Partition Table Generator for Cloud OS

Perla Ravi Theja
SCSE
VIT University, Vellore-14

Govinda K
Assistant Professor (SG)
SCSE
VIT University, Vellore-14

P.Swarnalatha
Assistant Professor (SG)
SCSE
VIT University, Vellore-14

ABSTRACT
Each operating system has its own partition utility like parted for Unix based operating systems, disk management utility for Windows operating system and partedUtil for ESX operating system. A partition utility for a particular operating system can able to detect only certain disk labels of other operating systems. We need to develop a Utility by named partition table generator utility which generates a partition table on a raw-disk for a specified disk label. This utility is used to test the capabilities of partition-utility of a particular operating system in cloud environment.

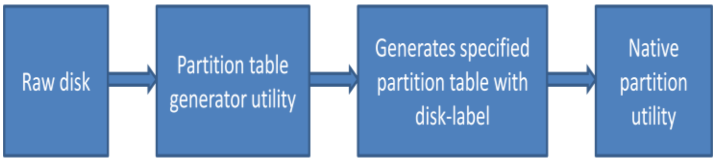
Keywords
Partition table, Partition Utility, Disk

1. INTRODUCTION
Most operating systems allow users to divide a hard disk into multiple partitions in order to organize his data more effectively, making one physical hard disk into several smaller logical hard disks. A hard disk partition is a defined storage space on a hard drive. We store data in file systems, where file systems are stored in hard disk partitions. The Partition table is used to store information about partitions. Every hard disk has a reserved area at the beginning to store Partition table. A partition utility is a partition editor, used for creating, destroying, resizing, copying and checking partitions and file systems on them. Partition utility also creates a specified new disk label which must me one among supported disk labels. The created new disk label will have no partitions.

2. MOTIVATION
The Cloud operating system manages several storage resources and schedules the computing resources in a cluster of servers span across multiple data centers at geographically distributed locations. When a system with particular operating system in cloud demands a storage resource for cloud, suppose cloud serves the storage resource which has been already a partition table with a disk label to the requested system. If the requested system's partition utility cannot recognize the already existing partition table, it may lead to data corruption.

So there is a need to know the capabilities of a native partition utility of a particular operating system that is to know how many disk labels it can recognize to avoid data corruption. To test this in reality we need to install different operating systems creating different partition tables on hard disks, then plucking them and attaching to the system to test and to resume testing process, which takes lot of time and resources. To simplify the process of testing we have to develop a partition generator utility which provide specified partition table with disk label on the raw disk and serve the output to the partition utility to test.

3. ARCHITECTURE FOR PARTITION TABLE GENERATOR UTILITY



4. GUID PARTITION TABLE (GPT) GENERATION ALGORITHM

Algorithm for Guid Partition Table

Input: Disk-name, Disk-size (in Giga Bytes).
Output: Guid Partition table

Algorithm for Guid Partition table
Begin
```
alter←(atoi(Disk-size)*2097152)-1
Filedescriptor1←open(Disk-name, O_RDWR)
read(Filedescriptor1, buffer, 34 * 512)
sudoMbr←(Partition_MBR *) (buffer + 446)
lseek(Filedescriptor1, 446, SEEK_SET)
sudoMbr[0].type←0xee
sudoMbr[0].boot_ind←0x0
sudoMbr[0].startHead←0x0
sudoMbr[0].startSector←0x1
sudoMbr[0].startCylinder←0x0
sudoMbr[0].endHead←0xfe
sudoMbr[0].endSector←0xff
sudoMbr[0].endCylinder←0xff
sudoMbr[0].firstSector←0x1
sudoMbr[0].numSectors←alter
write(Filedescriptor1, sudoMbr, 512)
lseek(Filedescriptor1, 510, SEEK_SET)
label ← (unsigned short *)(buffer + 510)
*label←0xaa55
write(Filedescriptor1, label, 512)
gptHeader ← (Partition_GptHeader *) (buffer + 512)
lseek(Filedescriptor1, 512, SEEK_SET)
gptHeader→signature←0x5452415020494645
gptHeader→revision←0x00010000
gptHeader→headerSize←0x5C
gptHeader→reserved1←0x0
```

```
gptHeader→myLba←0x1
gptHeader→alternateLba←alter
gptHeader→firstUsableLba←0x22
gptHeader→lastUsableLba←gptHeader→alternateLba-33
gptHeader→diskGuid←GUID_SYS
gptHeader→partitionEntryLba←0x2
gptHeader→numberOfPartitionEntries←0x80
gptHeader→sizeofPartitionEntry←0x80
gptEntries→(Partition_GPT *)(buffer + 1024)
gptHeader→partitionEntryArrayCrc32←0x0
calculatedCrc←efi_crc32(gptEntries,gptHeader→sizeofP
artitionEntry*gptHeader→numberOfPartitionEntries)
gptHeader→partitionEntryArrayCrc32←calculatedCrc
gptHeader→headerCrc32=0x0
calCrc←efi_crc32(gptHeader, gptHeader→headerSize)
gptHeader→headerCrc32←calCrc
write(Filedescriptor1, gptHeader, 512)
close(Filedescriptor1)
Filedescriptor2←open(Disk-name, O_RDWR)
lseek(Filedescriptor2,
512*gptHeader→alternateLba, SEEK_SET)
read(Filedescriptor2, buffer1, 512)
gptHeader1←(Secondary_Partition_GptHeader *)(buffer1)
gptHeader1→signature←0x5452415020494645
gptHeader1→revision←0x00010000
gptHeader1→headerSize←0x5C
gptHeader1→reserved1←0x0
gptHeader1→myLba←alter
gptHeader1→alternateLba←0x1
gptHeader1→firstUsableLba←0x22
gptHeader1→lastUsableLba←gptHeader→myLba-33
gptHeader1→diskGuid←GUID_SYS
gptHeader1→partitionEntryLba←0x2
gptHeader1→numberOfPartitionEntries←0x80
gptHeader1→sizeofPartitionEntry←0x80
gptHeader1→partitionEntryArrayCrc32←calculatedCrc
calCrc1←efi_crc32(gptHeader1,gptHeader1→headerSize)
gptHeader1→headerCrc32←calCrc1
write(Filedescriptor2, gptHeader1, 512)
close(Filedescriptor2)
End
```

In computer hardware, GUID Partition Table (GPT)[7][2] is a standard for the layout of the partition table on a physical hard disk. Although it forms a part of the Extensible Firmware Interface (EFI) standard (Intel's proposed replacement for the PC BIOS),[1] it is also used on some BIOS systems because of the limitations of MBR partition tables, which restrict a disk partition's size to a maximum of 2.19 Terabytes. GPT allows for a maximum disk and partition size of 9.4 Zettabytes .

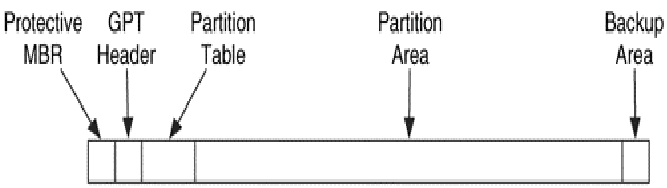


Fig 4.1: A GPT disk has five areas in its layout

As of 2010, most current operating systems support GPT, although some (including Mac OS X and Windows) only support booting to GPT partitions on systems with EFI firmware.

Features

MBR-based partition table schemes insert the partitioning information in the master boot record (MBR) (which on a BIOS system is also the container for code that begins the process of booting the system). In a GPT, partition table information is stored in the GPT header, but to maintain compatibility, GPT retains the MBR entry as the first sector on the disk followed by a primary partition table header, the actual beginning of a GPT.

Like modern MBRs, GPTs use logical block addressing (LBA) in place of the historical cylinder-head-sector (CHS) addressing. Legacy MBR information is contained in LBA 0, the GPT header is in LBA 1, and the partition table itself follows. 64-bit Windows operating systems reserve 16,384 bytes (or 32 sectors) for the GPT, leaving LBA 34 as the first usable sector on the disk.

Hard disk manufacturers are transitioning to 4096-byte sectors. As of 2010, the first such drives continue to present 512-byte physical sectors to the OS, so degraded performance can result when the drive's (hidden) internal 4KiB sector boundaries do not coincide with the 4KiB logical blocks, clusters and virtual memory pages common in many operating systems and file systems. This is a particular problem on writes when the drive is forced to perform two read-modify-write operations to satisfy a single misaligned 4KiB write operation.[5] Such a misalignment occurs by default if the first partition is placed immediately after the GUID partition table, as the next block is LBA 34. The next 4KiB boundary begins with LBA 40.

Drives which boot Intel-based Macs are typically formatted with a GUID Partition Table, rather than with the Apple Partition Map (APM).GPT also provides redundancy, writing the GPT header and partition table both at the beginning and at the end of the disk. Microsoft Windows implementations limit the number of possible partitions to 128.

Legacy MBR (LBA 0)

In the GPT specification, the location corresponding to the MBR in MBR-based disks is structured in a way that prevents MBR-based disk utilities from mis-recognizing, and possibly over-writing, GPT disks. This is referred to as a "protective MBR". In operating systems that support GPT-based boot, it is also used to store the first stage of the boot loader code. A single partition type of 0xEE, encompassing the entire GPT drive, is indicated and identifies it as GPT [3]. Operating systems which cannot read GPT disks will generally recognize the disk as containing one partition of unknown type and no empty space, and will typically refuse to modify the disk unless the user explicitly requests and confirms the deletion of this partition. This minimizes accidental erasures. Furthermore, GPT-aware operating systems will check the protective MBR and if the enclosed partition type is not of type 0xEE or if there are multiple partitions defined on the target device, the device should not be manipulated.

If the disk exceeds 2 Terabytes -the maximum partition size represent able using the 32-bit LBAs of the legacy MBR (assuming a 512 byte block size)—-the size of this partition is marked as 2 Terabytes, ignoring the rest of disk.

Apple's Boot Camp Intel based Apple Mac’s software creates a hybrid partition table to allow the booting of Windows (which at the time of Boot Camp's creation did not support GPT or EFI). In this system the protective partition is reduced in size to cover from sector 1 to the sector before the first regular partition included in the hybrid MBR. Additional MBR partitions are then defined to correspond to the next three GPT partitions.

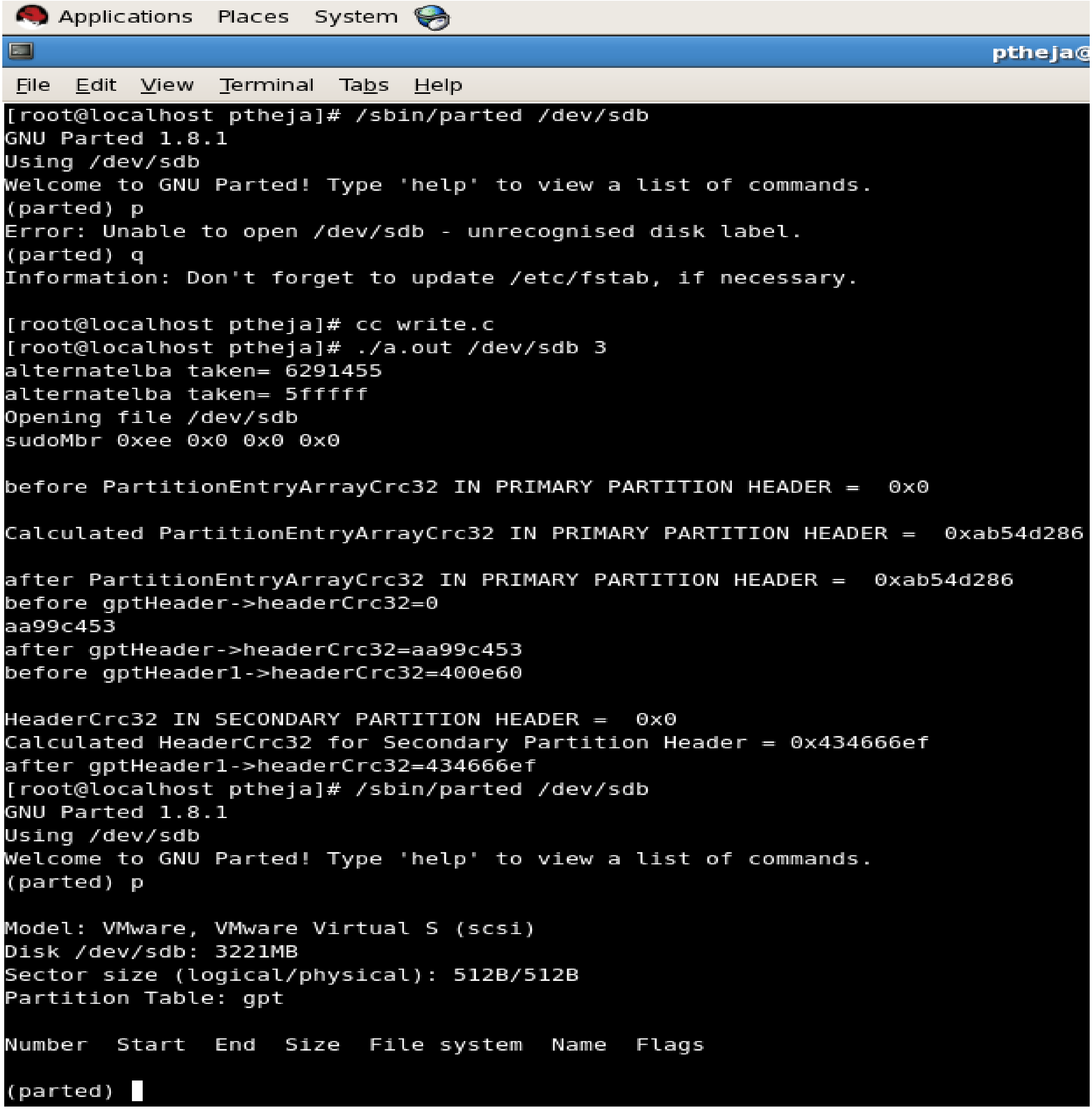


Fig 4.3: Screen shot for creating GUID Partition Table

Partition table header (LBA 1)

The partition table header defines the usable blocks on the disk. It also defines the number and size of the partition entries that make up the partition table. On 64-bit Windows Server 2003 machines, 128 partitions can be created. There are 128 partition entries reserved, each 128 bytes long. (The EFI specification requires that a minimum of 16,384 bytes be reserved for the partition table, so this gives space for 128 partition entries. The header contains the disk GUID(Globally Unique Identifier)[7]. It records its own size and location (always LBA 1) and the size and location of the secondary GPT header and table (always the last sectors on the disk). Importantly, it also contains a CRC32 checksum for itself and for the partition table, which may be verified by the firmware, boot loader and/or operating system on boot. Because of this, hex editors should not be used to modify the contents of the GPT. Such modification would render the checksum invalid. In this case, the primary GPT may be overwritten with the secondary one by disk recovery software. If both GPTs contain invalid checksums, the disk would be unusable.

Table 4.1: GUID Partition Table Contents

Mnemonic	Byte Offset	Byte Length	Description
Signature	End	Last	Identifies EFI-compatible partition table header. This value must contain the string “EFI PART”, 0x5452415020494645.
Revision	8	4	The specification revision number that this header complies to. For version 1.0 of the specification the correct value is 0x00010000.
HeaderSize	12	4	Size in bytes of the GUID Partition Table Header.
HeaderCRC32	16	4	CRC32 checksum for the GUID Partition Table Header structure. The ranged defined by HeaderSize is “check-summed”.
Reserved	20	4	Must be Zero.
MyLBA	24	8	The LBA that contains this data structure.
AlternateLBA	32	8	LBA address of the alternate GUID Partition Table Header.
FirstUsableLBA	40	8	The first usable logical block that may be contained in a GUID Partition Entry.
LastUsableLBA	48	8	The last usable logical block that may be contained in a GUID Partition Entry.
DiskGUID	56	16	GUID that can be used to uniquely identify the disk.
PartitionEntryLBA	72	8	The starting LBA of the GUID Partition Entry array
NumberOfPartitionEntries	80	4	The number of Partition Entries in the GUID Partition Entry array.
SizeOfPartitionEntry	84	4	The size, in bytes, of each the GUID Partition Entry structures in the GUID Partition Entry array. Must be a multiple of 8.
PartitionEntryArrayCRC32	88	4	The CRC32 of the GUID Partition Entry array. Starts at Partition Entry LBA and is (NumberOfPartitionEntries)* (SizeOfPartitionEntry in byte length.)
Reserved	92	92	The rest of the block is reserved by EFI and must be zero.

Partition entries (LBA 2–33)

The GPT [2][7] uses simple and straightforward entries to describe partitions. The first 16 bytes designate the partition type GUID. For example, the GUID for an EFI System partition is {C12A7328-F81F-11D2-BA4B-00A0C93EC93B}. The second 16 bytes contain a GUID unique to the partition. Starting and ending 64-bit LBAs are also recorded here, and space is allocated for partition names and attributes. As is the nature and purpose of GUIDs, no central registry is needed to ensure the uniqueness of the GUID partition on type designators.

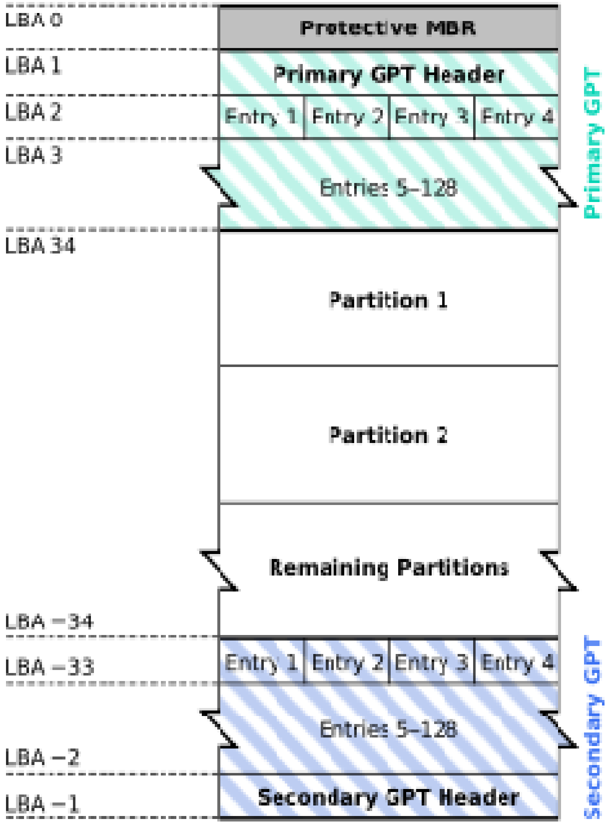


Fig 4.2: GUID Partition Table (GPT) Scheme

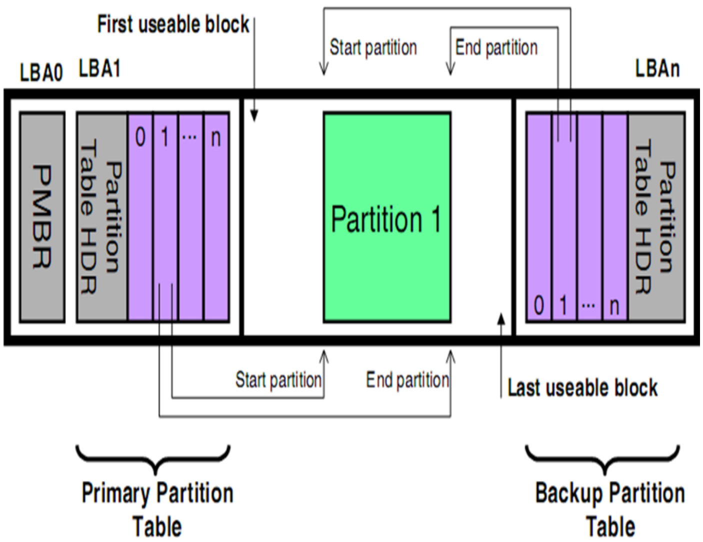


Fig 4.4: GUID Partition Table (GPT) Scheme

The following test must be performed to determine if a GUID Partition Table [2] is valid:

a.)Check the GUID Partition Table Signature.

b.)Check that the MyLBA entry points to the LBA that contains the GUID Partition Table.

c.)Check the CRC of the GUID Partition Entry Array.

If the GUID Partition Table is the primary table, stored at LBA 1:
a.) Check the AlternateLBA to see if it is a valid GUID Partition Table

If the primary GUID Partition Table is corrupt:
i.) Check the last LBA of the device to see if it has a valid GUID Partition Table.
ii.) If valid backup GUID Partition Table found, restore primary GUID Partition Table.

The primary and backup GUID Partition Tables must be valid before an attempt is made to grow the size of a physical volume. This is due to the GUID Partition Table recovery scheme depending on locating the backup GUID Partition Table at the end of the physical device. A volume may grow in size when disks are added to a RAID device [5]. As soon as the volume size is increased the backup GUID Partition Table must be moved to the end of the volume and the primary and backup GUID Partition Table Headers must be updated to reflect the new volume size.

The SizeOfPartitionEntry variable in the GUID Partition Table Header defines the size of a GUID Partition Entry. The GUID Partition Entry starts in the first byte of the GUID Partition Entry and any unused space at the end of the defined partition entry is reserved space and must be set to zero.

Each partition record [2] contains a Unique Partition GUID variable that uniquely identifies every partition that will ever be created. Any time a new partition record is created a new GUID must be generated for that partition, and every partition is guaranteed to have a unique GUID [5]. The partition record also contains 64-bit logical block addresses for the starting and ending block of the partition. The partition is defined as all the logical blocks inclusive of the starting and ending usable LBA defined in the GUID Partition Table Header. The partition record contains a partition type GUID that identifies the contents of the partition. This GUID is similar to the OS type field in the legacy MBR. Each file system must publish its unique GUID.

Table 4.2: GUID Partition Entry Contents

Mnemonic	**Byte Offset**	**Byte Length**	**Description**
Partition Type Guid	0	16	Unique id that defines the purpose and type of this Partition. A value of zero defines that this partition record is not being used.
Unique Partition Guid	16	16	Guid that is unique for every partition record.
StartingLBA	32	16	Starting LBA of the partition defined by this record.
EndingLBA	40	8	Ending LBA of the partition defined by this record.
Attributes	48	8	Attribute bits, all bits reserved by EFI.
Partition Name	56	72	Unicode string.

5. MASTER BOOT RECORD PARTITION TABLE (MBR) GENERATION ALGORITHM

Algorithm for Master Boot Record Partition Table

```
Input: Disk-name
Output: MBR Partition table

Algorithm for MBR Partition table
Begin
    Char buffer[512]
    Filedescriptor1←open(Disk-name, O_RDWR)
    lseek(Filedescriptor1, 510, SEEK_SET)
    label←(unsigned short *)(buffer +446)
    *label←0xaa55
     write(Filedescriptor1,label,512)
     close(Filedescriptor1)
End
```

A master boot record (MBR) is a type of boot sector popularized by the IBM Personal Computer [6]. It consists of a sequence of 512 bytes located at the first sector of a data storage device such as a hard disk [4]. MBRs are usually placed on storage devices intended for use with IBM PC-compatible systems. The most commonly encountered partition system is the DOS-style partition. DOS partitions have been used with Intel IA32 hardware (i.e., i386 / x86) for many years, yet there is no official specification. There are many Microsoft and non-Microsoft documents that discuss the partitions, but there is no standard reference. In addition to there being no standard reference, there is also no standard name. Microsoft now calls disks using this type of partition system Master Boot Record (MBR) disks.

The MBR [6] may be used for one or more of the following:

- Holding a partition table, describes the partitions of a storage device. In this context the boot sector may also be called a partition sector.
- Bootstrapping an operating system. The BIOS built into a PC-compatible computer loads the MBR from the storage device and passes execution to machine code instructions at the beginning of the MBR.
- Uniquely identifying individual disk media, with a 32-bit disk signature, even though it may never be used by the operating system.

Because of the broad popularity of PC-compatible computers, the MBR format is widely used, to the extent of being supported by computer operating systems in addition to other pre-existing or cross-platform standards for bootstrapping and partitioning.

The MBR is the first block (sector) on the disk media. The boot code on the MBR is not executed by EFI firmware. The MBR may optionally contain a signature. The MBR signature must be maintained by operating systems, and is never maintained by EFI firmware. The unique signature in the MBR is only 4 bytes in length, so it is not a GUID. EFI does not specify the algorithm that is used to generate the unique signature. The uniqueness of the signature is defined as all disks in a given system having a unique value in this field.

The MBR contains four partition records that define the beginning and ending LBA addresses that a partition consumes on a hard disk. The partition record contains a legacy Cylinder Head Sector (CHS) address that is not used in EFI. EFI utilizes the starting LBA entry to define the starting LBA of the partition

on the disk. The size of the partition is defined by the size in LBA field.
The boot indicator field is not used by EFI firmware. The operating system indicator value of 0xEFdefines a partition that contains an EFI file system. The other values of the system indicator are not defined by this specification. This will allow drivers and applications, including OS loaders, to easily search for handles that represent EFI System Partitions.

EFI [2] defines a valid legacy MBR as follows. The signature at the end of the MBR must be 0xaa55. Each MBR partition record must be checked to make sure that the partition that it defines physically resides on the disk. Each partition record must be checked to make sure it does not overlap with other partition records [1]. A partition record that contains an OS Indicator value of zero or a SizeInLBA value of zero may be ignored. If any of these checks fail the MBR is not considered valid.

The MBR [6] is a simple method of describing up to four partitions. However, many systems require more partitions than that. For example, consider a 12GB disk that the user wants to divide into six 2GB partitions because he is using multiple operating systems. We cannot describe the six partitions by using the four partition table entries. The solution to this design problem is what makes DOS partitions so complex. The basic theory behind the solution is to use one, two, or three of the entries in the MBR for normal partitions and then create an "extended partition" that will fill up the remainder of the disk. A primary file system partition is a partition whose entry is in the MBR and the partition contains a file system or other structured data. A primary extended partition is a partition whose entry is in the MBR, and the partition contains additional partitions.

Table 5.1: MBR Partition Table Contents

BootCode	0	440	Code used on legacy Intel architecture system to select a partition record and load the first block (sector) of the partition pointed to by the partition record. This code is not executed on EFI systems.
uniqueMBR Signature	440	4	Unique Disk Signature, this is an optional feature and not on all hard drives. This value is always written by the OS and is never written by EFI firmware.
Unknown	444	2	Unknown
Partition Record	446	16*4	Array of four MBR partition records.
Signature	510	2	Must be 0xaa55.

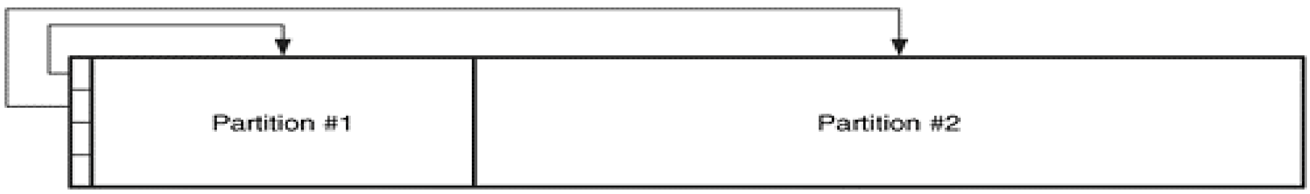


Fig 5.1: A Basic DOS disk with two partitions and one MBR

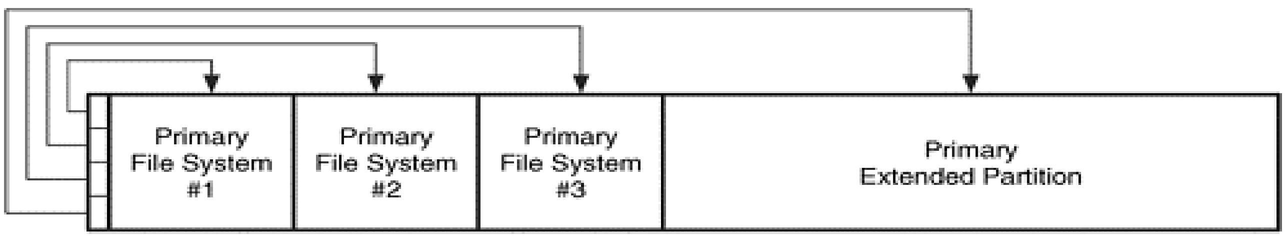
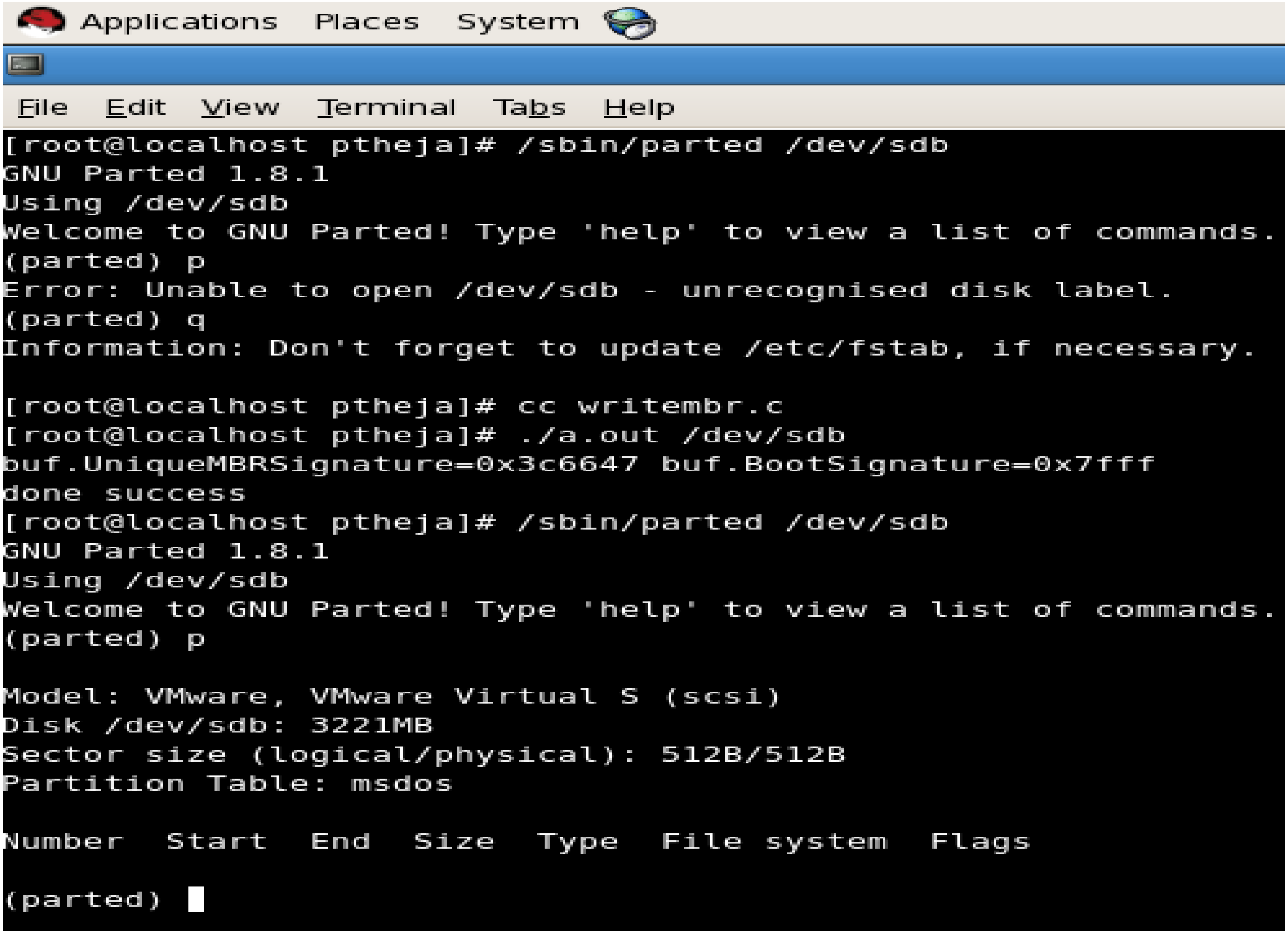


Fig 5.2: A DOS disk with three primary file system partitions and one primary secondary partition.



```
Applications   Places   System
File   Edit   View   Terminal   Tabs   Help
[root@localhost ptheja]# /sbin/parted /dev/sdb
GNU Parted 1.8.1
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) p
Error: Unable to open /dev/sdb - unrecognised disk label.
(parted) q
Information: Don't forget to update /etc/fstab, if necessary.

[root@localhost ptheja]# cc writembr.c
[root@localhost ptheja]# ./a.out /dev/sdb
buf.UniqueMBRSignature=0x3c6647 buf.BootSignature=0x7fff
done success
[root@localhost ptheja]# /sbin/parted /dev/sdb
GNU Parted 1.8.1
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) p

Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 3221MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start  End  Size  Type  File system  Flags

(parted)
```

Fig 5.3: Screen shot for creating MBR Partition Table

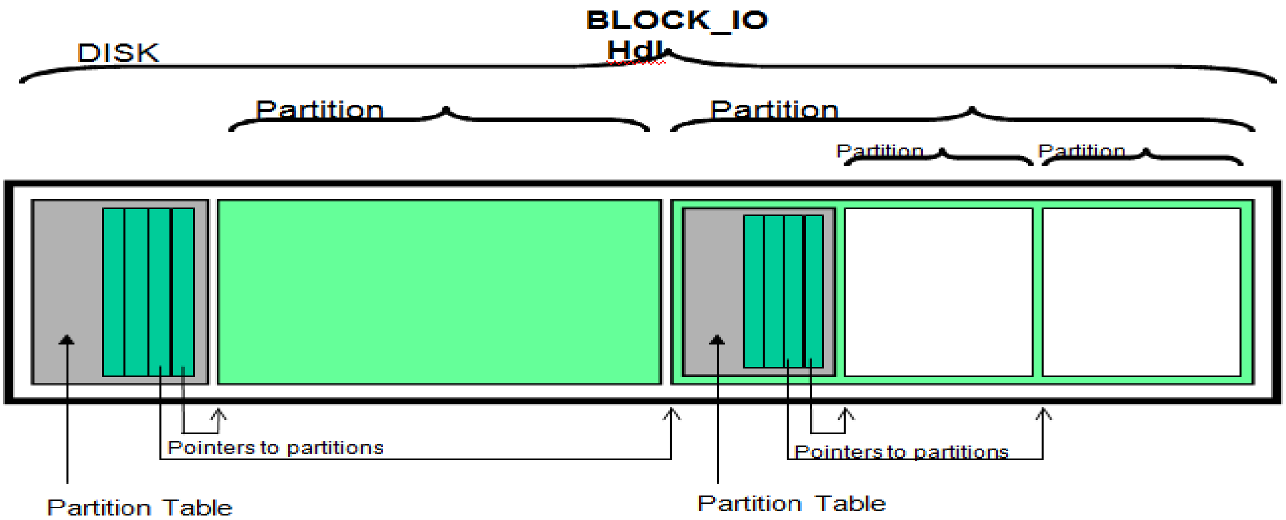


Fig 5.4: Nesting of Legacy MBR Partition Table

6. MAC PARTITION TABLE (MAC) GENERATION ALGORITHM

Algorithm for MAC Partition Table

```
Input: Disk-name
Output: MAC Partition table

Algorithm for MAC Partition table
Begin
    Char buf[1024]
    Filedescriptor1←open(Disk-name, O_RDWR)
    info←(unsigned long *)(buf+0)
    lseek(Filedescriptor1, 0, SEEK_SET)
    *info←0x600000025245
     write(Filedescriptor1,info,6)
     signature←(unsigned short *)(buf+512)
    lseek(Filedescriptor1, 512, SEEK_SET)
    *signature←0x4D50
     write(Filedescriptor1,signature,2);
     map_count←(unsigned int *)(buf+516)
    lseek(Filedescriptor1, 516, SEEK_SET)
    * map_count ← 0x02000000
     write(Filedescriptor1, map_count,4);
     block_count←(unsigned int *)(buf+520)
    lseek(Filedescriptor1, 520, SEEK_SET)
    * block_count ← 0x01000000
     write(Filedescriptor1, block_count,4);
     size←(unsigned int *)(buf+524)
    lseek(Filedescriptor1, 524, SEEK_SET)
    *size← 0x3f000000
     write(Filedescriptor1,size,4);
     name←(unsigned long *)(buf+528)
    lseek(Filedescriptor1, 528, SEEK_SET)
    *name← 0x000000656c707041
     write(Filedescriptor1,name,8);
     type1←(unsigned long *)(buf+560)
    lseek(Filedescriptor1, 560, SEEK_SET)
    *type1← 0x61705f656c707041
     write(Filedescriptor1,type1,8);
     type2←(unsigned long *)(buf+568)
    lseek(Filedescriptor1, 568, SEEK_SET)
    *type2← 0x5f6e6f6974697472
     write(Filedescriptor1,type2,8);
     type3←(unsigned long *)(buf+576)
    lseek(Filedescriptor1, 576, SEEK_SET)
    *type3← 0x0070616d
     write(Filedescriptor1,type3,4);
     boot_load2←(unsigned int *)(buf+598)
    lseek(Filedescriptor1, 598, SEEK_SET)
    *boot_load2←0x4D50
     write(Filedescriptor1,boot_load2,4);
     close(Filedescriptor1)
End
```

Apple Partitions

Systems running the Apple Macintosh operating system [1] are not as common as those running Microsoft Windows, but they have been increasing in popularity with the introduction of Mac OS X, a UNIX-based operating system. The partitions that we will describe here can be found in the latest Apple laptops and desktops running OS X, older systems that are running Macintosh 9. The partition map also can be used in the disk image files that a Macintosh system uses to transmit files. The disk image file is similar to a zip file in Windows or a tar file in Unix. The files in the disk image are stored in a file system, and the file system may be in a partition. The design of the partition system in an Apple system is a nice balance between the complexity of DOS-based partitions and the limited number of partitions that we will see in the BSD disk labels. The Apple partition can describe any number of partitions, and the data structures are in consecutive sectors of the disk.

The Apple partitions [1][3] are described in the partition map structure, which is located at the beginning of the disk. The firmware contains the code that processes this structure, so the map does not contain boot code like we saw in the DOS partition table. Each entry in the partition map describes the starting sector of the partition, the size, the type, and the volume name. The data structure also contains values about data inside of the partition, such as the location of the data area and the location of any boot code. The first entry in the partition map is typically an entry for itself, and it shows the maximum size that the partition map can be. Apple creates partitions to store hardware drivers, so the main disk for an Apple system has many partitions that contain drivers and other non-file system content. Figure 6.2shows an example layout of an Apple disk with three file system partitions and the partition for the partition map.

TABLE 6.1: APPLE PARTITION TABLE CONTENTS

Byte Range	Description	Essential
0-1	Signature Value (0x504D)	NO
2-3	Reserved	NO
4-7	Total Number of partitions	YES
8-11	Starting sector of partition	YES
12-15	Size of partition in sectors	YES
16-47	Name of partition in ASCII	NO
48-79	Type of partition in ASCII	NO
80-83	Starting sector of data area in partition	NO
84-87	Size of data area in sectors	NO
88-91	Status of partition	NO
92-95	Starting sector of boot code	NO
96-99	Size of boot code in sectors	NO
100-103	Address of boot loader code	NO
104-107	Reserved	NO
108-111	Boot code entry point	NO

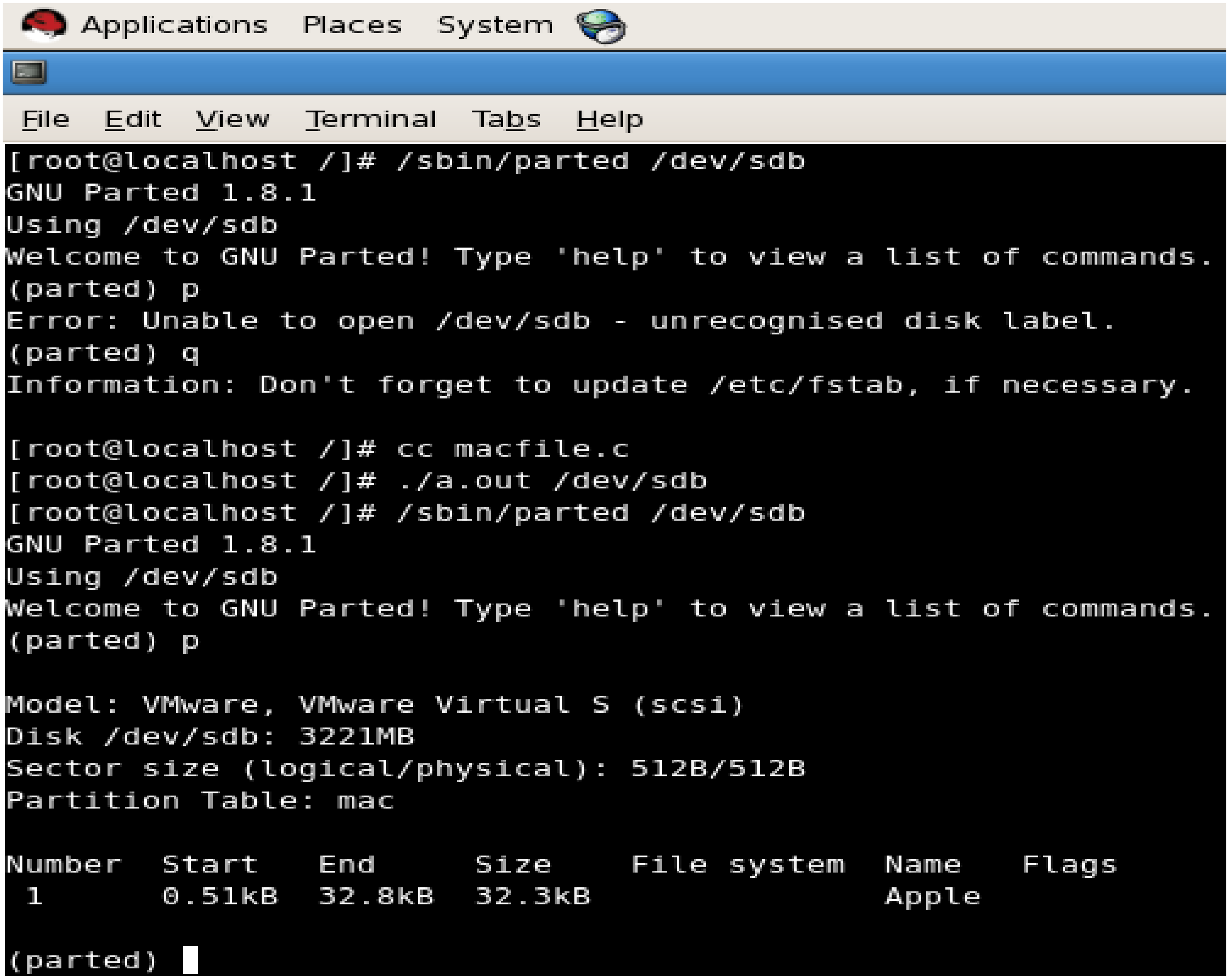


Fig 6.1: Screen shot for creating MAC Partition Table

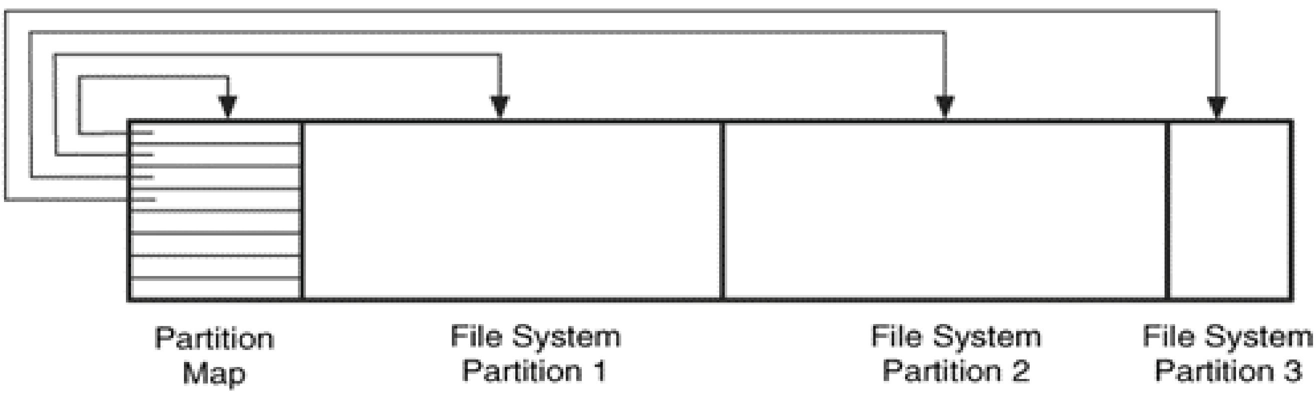


Fig 6.2: An Apple disk with one partition map partition and three file system partitions.

7. SUN PARTITION TABLE (SUN) GENERATION ALGORITHM

Algorithm for SUN Partition Table

```
Input: Disk-name
Output: SUN Partition table

Algorithm for SUN Partition table
Begin
    Char buffer[512]
    Filedescriptor1←open(Disk-name, O_RDWR)
    nheads← (unsigned short *)(buf+437)
    lseek(Filedescriptor1,437,SEEK_SET)
    *nheads←0xff
    write(Filedescriptor1,nheads,1)
    ntracks← (unsigned short *)(buf+439)
    lseek(Filedescriptor1,439,SEEK_SET)
    *ntracks←0x003f
    write(Filedescriptor1,ntracks,1)
    label1← (unsigned short *)(buf+508)
    lseek(Filedescriptor1,508,SEEK_SET)
    label1← (unsigned short *)(buf+508)
    *label1←0xbeda
    write(Filedescriptor1,label1,2)
    disk_speed← (unsigned short *)(buf+420)
    lseek(Filedescriptor1,420,SEEK_SET)
    *disk_speed←0x1815
    write(Filedescriptor1,disk_speed,2)
    phy_cylinders← (unsigned short *)(buf+422)
    lseek(Filedescriptor1,422,SEEK_SET)
    *phy_cylinders←0x0501
    write(Filedescriptor1,phy_cylinders,2)
    version← (unsigned short *)(buf+431)
    lseek(Filedescriptor1,431,SEEK_SET)
    *version←0x01
    write(Filedescriptor1,version,1)
    phy_cylinders1← (unsigned short *)(buf+432)
    lseek(Filedescriptor1,432,SEEK_SET)
    *phy_cylinders1←0x0501
    write(Filedescriptor1,phy_cylinders1,2)
    disk_size← (unsigned long *)(buf+464)
    lseek(Filedescriptor1,464,SEEK_SET)
    *disk_size←0xc5fa3f00
    write(Filedescriptor1,disk_size,4)
    checksum← (unsigned short *)(buf+510)
    lseek(Filedescriptor1,510,SEEK_SET)
    *checksum←0x9e2e
    write(Filedescriptor1,checksum,2)
End
```

The Solaris operating system from Sun Microsystems is used in large servers and desktop systems [5]. It uses two different types of partitioning systems depending on the size of the disk and the version of Solaris. Solaris 9 introduced support for file systems larger than 1-terrabyte and uses EFI partition tables because they have a 64-bit address field. All other versions of Solaris use data structures that are similar to the BSD disk label. In fact, the primary data structure is also called a disk label, although the actual layout of the structure is different. This may not be surprising considering that the layout is even different for Sparc-based Solaris and i386-based Solaris.

On a Sparc system, the disk label structure is created in the first sector of the disk, sector 0.Sectors 1–15 contain the "bootblock," which is the boot code for the system, and sectors 16 and above are partitioned to store file systems and swap space. Solaris uses a UFS file system, and we will see in Chapter 16 that the file system starts in sector 16. We can see the layout of an example Sparc disk in Figure 7.1.

The layout of the disk label can be confusing because the layout information for the Solaris partitions is not in one location. There are two data structures within the disk label structure that hold the partition data. The VTOC structure contains the number of partitions and the type, permissions, and timestamps for each, but the starting location and size of each partition is stored in the disk map structure. The contents of the Sparc disk label are given in Table 7.1.

Table 7.1: Data Structure for the SUN SPARC Disk label.

Byte Range	Description	Essential
0-127	ASCII Label	NO
128-261	Sparc VTOC	YES
262-263	Sectors to skip, writing	NO
264-265	Sectors to skip, reading	NO
266-419	Size of partition in sectors	NO
420-421	Disk speed	NO
422-423	Number of physical cylinders	NO
424-425	Alternates per cylinder	NO
426-429	Reserved	NO
430-431	Interleave	NO
432-433	Number of data cylinders	NO
434-435	Number of alternate cylinders	NO
436-437	Number of heads	YES
438-439	Number of sectors per track	YES
440-443	Reserved	NO
444-451	Partition #1 disk map	YES
452-459	Partition #2 disk map	YES
460-467	Partition #3 disk map	YES
468-475	Partition #4 disk map	YES
476-483	Partition #5 disk map	YES
484-491	Partition #6 disk map	YES
492-499	Partition #7 disk map	YES
500-507	Partition #8 disk map	YES
508-509	Signature Value (0xDABE)	NO
510-511	Checksum	NO

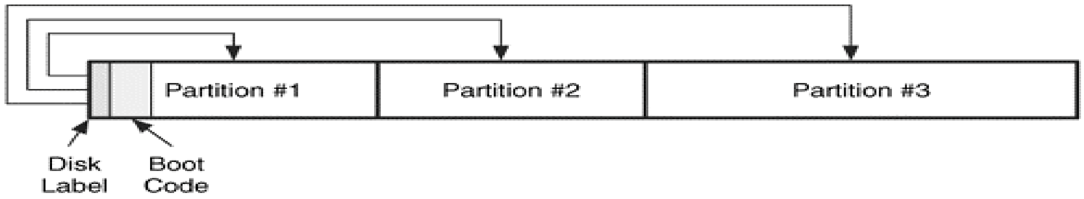


Fig 7.1: An SUN disk with three dos partitions, the final one contains a disk label and three SUN partitions.

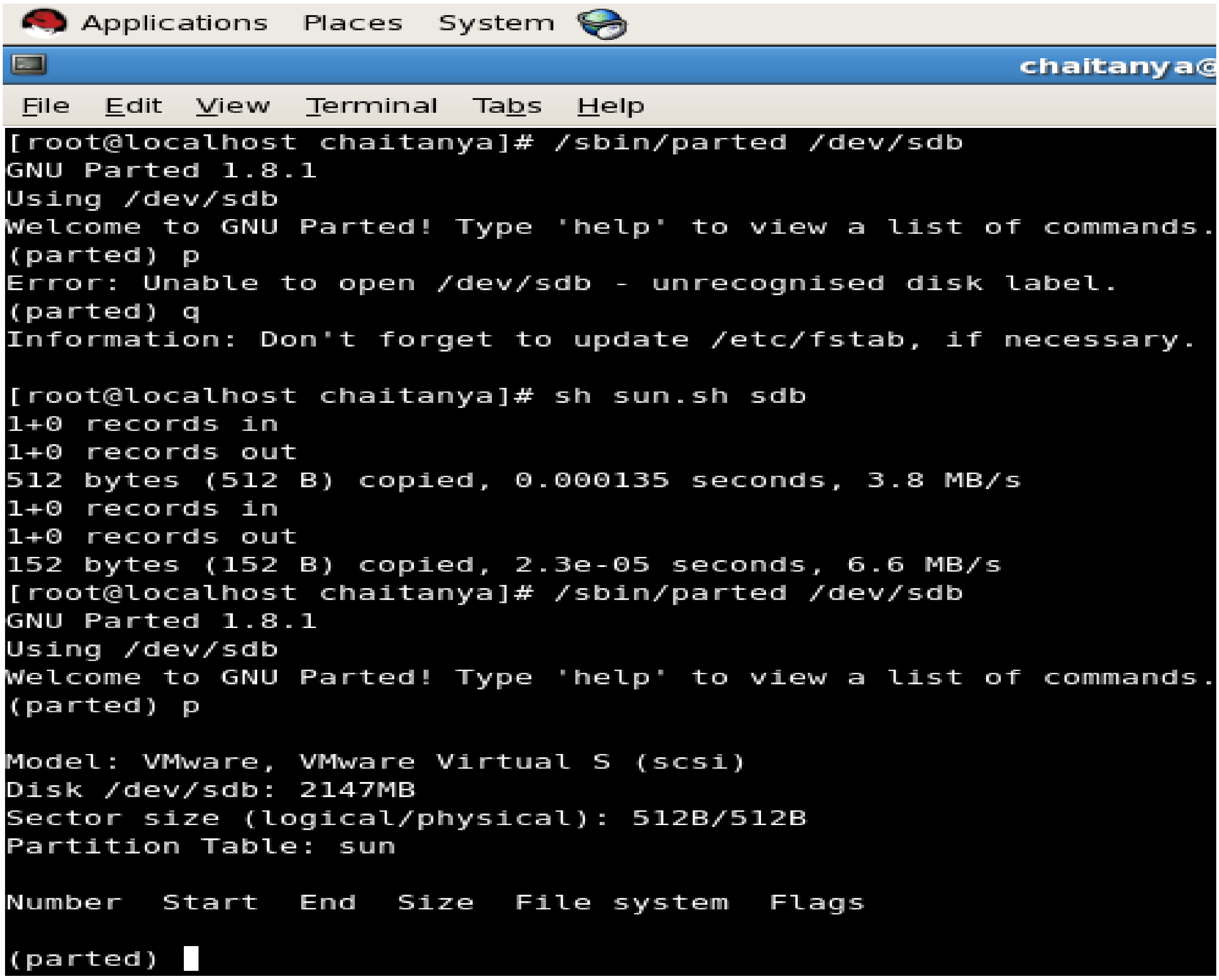


Fig 7.2: Screen shot for creating SUN Partition Table.

Table 7.2: Data Structure for the SUN SPARC Disk label Disk Map.

Byte Range	Description	Essential
0-3	Starting Cylinder	YES
4-7	Size	YES

The VTOC can be found in bytes 128 to 261. This structure tells you how many partitions there are (bytes 12–13) and the flags, type, and a timestamp for each partition. The VTOC has the fields given in Table 7.3.

Table 7.3: Data Structure for the VTOC in SUN SPARC Disk label.

Byte Range	Description	Essential
0-3	Version (0x01)	NO
4-11	Volume Name	NO
12-13	Number of Partitions	YES
14-15	Partition #1 type	NO
16-17	Partition #1 flags	NO

18-19	Partition #2 type	NO
20-21	Partition #2 flags	NO
22-23	Partition #3 type	NO
24-25	Partition #3 flags	NO
26-27	Partition #4 type	NO
28-29	Partition #4 flags	NO
30-31	Partition #5 type	NO
32-33	Partition #5 flags	NO
34-35	Partition #6 type	NO
36-37	Partition #6 flags	NO
38-39	Partition #7 type	NO
40-41	Partition #7 flags	NO
42-43	Partition #8 type	NO
44-45	Partition #8 flags	NO
46-57	Boot info	NO
58-59	Reserved	NO
60-63	Signature Value - 0x600DDEEE	NO
64-101	Reserved	NO
102-105	Partition #1 timestamp	NO
106-109	Partition #2 timestamp	NO
110-113	Partition #3 timestamp	NO
114-117	Partition #4 timestamp	NO
118-121	Partition #5 timestamp	NO
122-125	Partition #6 timestamp	NO
126-129	Partition #7 timestamp	NO
130-133	Partition #8 timestamp	NO

8. AMIGA PARTITION TABLE (AMIGA) GENERATION ALGORITHM

Algorithm for AMIGA Partition Table

```
Input: Disk-name
Output: AMIGA Partition table

Algorithm for AMIGA Partition table
Begin
    Char buffer[1024]
    Filedescriptor1←open(Disk-name, O_RDWR)
    RIGIDISK_ID_NAME← (unsigned int*)(buf+1024)
    lseek(Filedescriptor1,1024,SEEK_SET)
    * RIGIDISK_ID_NAME ←0x4b534452
    write(Filedescriptor1, RIGIDISK_ID_NAME,4)
    NUM_BOOT_BLOCKS← (unsigned int*)(buf+1028)
    lseek(Filedescriptor1,1028,SEEK_SET)
    * NUM_BOOT_BLOCKS ←0x40000000
    write(Filedescriptor1, NUM_BOOT_BLOCKS,4)
    reserved1← (unsigned int*)(buf+1048)
    lseek(Filedescriptor1,1048,SEEK_SET)
    * reserved1←0xffffffff
    write(Filedescriptor1, reserved1,4)
    reserved2← (unsigned int*)(buf+1052)
    lseek(Filedescriptor1,1052,SEEK_SET)
    * reserved2←0xffffffff
    write(Filedescriptor1, reserved2,4)
    reserved3← (unsigned int*)(buf+1056)
    lseek(Filedescriptor1,1056,SEEK_SET)
    * reserved3←0xffffffff
    write(Filedescriptor1, reserved3,4)
    reserved4← (unsigned int*)(buf+1060)
    lseek(Filedescriptor1,1060,SEEK_SET)
    * reserved4←0xffffffff
    write(Filedescriptor1, reserved4,4)
    reserved5← (unsigned int*)(buf+1064)
    lseek(Filedescriptor1,1064,SEEK_SET)
    * reserved5←0xffffffff
    write(Filedescriptor1, reserved5,4)
    checksum← (unsigned int*)(buf+1032)
    lseek(Filedescriptor1,1032,SEEK_SET)
    * checksum←0x7aaabbad
    write(Filedescriptor1, checksum,4)
    close(Filedescriptor1)
End
```

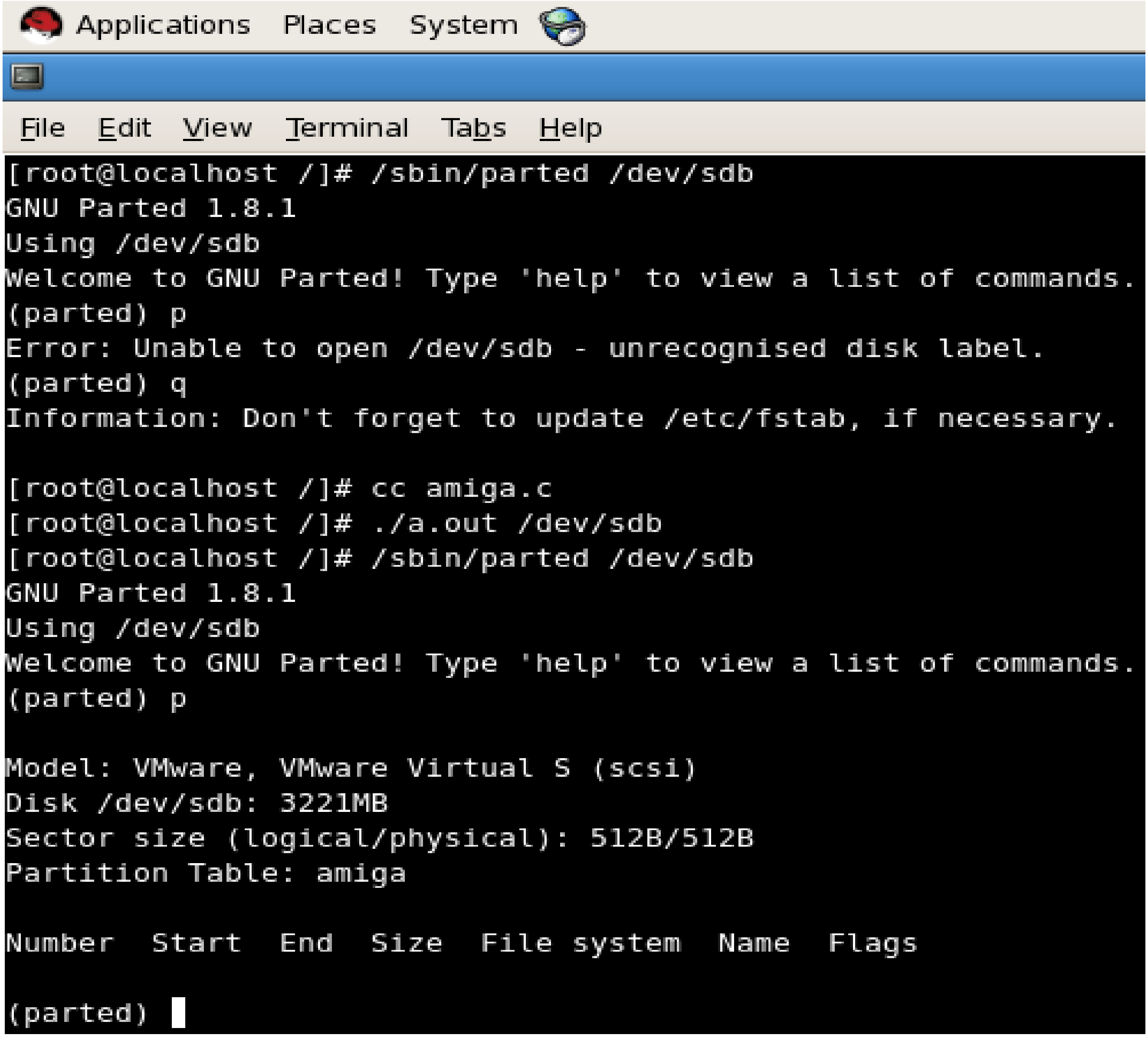


Fig 8.1: Screen shot for creating AMIGA Partition Table

9. BSD PARTITION TABLE (BSD) GENERATION ALGORITHM

Algorithm for BSD Partition Table

```
Input: Disk-name
Output: BSD Partition table

Algorithm for BSD Partition table
Begin
    Char buffer[512]
    Filedescriptor1←open(Disk-name, O_RDWR)
    lseek(Filedescriptor1, 64, SEEK_SET)
    label←(unsigned short *)(buffer +64)
    *label←0x82564557
    write(Filedescriptor1,label,4)
    close(Filedescriptor1)
End
```

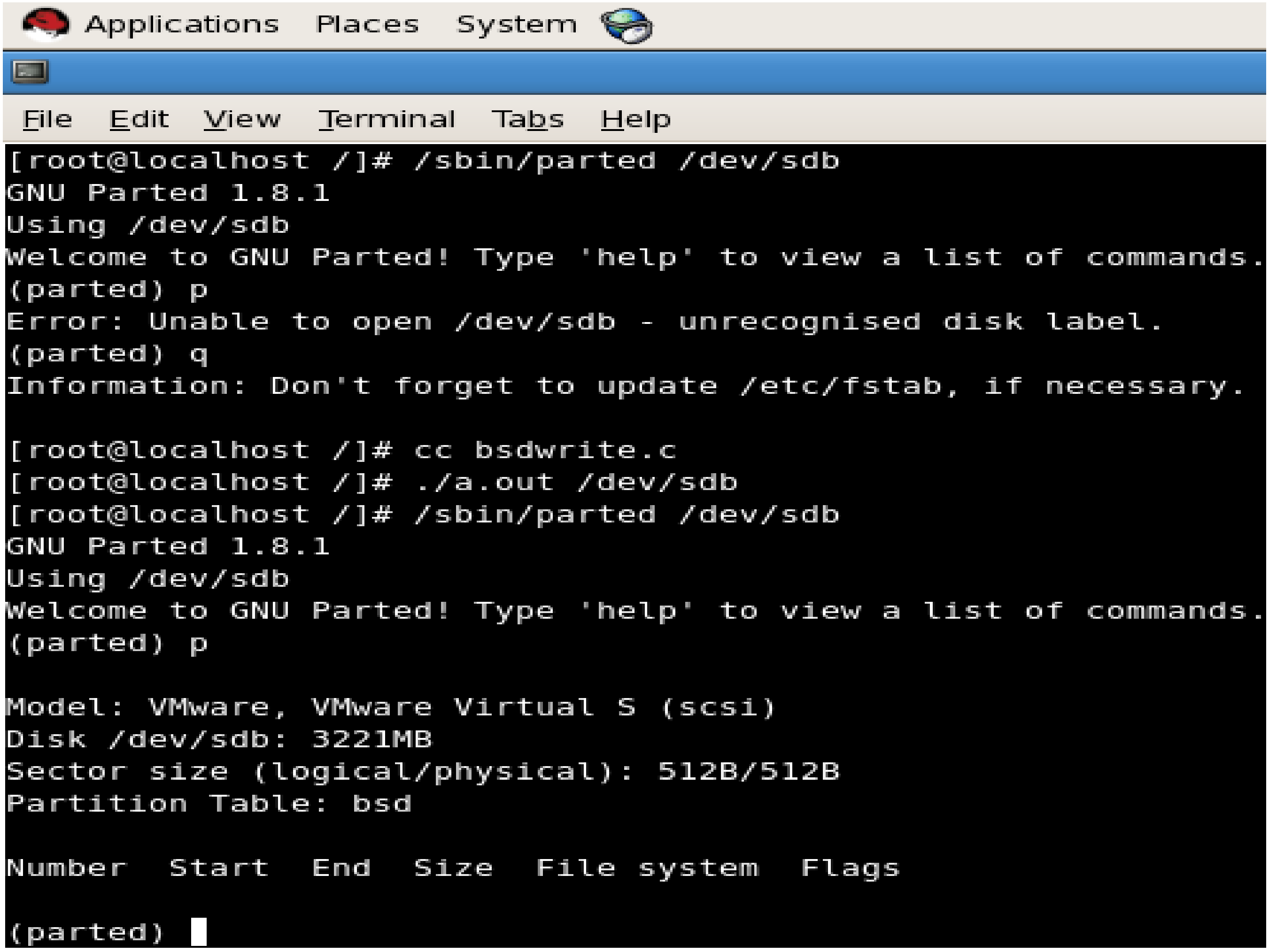
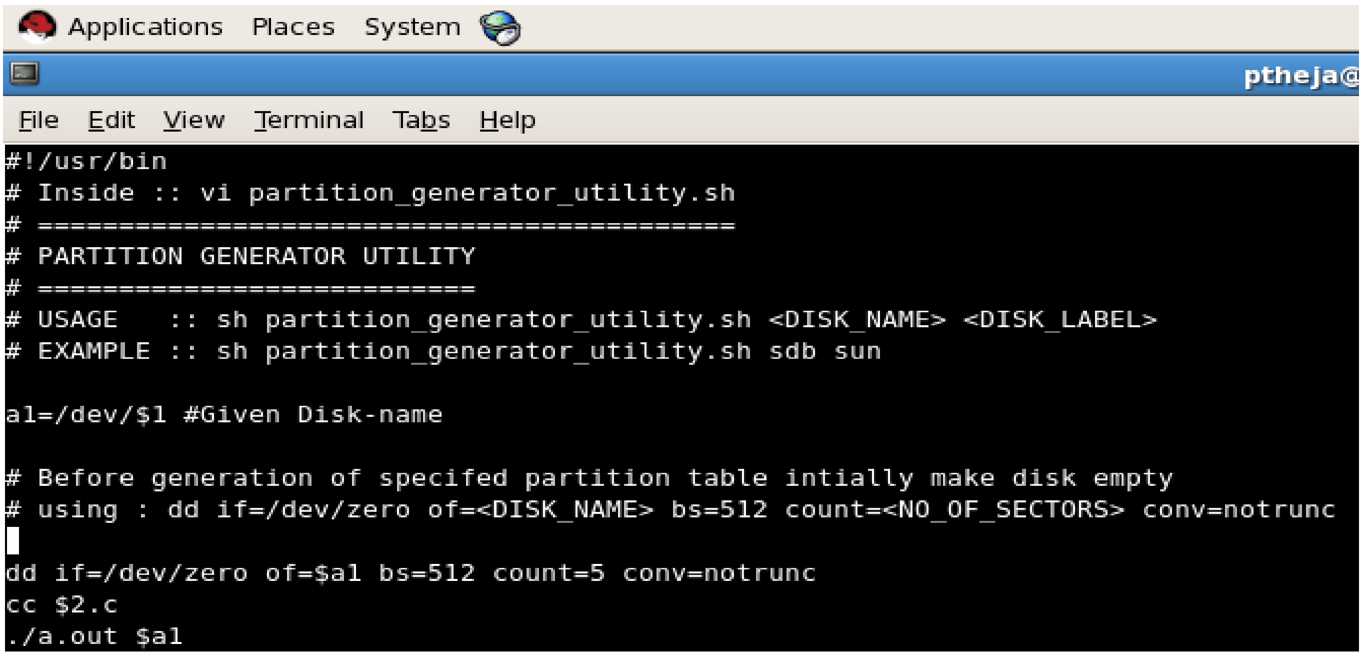


Fig 9.1: Screen shot for creating BSD Partition Table.

Fig 10.1: Screen shot for Partition Generator Utility.



10. CONCLUSION

We clearly stated algorithms of how to design different major partition tables with disk labels. In figure 10.1 we had shown the partition generator utility which generates the specified partition table on the specified disk and serves to the partition utility which is to be tested. We have performed analysis on all major partition tables and succeeded in generating them. In future enhancement, the analysis will be extended on remaining partition tables and make this utility strong enough of generating any specified partition table. The conclusion of the paper is we can improve the performance of a native partition utility, avoid data corruption and saves lot of time for testing by using partition generator utility.

11. REFERENCES

[1] Brain Carrier. (2005) ‘File System Forensic Analysis’, ISBN:0-32-126817-2 , pp-66-109.

[2] Intel. (2000) Extensible Firmware Interface Specification, Version 1.02, pp- 305-316.

[3] Michael Graves. (2004) A+ Guide to PC Hardware Maintenance and Repair, 1st ed., Cengage Learning, United States.

[4] Roderick W. Smith. (2000) The Multi-Boot Configuration Handbook, 1st ed., Que Publishing, United States.

[5] Roderick W. Smith, aka rodsbooks.com, (2011) [online]. Oberlin College. Available from: http://rodsbooks.com/gdisk. [Accessed 10 Jan 2011].

[6] HUANG Hui; LIU Zhenling. A Methodology for the Exploration of the Partition Table. IEEE, 2010: 978-1-4244-5265-1.

[7] CONDIE, T., CONWAY, N., ALVARO, P., AND HELLERSTEIN, J.MapReduce online. In NSDI (2010).