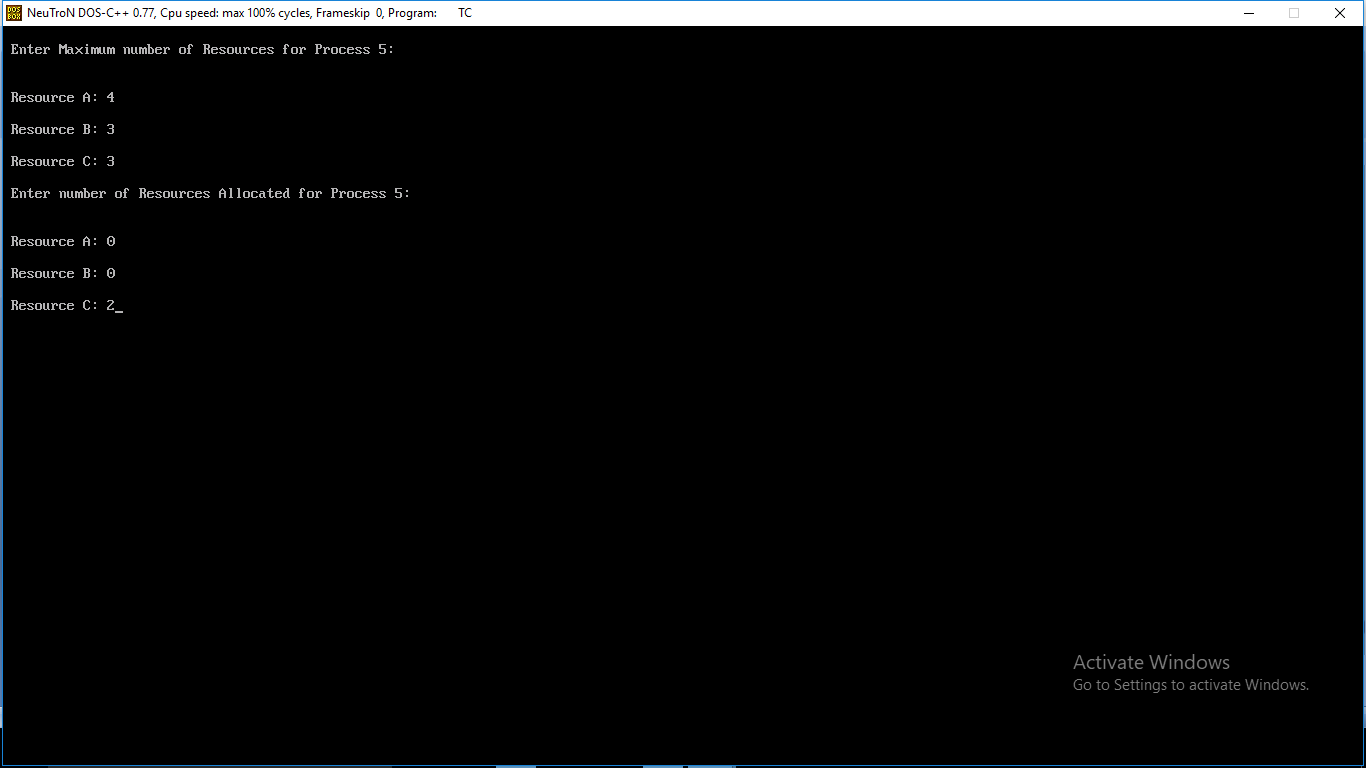


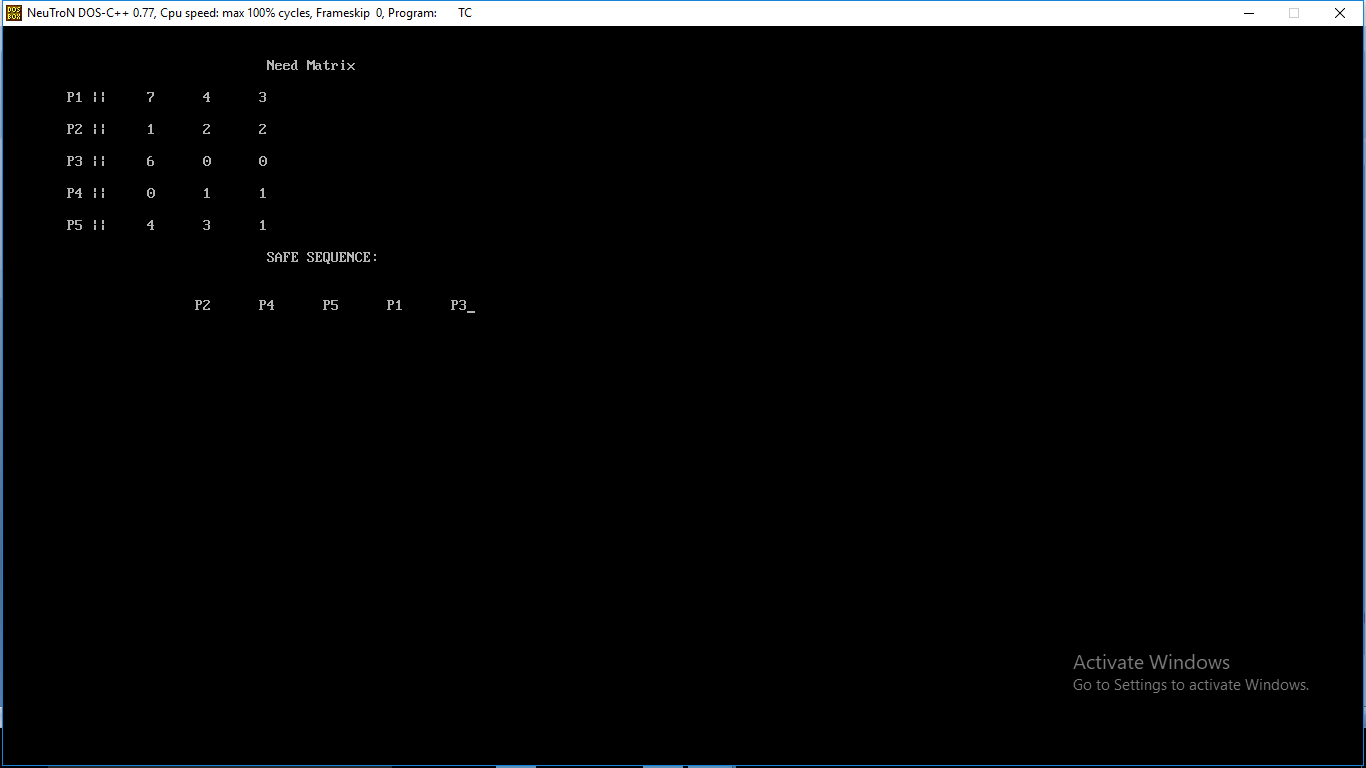












**Week 8 & 9**

**Simulate the below page replacement algorithms**

**a.FIFO b.LRU c.LFU d.OPTIMAL**

**Theory:**

**a) FIFO (First in First Out) algorithm**: FIFO is the simplest page replacement algorithm, the idea behind this is, “Replace a page that page is oldest page of main memory” or “Replace the page that has been in memory longest”. FIFO focuses on the length of time a page has been in the memory rather than how much the page is being used.

**b) LRU (Least Recently Used):** the criteria of this algorithm is “Replace a page that has been used for the longest period of time”. This strategy is the page replacement algorithm looking backward in time, rather than forward.

**c) LFU (Least Frequently Used):** The least frequently used algorithm “select a page for replacement, if the page has not been used for the often in the past” or “Replace page that page has smallest count” for this algorithm each page maintains as counter which counter value shows the least count, replace that page. The frequency counter is reset each time is page is loaded.

**c) OPTIMAL:** The OPTIMAL algorithm “select a page for replacement, if the page has not been used for the often in the future”.

**FIFO**

**Source Code**

**#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;**

**}**

**OUTPUT:**

enter the number of frames

20

Enter The Page Size

3

Enter the Page Sequence

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Total Page Faults

15

**LRU**

**Source Code**

**#include<stdio.h>**

**void main()**

**{**

**int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];**

**printf("Enter no of pages:");**

**scanf("%d",&n);**

**printf("Enter the reference string:");**

**for(i=0;i<n;i++)**

**scanf("%d",&p[i]);**

**printf("Enter no of frames:");**

**scanf("%d",&f);**

**q[k]=p[k];**

**printf("\n\t%d\n",q[k]);**

**c++;**

**k++;**

**for(i=1;i<n;i++)**

**{**

**c1=0;**

**for(j=0;j<f;j++)**

**{**

**if(p[i]!=q[j])**

**c1++;**

**}**

**if(c1==f)**

**{**

**c++;**

**if(k<f)**

**{**

**q[k]=p[i];**

**k++;**

**for(j=0;j<k;j++)**

**printf("\t%d",q[j]);**

**printf("\n");**

**}**

**else**

**{**

**for(r=0;r<f;r++)**

**{**

**c2[r]=0;**

**for(j=i-1;j<n;j--)**

**{**

**if(q[r]!=p[j])**

**c2[r]++;**

**else**

**break;**

**}**

**}**

**for(r=0;r<f;r++)**

**b[r]=c2[r];**

**for(r=0;r<f;r++)**

**{**

**for(j=r;j<f;j++)**

**{**

**if(b[r]<b[j])**

**{**

**t=b[r];**

**b[r]=b[j];**

**b[j]=t;**

**}**

**}**

**}**

**for(r=0;r<f;r++)**

**{**

**if(c2[r]==b[0])**

**q[r]=p[i];**

**printf("\t%d",q[r]);**

**}**

**printf("\n");**

**}**

**}**

**}**

**printf("\nThe no of page faults is %d",c);**

**}**

**OUTPUT:**

enter the number of frames

20

Enter The Page Size

3

Enter the Page Sequence

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Total Page Faults

12

**OPTIMAL**

**#include<stdio.h>**

**#include<conio.h>**

**void main()**

**{**

**int fr[5],i,j,k,t[5],p=1,flag=0,page[25],psz,nf,t1,u[5];**

**clrscr();**

**printf("enter the number of frames:");**

**scanf("%d",&nf);**

**printf("\n enter the page size");**

**scanf("%d",&psz);**

**printf("\nenter the page sequence:");**

**for(i=1; i<=psz; i++)**

**scanf("%d",&page[i]);**

**for(i=1; i<=nf; i++)**

**fr[i]=-1;**

**for(i=1; i<=psz; i++)**

**{**

**if(full(fr,nf)==1)**

**break;**

**else**

**{**

**flag=0;**

**for(j=1; j<=nf; j++)**

**{**

**if(page[i]==fr[j])**

**{**

**flag=1;**

**printf(" \t%d:\t",page[i]);**

**break;**

**}**

**}**

**if(flag==0)**

**{**

**fr[p]=page[i];**

**printf(" \t%d:\t",page[i]);**

**p++;**

**}**

**for(j=1; j<=nf; j++)**

**printf(" %d ",fr[j]);**

**printf("\n");**

**}**

**}**

**p=0;**

**for(; i<=psz; i++)**

**{**

**flag=0;**

**for(j=1; j<=nf; j++)**

**{**

**if(page[i]==fr[j])**

**{**

**flag=1;**

**break;**

**}**

**}**

**if(flag==0)**

**{**

**p++;**

**for(j=1; j<=nf; j++)**

**{**

**for(k=i+1; k<=psz; k++)**

**{**

**if(fr[j]==page[k])**

**{**

**u[j]=k;**

**break;**

**}**

**else**

**u[j]=21;**

**}**

**}**

**for(j=1; j<=nf; j++)**

**t[j]=u[j];**

**for(j=1; j<=nf; j++)**

**{**

**for(k=j+1; k<=nf; k++)**

**{**

**if(t[j]<t[k])**

**{**

**t1=t[j];**

**t[j]=t[k];**

**t[k]=t1;**

**}**

**}**

**}**

**for(j=1; j<=nf; j++)**

**{**

**if(t[1]==u[j])**

**{**

**fr[j]=page[i];**

**u[j]=i;**

**}**

**}**

**printf("page fault\t");**

**}**

**else**

**printf(" \t");**

**printf("%d:\t",page[i]);**

**for(j=1; j<=nf; j++)**

**printf(" %d ",fr[j]);**

**printf("\n");**

**}**

**printf("\ntotal page faults: %d",p+3);**

**// getch();**

**}**

**int full(int a[],int n)**

**{**

**int k;**

**for(k=1; k<=n; k++)**

**{**

**if(a[k]==-1)**

**return 0;**

**}**

**return 1;**

**}**

**OUTPUT:**

enter the number of frames

20

Enter The Page Size

3

Enter the Page Sequence

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Total Page Faults 9

**Week 10 & 11**

**Simulate Various Disk Scheduling Algoirthms**

**a)FCFS b)SSTF c)SCAN d)C SCAN**

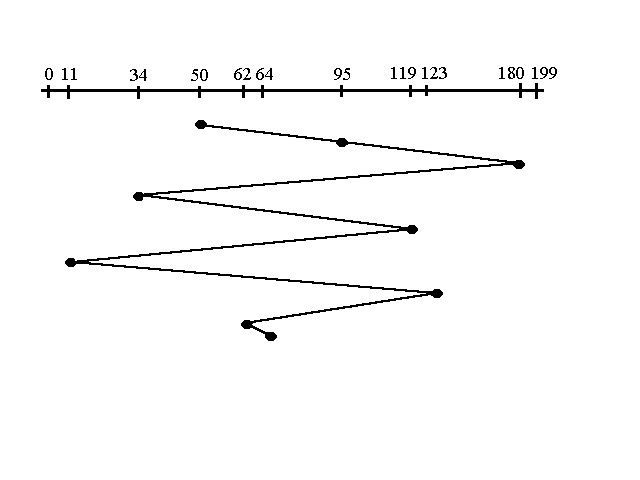
**Disk scheduling**is is done by operating systems to schedule I/O requests arriving for disk. Disk scheduling is also known as I/O scheduling.

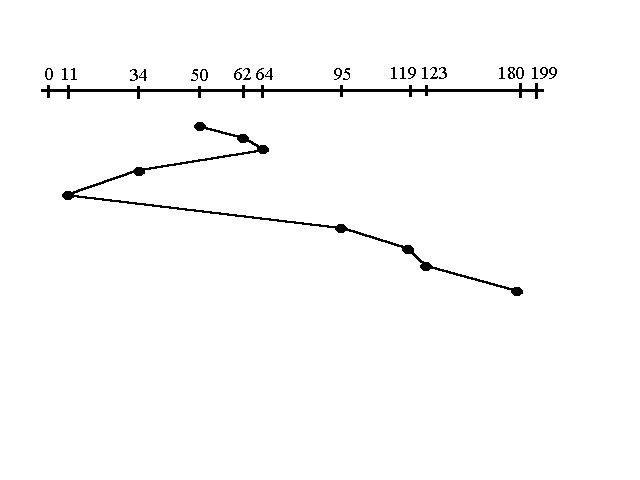
Disk scheduling is important because:

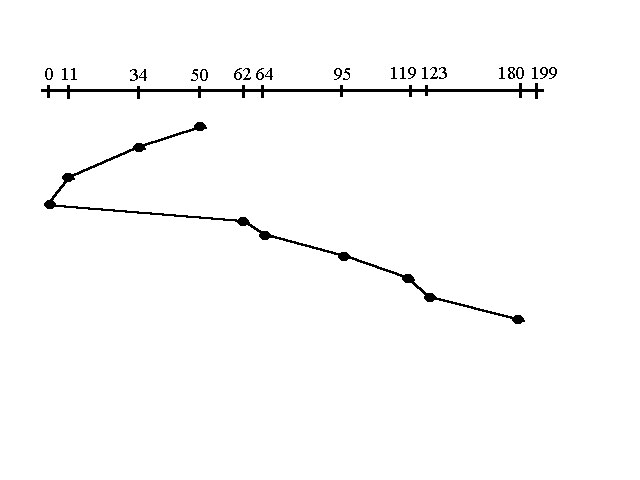
* Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by disk controller. Thus other I/O requests need to wait in waiting queue and need to be scheduled.
* Two or more request may be far from each other so can result in greater disk arm movement.
* Hard drives are one of the slowest parts of computer system and thus need to be accessed in an efficient manner.

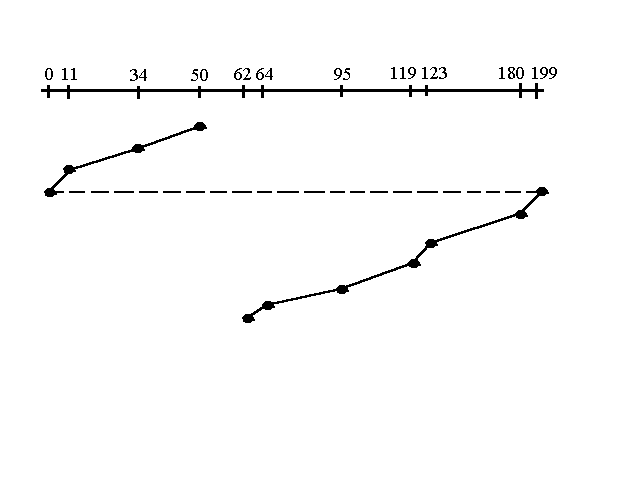
First Come-First Serve (FCFS)  
Shortest Seek Time First (SSTF)  
Elevator (SCAN)   
Circular SCAN (C-SCAN)  
  
Given the following queue -- 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199 let us now discuss the different algorithms.

1. *First Come -First Serve*

*(FCFS) * All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next. For this case it went from 50 to 95 to 180 and so on. From 50 to 95 it moved 45 tracks. If you tally up the total number of tracks you will find how many tracks it had to go through before finishing the entire request. In this example, it had a total head movement of 640 tracks. The disadvantage of this algorithm is noted by the oscillation from track 50 to track 180 and then back to track 11 to 123 then to 64. As you will soon see, this is the worse algorithm that one can use.

2. *Shortest Seek Time First (SSTF)    
In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of. For example the next case would be to move from 62 to 64 instead of 34 since there are only 2 tracks between them and not 18 if it were to go the other way. Although this seems to be a better service being that it moved a total of 236 tracks, this is not an optimal one. There is a great chance that starvation would take place. The reason for this is if there were a lot of requests close to eachother the other requests will never be handled since the distance will always be greater.*

*3. Elevator (SCAN)    
This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks. Once again this is more optimal than the previous algorithm, but it is not the best.*

*4. Circular Scan (C-SCAN)    
Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 track, but still this isn't the mose sufficient.*

#include<stdio.h>

#include<math.h>

void fcfs(int noq, int qu[10], int st)

{

int i,s=0;

for(i=0;i<noq;i++)

{

s=s+abs(st-qu[i]);

st=qu[i];

}

printf("\n Total seek time :%d",s);

}

void sstf(int noq, int qu[10], int st, int visit[10])

{

int min,s=0,p,i;

while(1)

{

min=999;

for(i=0;i<noq;i++)

if (visit[i] == 0)

{

if(min > abs(st - qu[i]))

{

min = abs(st-qu[i]);

p = i;

}

}

if(min == 999)

break;

visit[p]=1;

s=s + min;

st = qu[p];

}

printf("\n Total seek time is: %d",s);

}

void scan(int noq, int qu[10], int st, int ch)

{

int i,j,s=0;

for(i=0;i<noq;i++)

{

if(st < qu[i])

{

for(j=i-1; j>= 0;j--)

{

s=s+abs(st - qu[j]);

st = qu[j];

}

if(ch == 3)

{

s = s + abs(st - 0);

st = 0;

}

for(j = 1;j < noq;j++)

{

s= s + abs(st - qu[j]);

st = qu[j];

}

break; } }

printf("\n Total seek time : %d",s);

}

int main()

{

int n,qu[20],st,i,j,t,noq,ch,visit[20];

printf("\n Enter the maximum number of cylinders : ");

scanf("%d",&n);

printf("enter number of queue elements");

scanf("%d",&noq);

printf("\n Enter the work queue");

for(i=0;i<noq;i++)

{

scanf("%d",&qu[i]);

visit[i] = 0; }

printf("\n Enter the disk head starting posision: \n");

scanf("%d",&st);

while(1)

{

printf("\n\n\t\t MENU \n");

printf("\n\n\t\t 1. FCFS \n");

printf("\n\n\t\t 2. SSTF \n");

printf("\n\n\t\t 3. SCAN \n");

printf("\n\n\t\t 4. EXIT \n");

printf("\nEnter your choice: ");

scanf("%d",&ch);

if(ch > 2)

{

for(i=0;i<noq;i++)

for(j=i+1;j<noq;j++)

if(qu[i]>qu[j])

{

t=qu[i];

qu[i] = qu[j];

qu[j] = t;

}

}

switch(ch)

{

case 1: printf("\n FCFS \n");

printf("\n\*\*\*\*\*\n");

fcfs(noq,qu,st);

break;

case 2: printf("\n SSTF \n");

printf("\n\*\*\*\*\*\n");

sstf(noq,qu,st,visit);

break;

case 3: printf("\n SCAN \n");

printf("\n\*\*\*\*\*\n");

scan(noq,qu,st,ch);

break;

case 4: exit(0); } } }

**OUTPUT:**

Enter the maximum number of cylinders

200

Enter number of queue elements

8

Enter the work queue

98,183,37,122,14,124,65,67

Enter the disk head starting posision

53

MENU

1. FCFS
2. SSTF

3. SCAN

4. EXIT

Enter your choice:1

Total Head movements 640

Enter your choice:2

Total Head Movements 236

Enter your choice:3

Total Head Movements 236

**C SCAN**

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define max 20

#define cymax 199

int i,j,req,ttl\_tracks=0,cp,np,cposn,nposn;

int cyposn[max],temp;

void input()

{

do

{

clreol();

printf("\n Enter the current header position : ");

scanf("%d",&cposn);

}while(cposn>cymax || cposn <=0);

printf("\n Enter the %d I/O Requests : ",req);

cyposn[0] = cposn;

for(i=1;i<=req;i++)

scanf("%d",&cyposn[i]);

}

void CSCAN()

{

for(i=0;i<=req;i++)

{

for(j=0;j<req-i;j++)

{

if(cyposn[j] > cyposn[j+1])

{

temp = cyposn[j];

cyposn[j] = cyposn[j+1];

cyposn[j+1] = temp;

}

}

}

cp=0;

do

{

if(cyposn[cp] == cposn)

break;

cp++;

}while(cp!=req);

printf("\nS.No. Current Position Next Position Displacement \n");

printf("---------------------------------------------------------- \n\n");

i=0,j=cp;

cposn = cyposn[cp];

do

{

if(cposn == cyposn[req])

{ nposn = 199; cp = -1; }

else

nposn = cyposn[++cp];

printf(" %d\t\t%d\t\t%d\t\t%d\n",++i,cposn,nposn,abs(cposn-nposn));

ttl\_tracks += (abs(cposn-nposn));

cposn = nposn == 199 ? 0 : nposn;

}while(nposn != cyposn[j-1]);

printf("---------------------------------------------------------- \n\n");

printf(" Total Tracks Displaced : %d",ttl\_tracks);

}

void main()

{

do

{

clrscr();

printf("\n Enter the number of requests : ");

scanf("%d",&req);

}while(req>max || req <=0);

input();

CSCAN();

getch();

}