Coding for Hard-disk partition of Drive by Linux Block Device Driver

Amit Kr. Singh , 2 Asst. Prof. Navneet Kr. Pandey, 3 Asst. Prof. Diwakar Singh & 4 Prof. Shailendra Tahilyani

1(Mtech Scholar Digital Communication Dept, AKTU India) 2((ECE dept, BBDNITM/AKTU India), 3(ECE dept, BBDNITM/AKTU India)
4(HoD ECE dept, BBDNITM/AKTU India)
Corresponding Author: Amit Kr. Singh

Abstract: In this paper presenting coding required to understand the partition concept of Hard disk by linux device Driver.

Efficient block drivers are critical for its performance—and not just for explicit read and write in user applications. Modern systems with virtual memory work by shifting (hopefully) unneeded data to secondary storage, which is usually a disk drive. Block drivers are the conduit (midway) between core memory and secondary storage; therefore, they can be seen as making up part of the virtual memory subsystem.

In order to write the coding for partition of drive we have to go understand the following concept Master Boot Record: It is a special type of boot sector present at very beginning of partition drive. It hold information that how logical partition, extended partition, file system is being organized on that medium inspite of that MBR also contain executable code which act as loader function in installed OS (operating system). It usually does by passing control over to loader second stage. This MBR code usually referred to as BOOT LOADER.

The Structure of MBR (512 bytes) layout as follows

- The first part contain Bootstrap code area which is also called the Boot Loader which provide the OS to enter to door of RAM to load desired OS into the system. It contain space of 446 byte.
- Second part contain PARTITION ENTRY. There exist only four primary partition which had decided only since time of manufacturer we cannot create more than four primary partition instead of that we can make extended partition on that. Because of four primary partition only four entry is being there in structure of MBR. Each partition contain space of 16 byte. The information of 16 byte is shown in Appendix 2.
- After that there is boot signature. MBR Boot Signature is signature introduced in IBM PC for compatible fixed disk and removable drive. It generally used to define which primary sector is being active.
- The corresponding address along with decimal and hexadecimal is shown in figure below,

Appendix 1

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000h</td>
<td>Bootstrap code area</td>
<td>446</td>
</tr>
<tr>
<td>1BEh</td>
<td>Partition entry #1</td>
<td>16</td>
</tr>
<tr>
<td>1CEh</td>
<td>Partition entry #2</td>
<td>16</td>
</tr>
<tr>
<td>1DEh</td>
<td>Partition entry #3</td>
<td>16</td>
</tr>
<tr>
<td>1EEh</td>
<td>Partition entry #4</td>
<td>16</td>
</tr>
<tr>
<td>1FFh</td>
<td>Boot signature[a]</td>
<td>2</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td>512</td>
</tr>
</tbody>
</table>

Total size: 446 + 4x16 + 2 = 512
With above description of MBR now to define structure of partition entry.

**Partition Entry Scheme:**

With partition entry structure we see that the 16 byte is restricted for partition entry. The top 440 byte of MBR used to first piece of boot code which is loaded by BIOS to boot system from disk. The partition entry structure contain geometry of hard disk i.e. Head, Cylinder, Sector. This geometry describe the (abs_start_sec) from where desired disk started and number of sector in partition is described by (sec_in_part).

- The following structure shows the number of bit utilized for head (8 bit), cylinder (10 bit) and sector (6 bit).
- The HCS describe the starting address and end address of partition. The following partition also contain its active/inactive of particular drive through its 1st bit.

Usable hard disk size in bytes = (Number of heads or disks) * (Number of tracks per disk) * (Number of sectors per track) * (Number of bytes per sector, i.e. sector size)

### Appendix 2

#### Layout of one 16-byte partition entry (all multi-byte fields are little-endian)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0h</td>
<td>1 byte</td>
<td>status / physical drive (bit 7 set: active / bootable, old MBRs only accept 80h),00h: inactive, 02h–7fh: invalid) [a]</td>
</tr>
<tr>
<td>+1h</td>
<td>3 bytes</td>
<td>CHS address of first absolute sector in partition. The format is described by 3 bytes, see the next 3 rows.</td>
</tr>
<tr>
<td>+1h</td>
<td>1 byte</td>
<td>[h] [0] head[c]</td>
</tr>
<tr>
<td>+2h</td>
<td>1 byte</td>
<td>[c] [0] sector in bits 5–0; bits 7–6 are high bits of cylinder[c]</td>
</tr>
<tr>
<td>+3h</td>
<td>1 byte</td>
<td>[c] [0] bits 7–0 of cylinder[c]</td>
</tr>
<tr>
<td>+4h</td>
<td>1 byte</td>
<td>Partition type</td>
</tr>
<tr>
<td>+5h</td>
<td>3 byte</td>
<td>CHS address of last absolute sector in partition. The format is described by 3 bytes, see the next 3 rows.</td>
</tr>
<tr>
<td>+5h</td>
<td>1 byte</td>
<td>[h] [0] head[c]</td>
</tr>
<tr>
<td>+6h</td>
<td>1 byte</td>
<td>[c] [0] sector in bits 5–0; bits 7–6 are high bits of cylinder[c]</td>
</tr>
<tr>
<td>+7h</td>
<td>1 byte</td>
<td>[c] [0] bits 7–0 of cylinder</td>
</tr>
<tr>
<td>+8h</td>
<td>4 bytes</td>
<td>LBA of first absolute sector in the partition</td>
</tr>
<tr>
<td>+Ch</td>
<td>4 bytes</td>
<td>Number of sectors in partition[d]</td>
</tr>
</tbody>
</table>

With the help of MBR and partition structure the following code is being designed.

### Source code

```c
#include <stdio.h>
#include <sys/types.h>
```

DOI: 10.9790/2834-1401021216   www.iosrjournals.org
```c
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

#define SECTOR_SIZE 512
#define MBR_SIZE SECTOR_SIZE
#define MBR_DISK_SIGNATURE_OFFSET 440
#define MBR_DISK_SIGNATURE_SIZE 4
#define PARTITION_TABLE_OFFSET 446
#define PARTITION_ENTRY_SIZE 16 // sizeof(PartEntry)
#define PARTITION_TABLE_SIZE 64 // sizeof(PartTable)
#define MBR_SIGNATURE_OFFSET 510
#define MBR_SIGNATURE_SIZE 2
#define MBR_SIGNATURE 0xAA55
#define BR_SIZE SECTOR_SIZE
#define BR_SIGNATURE_OFFSET 510
#define BR_SIGNATURE_SIZE 2
#define BR_SIGNATURE 0xAA55

typedef struct {
    unsigned char boot_type; // 0x00 - Inactive; 0x80 - Active (Bootable)
    unsigned char start_head;
    unsigned char start_sec:6;
    unsigned char start_cyl_hi:2;
    unsigned char start_cyl;
    unsigned char part_type;
    unsigned char end_head;
    unsigned char end_sec:6;
    unsigned char end_cyl_hi:2;
    unsigned char end_cyl;
    unsigned long abs_start_sec;
    unsigned long sec_in_part;
} PartEntry;

typedef struct {
    unsigned char boot_code[MBR_DISK_SIGNATURE_OFFSET];
    unsigned long disk_signature;
    unsigned short pad;
    unsigned char pt[PARTITION_TABLE_SIZE];
    unsigned short signature;
} MBR;

void print_computed(unsigned long sector) {
    unsigned long heads, cyls, tracks, sectors;
    sectors = sector % 63 + 1 /* As indexed from 1 */;
    cyls = sector / 63;
    tracks = cyls / 255 + 1 /* As indexed from 1 */;
    heads = cyls % 255;
    printf("(%3d/%5d/%1d)", heads, cyls, sectors);
}

int main(int argc, char* argv[]) {
    char* dev_file = "/dev/sda";
    int fd, i, rd_val;
    MBR m;
    PartEntry *p = (PartEntry *)(m.pt);
    if(argc == 2) {
        dev_file = argv[1];
    }
    ...
if((fd = open(dev_file, O_RDONLY)) == -1) {
    fprintf(stderr, "Failed opening %s: ", dev_file);
    perror(" ");
    return1;
}
if((rd_val = read(fd, &m, sizeof(m))) != sizeof(m)) {
    fprintf(stderr, "Failed reading %s: ", dev_file);
    perror(" ");
    close(fd);
    return2;
}

for(i = 0; i < 4; i++) {
    printf("%d:%d (%3d/%4d/%2d) (%3d/%4d/%2d) %02X %10d %9d
", i+1, !!(p[i].boot_type& 0x80), p[i].start_head, 1 + ((p[i].start_cyl_hi<< 8) | p[i].start_cyl), p[i].start_sec, p[i].end_head, 1 + ((p[i].end_cyl_hi<< 8) | p[i].end_cyl), p[i].end_sec, p[i].part_type, p[i].abs_start_sec, p[i].sec_in_part);
}

return0;

Conclusion:
By the help of partitioning scheme we can make partition of any external drive even USB so as to load the desired OS and protect our kernel from external threat.

References
[23]. https://en.wikipedia.org/wiki/Master_boot_record#partition_table_entries