IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 14, Issue 1, Ser. II (Jan.-Feb. 2019), PP 12-16
www.iosrjournals.org

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

[1]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.*

Date of Submission: 13-02-2019 Date of acceptance:28-02-2019

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 diskdrive. 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 contaion 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 reffered to as BOOT LOADER.

The Structure of MBR (512 bytes)layout is 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 compatiable 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

Structure of a classical generic MBR				
Address		**Description**		**Size**
Hex	Dec			(bytes)
+000h	0	Bootstrap code area		446
+1BEh	446	Partition entry #1	*Partition table*	16
+1CEh	462	Partition entry #2	(for primary partitions)	16
+1DEh	478	Partition entry #3		16
+1EEh	494	Partition entry #4		16
+1FEh	510	55h	Boot signature[a]	2
+1FFh	511	AAh		
Total size: 446 + 4×16 + 2				**512**

Figure 1. [23]

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 throught 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)				
Offset (bytes)		Field length	Description	
+0h		1 byte	status / physical drive (bit 7 set: active / bootable, old MBRs only accept 80h),00h: inactive, 01h–7Fh: invalid)[a]	
+1h		3 bytes	CHS address of first absolute sector in partition. The format is described by 3 bytes, see the next 3 rows.	
	+1h	1 byte	$h_{7–0}$: x x x x x x x x	head[c]
	+2h	1 byte	$c_{9–8}$ $s_{5–0}$: x x x x x x x x	sector in bits 5–0; bits 7–6 are high bits of cylinder[c]
	+3h	1 byte	$c_{7–0}$: x x x x x x x x	bits 7–0 of cylinder[c]
+4h		1 byte	Partition type	
+5h		3 byte	CHS address of last absolute sector in partition. The format is described by 3 bytes, see the next 3 rows.	
	+5h	1 byte	$h_{7–0}$: x x x x x x x x	head[c]
	+6h	1 byte	$c_{9–8}$ $s_{5–0}$: x x x x x x x x	sector in bits 5–0; bits 7–6 are high bits of cylinder[c]
	+7h	1 byte	$c_{7–0}$: x x x x x x x x	bits 7–0 of cylinder
+8h		4 bytes	LBA of first absolute sector in the partition	
+Ch		4 bytes	Number of sectors in partition[d]	

Figure [23]

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

Source code

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

```
#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

typedefstruct{
   unsigned charboot_type; // 0x00 - Inactive; 0x80 - Active (Bootable)
   unsigned charstart_head;
   unsigned charstart_sec:6;
   unsigned charstart_cyl_hi:2;
   unsigned charstart_cyl;
   unsigned charpart_type;
   unsigned charend_head;
   unsigned charend_sec:6;
   unsigned charend_cyl_hi:2;
   unsigned charend_cyl;
   unsigned longabs_start_sec;
   unsigned longsec_in_part;
} PartEntry;

typedefstruct{
   unsigned charboot_code[MBR_DISK_SIGNATURE_OFFSET];
   unsigned longdisk_signature;
   unsigned shortpad;
   unsigned charpt[PARTITION_TABLE_SIZE];
   unsigned shortsignature;
} MBR;

voidprint_computed(unsigned longsector) {
   unsigned longheads, cyls, tracks, sectors;

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

intmain(intargc, char*argv[]) {
   char*dev_file = "/dev/sda";
   intfd, 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;
   }
   close(fd);
   printf("\nDOS type Partition Table of %s:\n", dev_file);
   printf("  B Start (H/C/S)   End (H/C/S) Type  StartSec    TotSec\n");
   for(i = 0; i< 4; i++) {
      printf("%d:%d (%3d/%4d/%2d) (%3d/%4d/%2d)  %02X %10d %9d\n",
         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);
   }
   printf("\nRe-computed Partition Table of %s:\n", dev_file);
   printf("  B Start (H/C/S)   End (H/C/S) Type  StartSec    TotSec\n");
   for(i = 0; i< 4; i++) {
      printf("%d:%d ", i + 1, !!(p[i].boot_type& 0x80));
      print_computed(p[i].abs_start_sec);
      printf(" ");
      print_computed(p[i].abs_start_sec + p[i].sec_in_part - 1);
      printf(" %02X %10d %9d\n", p[i].part_type,
         p[i].abs_start_sec, p[i].sec_in_part);
   }
   printf("\n");
   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

[1]. A. Rubini, Linux Device Drivers, O'Reilly &Associates,Sebastopol, Calif., 1998
[2]. T. Burke, M.A. Parenti, A. Wojtas. Writing Device Drivers: Tutorial and Reference, Digital Press, Boston,1995.
[3]. Linux Operating System Documentation,http://www.sunsite.unc.edu/pub/Linux
[4]. Robert Love, Linux Kernel Development, Second Edition, 2005
[5]. A. Rubini, "Dynamic Kernels: Modularize Device Drivers," Linux J.,Issue 23, Mar. 1996,
[6]. Y. Zhou, M.S. Li, "Research and implementing of real-time support of Linux kernel", Journal of computer research and development,Vo1.39, No. 4, April 2002.
[7]. P. Mantegazza, E. Bianchi, L. Dozio, S. Papachar-alambous, RTAI: Real Time Application Interface,Linux Journal, April 2000.
[8]. Chen lijun, "Understanding Linux kernel source code deeply"[M],Beijing: Posts & Telecom Press. 2002.
[9]. Linux Kernel http://www.kernel.org.
[10]. ELF specifications be downloaded from ftp://sunsite.unc.edu/pub/Linux/GCC/ELF.doc.tar.gz.
[11]. Murli. B. A, "Linux Device Driver coding for Pseudo device" , International Journal of computational Engineering Research , Pg No. 17-29.
[12]. M. Spear et al. Solving the starting problem: Device drivers as selfdescribingartifacts. In *Eurosys*, 2006.
[13]. M. M. Swift, M. Annamalai, B. N. Bershad, and H. M. Levy.Recovering device drivers.In *OSDI*, 2004.
[14]. L. Wittie. Laddie: The language for automated device drivers (ver 1). Technical Report 08-2, Bucknell CS-TR, 2008.
[15]. A. Silberschatz, P. B. Galvin, and G. Gagne.*Operating System Concepts*.John Wiley and Sons, eighth edition, 2009.
[16]. M. M. Swift, M. Annamalai, B. N. Bershad, and H. M. Levy.Recovering device drivers.In *OSDI*, 2004.

[17]. A. Kadav and M. Swift.Live migration of direct-access devices.*Operating Systems Review*, 43(3):95–104, 2009.
[18]. B. Leslie et al. User-level device drivers: Achieved performance. *Jour. Comp. Sci. and Tech.*, 2005.
[19]. Z. Li, S. Lu, S. Myagmar, and Y. Zhou. CP-Miner: A tool for finding copy-paste and related bugs in operating system code. In *OSDI*, 2004.
[20]. F. M. David et al. CuriOS: Improving reliability through operating system structure. In *OSDI*, 2008.
[21]. D. Williams, P. Reynolds, K. Walsh, E. G. Sirer, and F. B. Schneider. Device driver safety through a reference validation mechanism.In *OSDI*, 2008.
[22]. http://opensourceforu.com/2012/01/device-drivers-partitions-hard-disk/
[23]. https://en.wikipedia.org/wiki/Master_boot_record#partition_table_entries

IOSR Journal of Electronics and Communication Engineering (IOSR-JECE) is UGC approved Journal with Sl. No. 5016, Journal no. 49082.

Amit Kr. Singh. " Coding for Hard-disk partition of Drive by Linux Block Device Driver." IOSR Journal of Electronics and Communication Engineering (IOSR-JECE) 14.1 (2019): 12-16