LAB # 01

OS Booting Process & Linux Installation

OS Booting Process:

A computer without a program running is just an inert hunk of electronics. The first thing a computer has to do when it is turned on is start up a special program called an *operating system*. The operating system's job is to help other computer programs to work by handling the messy details of controlling the computer's hardware. An operating system is sometimes described as "the first program," one that allows you to run other programs. However, it is usually stored as a file (or, more commonly, a collection of files) on a disk. How does this "first" program get to run?

The process of bringing up the operating system is called *booting.* Your computer knows how to boot because instructions for booting are built into one of its chips, the BIOS (or Basic Input/output System) chip. A boot loader is a program whose task is to load a bigger program, such as the operating system. When you turn on a computer, its memory is usually uninitialized. Hence, there is nothing to run. ROM (Read Only Memory) would contain a small bootloader that would have basic intelligence to read, say, the first sector (512 bytes) of a disk [1].

What is a POST?

- power-on self-test -- one of the first processes that a computer undergoes when booting
- POST tests the computer to ensure that it is working as it is supposed to.
- POST can detect some errors with the processor, motherboard, RAM and other memory, as well as the video card.
- Most BIOS chips use a system of beep codes to indicate the POST status to the user and each BIOS chipset uses a different code.
- The IBM PC BIOS code standard, for example, uses one short beep to indicate a successful POST and two short beeps to indicate a POST error while AMI BIOS uses these same beep codes to indicate a DRAM refresh failure or parity circuit failure, respectively.

What is a BIOS?

- Basic input/output system [2]
- BIOS software stored permanently (*) on a ROM chip on the motherboard

(*) In modern computers BIOS chip can be rewritten, allowing BIOS software to be upgraded.

- BIOS parameters stored in CMOS
- BIOS ROM may be password protected
- The first code run when a PC is powered on
- Identify system devices

What is CMOS?

Complementary Metal-Oxide-Semiconductor. Like most RAM chips, the chip that stores your BIOS settings is manufactured using the CMOS process. It holds a small amount of data, usually 256 bytes. The information on the CMOS chip includes what types of disk drives are installed on your computer, the current date and time of your system clock, and your computer's boot sequence [2].

Since this initial program had to be as small as possible, it would have minimal capabilities. What often happened is that the boot loader would load another boot loader, called a second stage loader, which was more sophisticated. This second stage loader would have error checking, among possibly other features, such as giving the user a choice of operating systems to boot, the ability to load diagnostic software, or enabling diagnostic modes in the operating system. This multi-stage boot loader, having a boot loader load a bigger boot loader, is called chain loading.

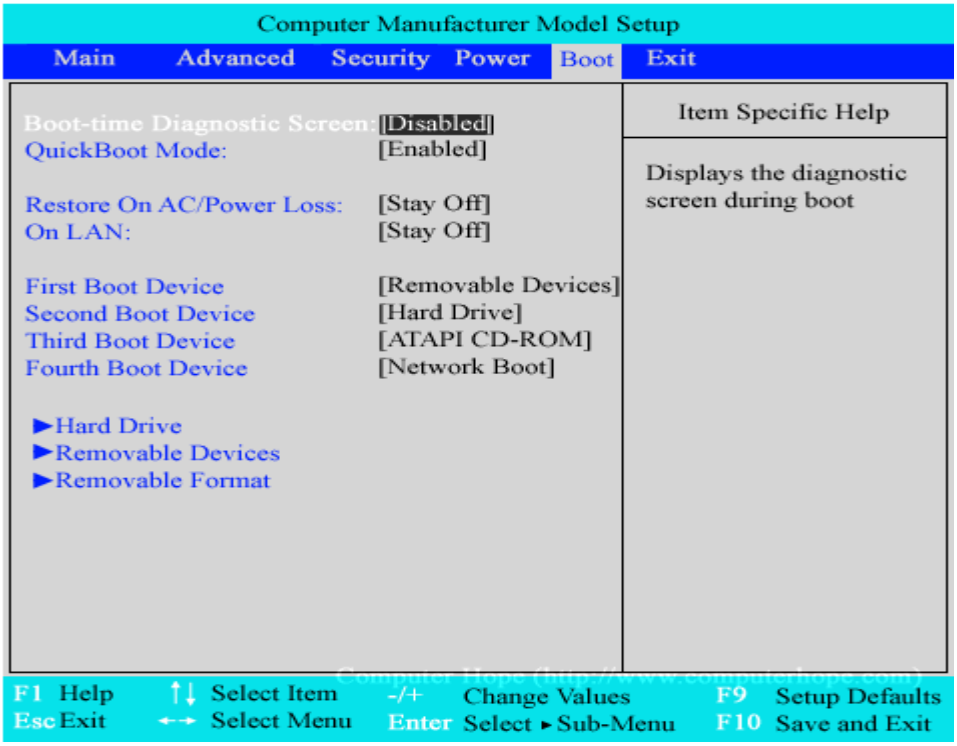


Figure 1: Computer Boot Sequence

The boot loader will often perform some core initialization of the system hardware and will then load the operating system. Once the operating system is loaded, the boot loader transfers control to it and is no longer needed. The operating system will initialize itself, configure the system hardware (e.g., set up memory management, set timers, set interrupts), and load device drivers, if needed.

You may or may not be able to see any of this going on. Back when UNIX systems used text consoles, you'd see boot messages scroll by on your screen as the system started up. Nowadays, Unixes often hide the boot messages behind a graphical splash screen. You may be able to see them by switching to a text console view with the key combination Ctrl-Shift-F1. If this works, you should be able to switch back to the graphical boot screen with a different Ctrl-Shift sequence; try F7, F8, and F9.

But getting the kernel (Operating System) fully loaded and running isn't the end of the boot process; it's just the first stage (sometimes called *run level 1*). After this first stage, the kernel hands control to a special process called 'init' which spawns several housekeeping processes. (Some recent Linuxes use a different program called 'upstart' that does similar things).

Example:

To make the example of the boot process concrete, let us take a look at 32-bit Intel-compatible PCs.

In computing, there exist two type processor i.e., 32-bit and 64-bit. These processor tells us how much memory a processor can have access from a CPU register. For instance,

A 32-bit system can access 2^{32} memory addresses, i.e., 4 GB of RAM or physical memory.
A 64-bit system can access 2^{64} memory addresses, i.e., actually 18-Billion GB of RAM.
In short, any amount of memory greater than 4 GB can be easily handled by it.

An IA–32 (Intel Architecture) based PC is expected to have a BIOS (Basic Input/output System, which comprises the bootloader firmware) in non-volatile memory (ROM). The BIOS contains low-level functions for accessing some basic system devices, such as performing disk I/O, reading from the keyboard, and accessing the video display. It also contains code to load a stage 1 boot loader.

When the CPU is reset at startup, the computer starts execution at memory location 0xffff0. The processor starts up in real mode, which gives it access to only a 20-bit memory address space and provides it with direct access to I/O, interrupts, and memory (32-bit addressing and virtual memory comes into play when the processor is switched to *protected mode*). The location at 0xffff0 is actually at the end of the BIOS ROM and contains a jump instruction to a region of the BIOS that contains start-up code.

1. Power-on self-test (POST)
2. Detect the video card's (chip's) BIOS and execute its code to initialize the video hardware
3. Detect any other device BIOSes and invoke their initialize functions
4. Display the BIOS start-up screen
5. Perform a brief memory test (identify how much memory is in the system)
6. Set memory and drive parameters
7. Configure Plug & Play devices (traditionally PCI bus devices)
8. Assign resources (DMA)
9. Identify the boot device

When the BIOS identifies the boot device (typically one of several disks that has been tagged as the bootable disk), it reads block 0 from that device into memory location 0x7c00 and jumps there.

Stage 1: the Master Boot Record

This first disk block, block 0, is called the Master Boot Record (MBR) and contains the first stage boot loader. Since the standard block size is 512 bytes, the entire boot loader has to fit into this space. The contents of the MBR are:

- First stage boot loader (≤ 440 bytes)
- Disk signature* (4 bytes)
- Disk partition table, which identifies distinct regions of the disk (16 bytes per partition × 4 partitions)

Stage 2: the Volume Boot Record

Once the BIOS transfers control to the start of the MBR that was loaded into memory, the MBR code scans through its partition table and loads the Volume Boot Record (VBR) for that partition. The VBR is a sequence of consecutive blocks starting at the first disk block of the designated partition. The first block of the VBR identifies the partition type and size and contains an Initial Program Loader (IPL), which is code that will load the additional blocks that comprise the second stage boot loader. On Windows (Windows 8), the IPL loads a program called NTLDR (NT Loader), which then loads the operating system.

---

*A disk signature is a unique, identifying number for a hard disk drive or other data storage device, stored as part of the master boot record. Disk signatures are used by the operating system to differentiate between storage devices on your computer.

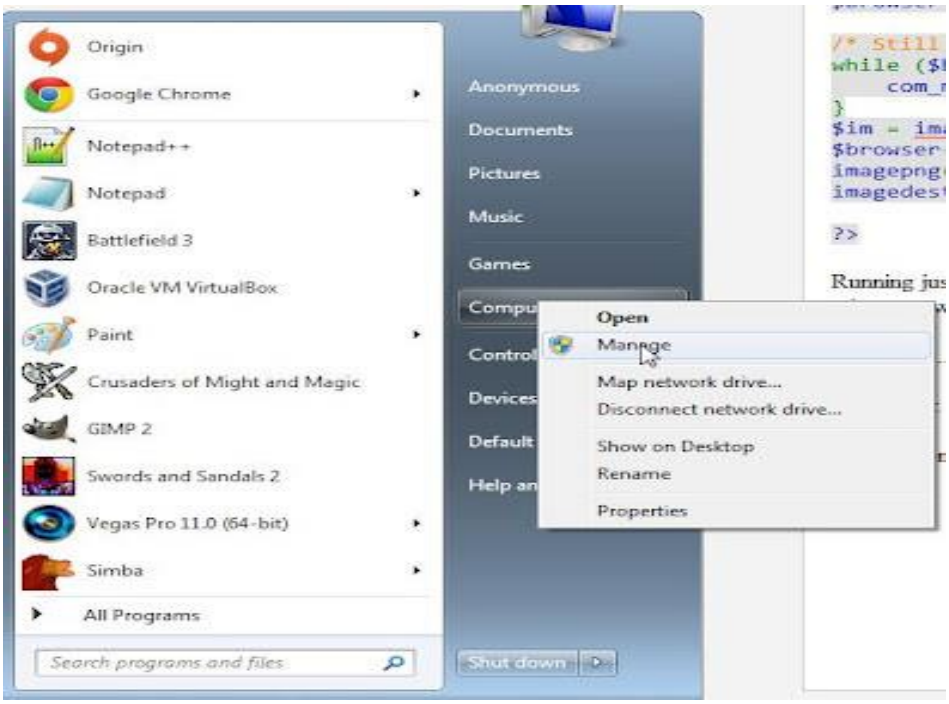
Linux Installation:

Step 1:

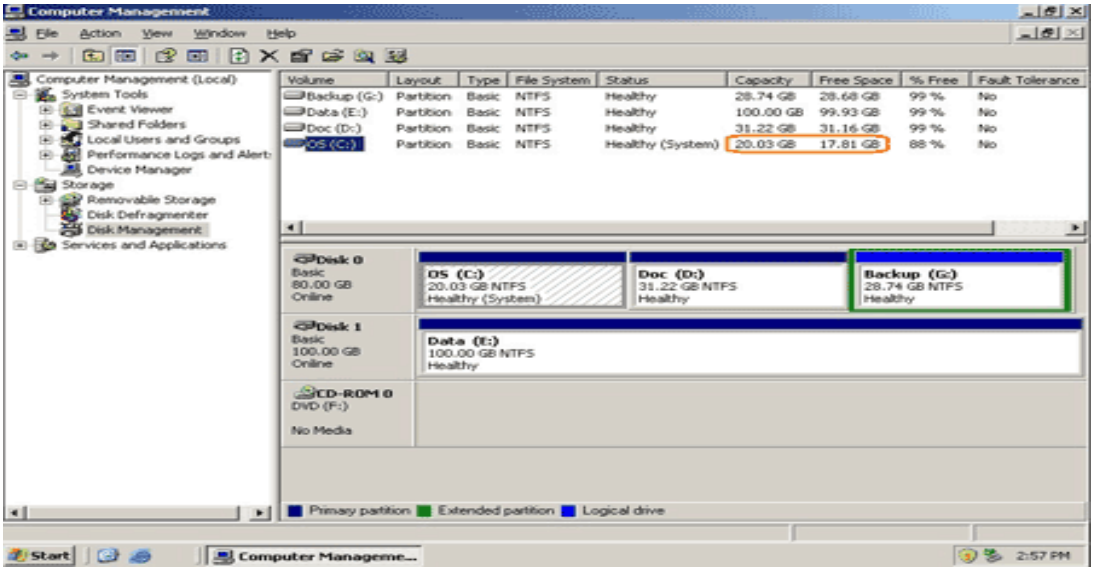
If you wish to install Linux not alongside Windows then installation process is the same as Windows installation. If you wish to dual boot then complete process is listed below.

Step 2:

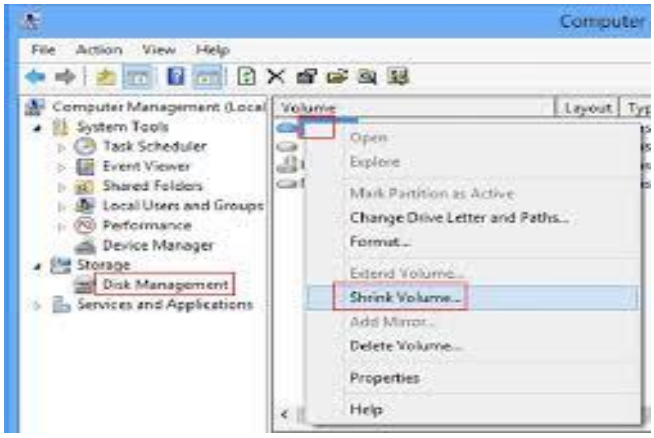
Right click on Computer/This PC and go to Manage

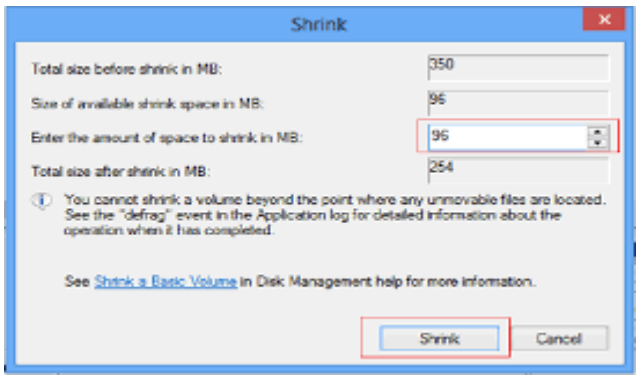


After this the following Window will appear



Right click on any of the Drive D or G and shrink anyone of them to create at least 20 GB free space.





After shrinking a particular Drive free space/ unallocated space will be created. Now it's time to install Ubuntu using this unallocated space.

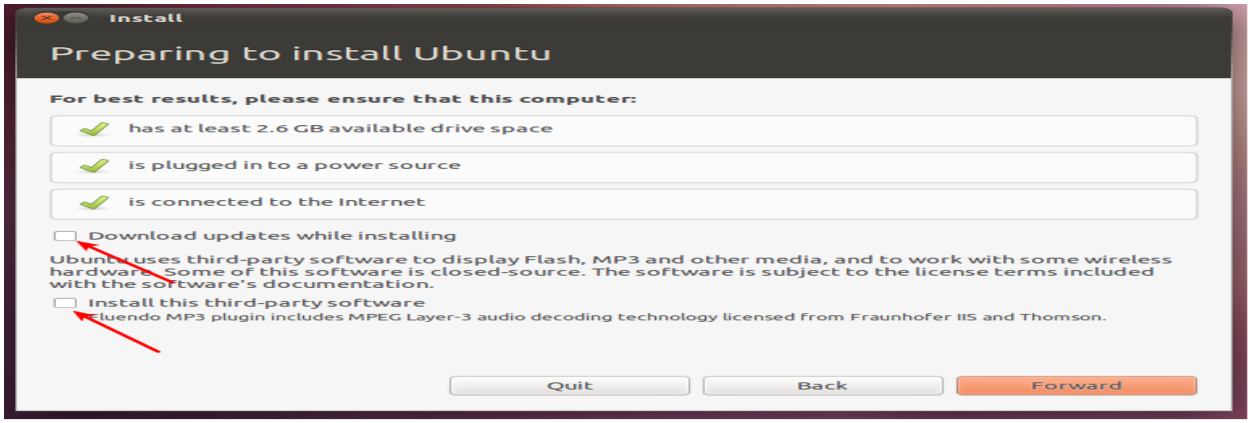
Step 3:
Make a USB bootable (min 8GB) using "Universal-USB-Installer-1.9.6.1".

Step 4:
Now Plugin the bootable USB and restart your computer.
Keep pressing a required key to go to Boot Menu. Select Boot from USB in the Boot Menu. The system will start booting from USB.

Step 5:

Install Ubuntu

Step 6:
Forward

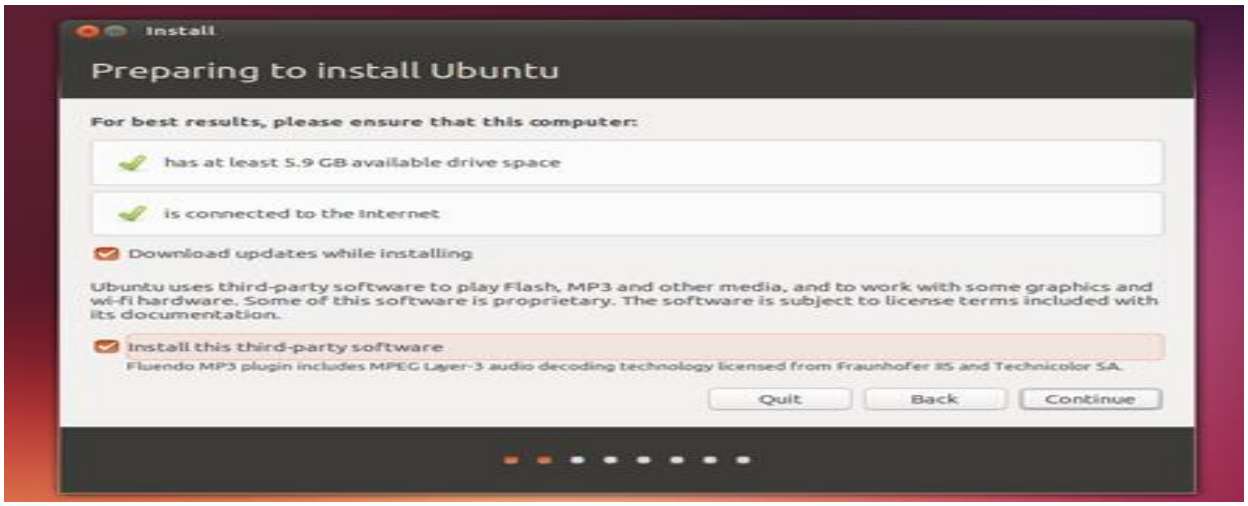


Step 7:

Select something else

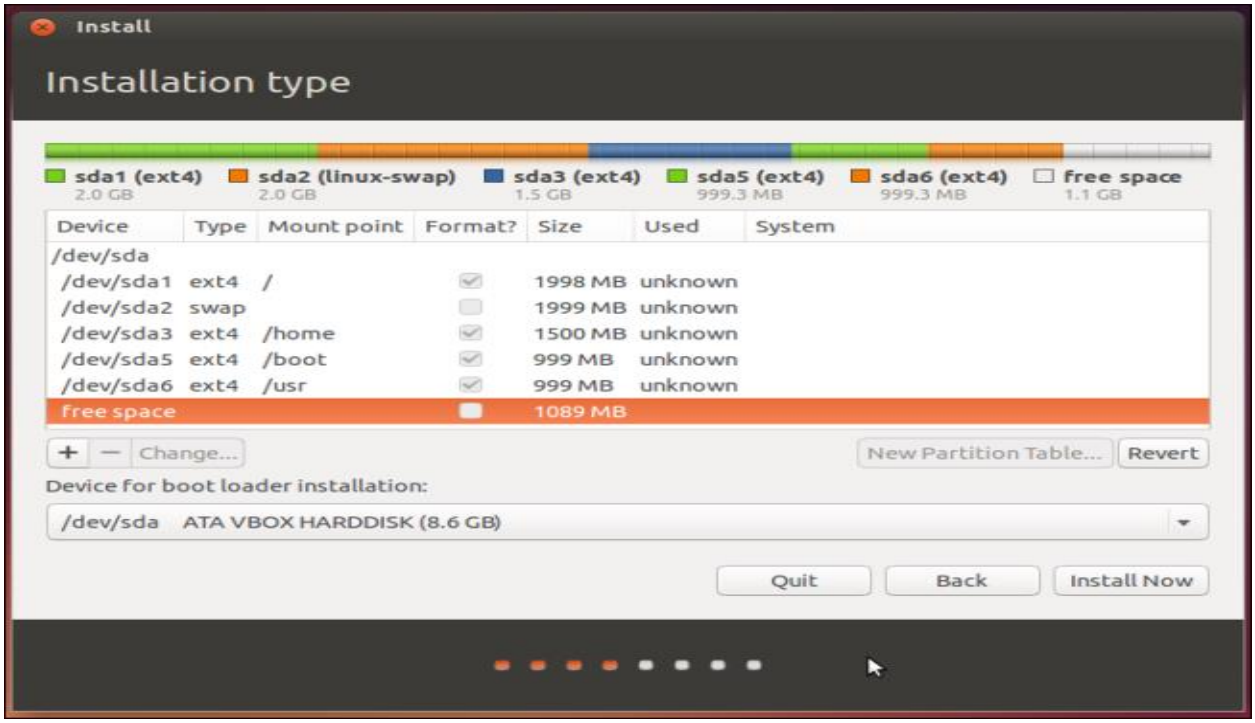
Step 8:

Continue



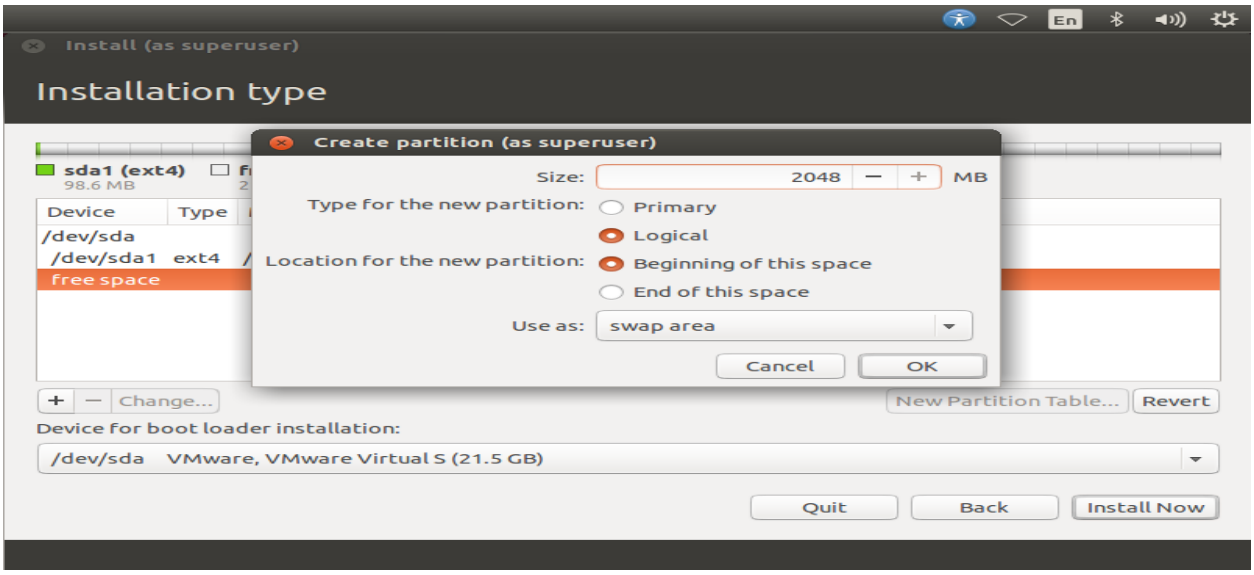
Step 9:

Select free space and then click {+} sign



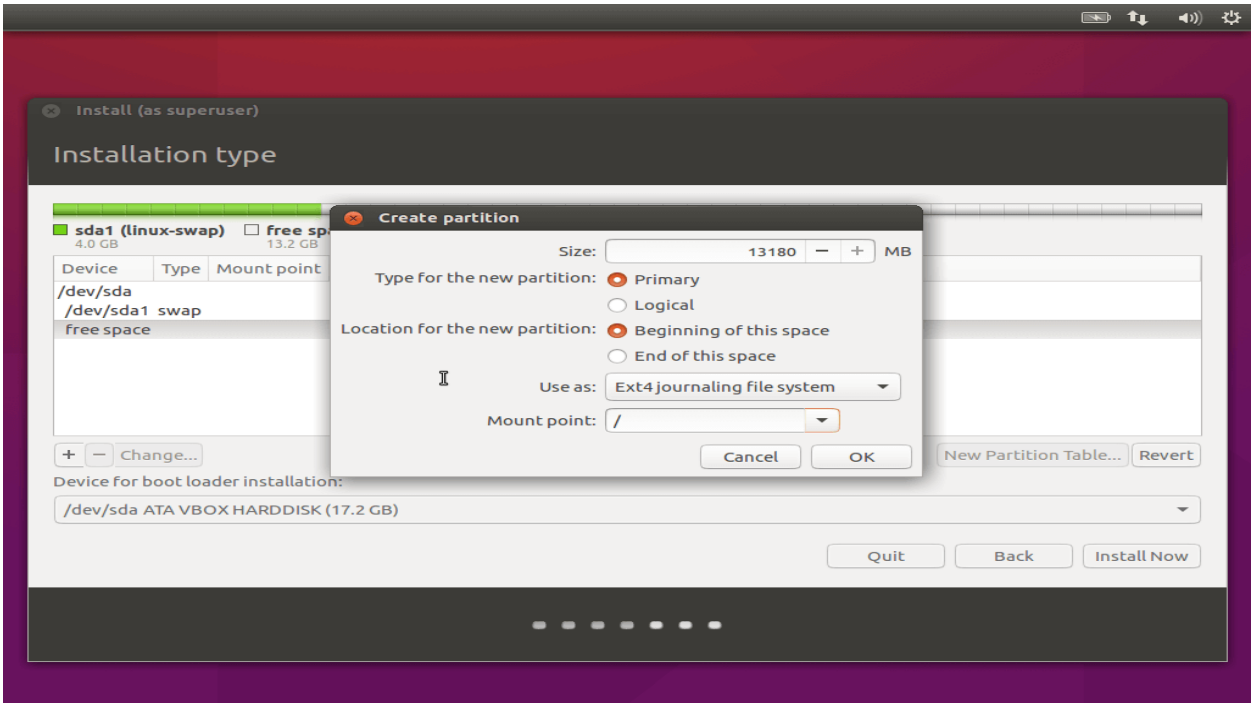
Step 10:

Create Swap Area double of your RAM size



Step 11:

Again click the free space and do as follow and click install Now



Step 10:

Just wait for 5 minutes and you are done with installation. Remove USB the system will get restart.

References

[1] "Booting an Operating System," 14 September 2010. [Online]. Available: https://www.cs.rutgers.edu/~pxk/416/notes/02-boot.html.

[2] "The Boot Process and Operating System," 2015. [Online]. Available: http://www.c-jump.com/CIS24/Slides/Booting/Booting.html.