Computer Science & Engineering Department
I. I. T. Kharagpur

Operating System: CS33007
3rd Year CSE: 5th Semester (*Autumn 2006 - 2007*)
Lecture XV (**File System Management**)

Goutam Biswas *Date:* 1st November, 2006

Ext2 and Ext3 File Systems

1. Linux native file systems:
   (a) *Minix* File System.
   (b) Extended File system, *Ext FS*.
   (c) Second Extended File System, *Ext2* (1994).
   (d) Third Extended File System, *Ext3*.
2. Some important features of Ext2 -
   (a) *Block size*: 1 KBytes to 4 KBytes - smaller block size reduces the *internal fragmentation*. Larger block size reduces the number of disk transfers.
   (b) Number of *inodes* can be chosen for a partition. This depends on the expected number of files in the file system.
   (c) The blocks in the disk partition are divided into *block groups*. Each group has its set of inodes and data blocks stored in adjacent tracks. This reduces the average seek time within the block.
   (d) The file system preallocate adjacent data blocks to a regular file before the actual use. This reduces file fragmentation.
   (e) Symbolic link names are stored in the inode itself provided the path name is restricted to a certain number of characters.
3. Ext3 file system - It is an enhancement of the Ext2 file system. Two main enhancements are:
   (a) Almost compatible with Ext2,
   (b) *Journaling* file system.
4. *Journaling File systems* -
   (a) It tries to avoid the time consuming file system check for inconsistency of the file system due to improper unmount of the file system (system failure).
   (b) Instead it looks into the disk area called *journal* that contains the most recent disk write operations.
   (c) First the block to be written is stored in the *journal* (committed to the journal), then the block is written in the file system (committed to the file system).

(d) Failures are divided into two case -

   i. Failure before committed to the journal: In this case the changes are lost, but the file system state is consistent.

   ii. Failure before committed to the file system: The new state is available in the journal.

   iii. Journaling is mostly done for the meta data blocks of file system e.g. *super block*, *group block descriptor*, *inodes*, *blocks for indirect addressing*, *data bitmap* and *inode bitmap*.

5. Ext2/Ext3 disk data structures:

```
$ df -h -t ext3
File system             Size  Used Avail Use% Mounted on
/dev/hda3               9.6G  3.3G  5.8G  36% /
/dev/hda6               4.0G   33M  3.7G   1% /backup
$
```

(a) Ext2/Ext3 partition -

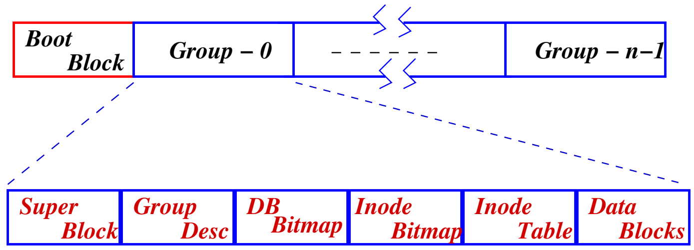

Figure 1: **Ext2 Layout**

(b) The first block of an Ext2 partition is reserved as boot block (it may not have boot code and data).

(c) Reading the boot block:

```
/**********************************************
 * Reads the boot block of the Ext3 file      *
 * readBootBlock.c                            *
 * The command line argument is the device    *
 * path name e.g. /dev/hda3                   *
 * *********************************************/
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
```

```
#define MAXBUF 1024

int main(int argc, char *argv[])
{
  int fd, i ;
  unsigned char buff[MAXBUF] ;

  if(argc < 2) {
printf("Device Path name missing\n") ;
exit(0) ;
 }
 fd = open(argv[1], O_RDONLY, 0) ;;
 printf("File Descriptor = %d\n", fd) ;
 read(fd, buff, MAXBUF) ;

  for(i=0; i<MAXBUF; ++i) {
     if(i%40 == 0) printf("\n") ;
     printf("%u ", buff[i]) ;
  }
  return 0 ;
}
```

(d) As mentioned earlier, each Ext2/Ext3 partition is divided into *block groups*. Each *block group* has a copy of the *super block* and the *group descriptor*.

(e) The data structure for the *super block* and the *group descriptor* are defined in `linux/ext3_fs.h`. Following code reads the super block from the *block group 0*. The super block structure is `struct ext3_super_block`.

```
/*************************************************
 * readSuperBlock.c                              *
 * Reads the super block of a partition          *
 * Pass the path name of the device as comand    *
 * line argument. The block-group number is hard- *
 * wired in the code                             *
 * **********************************************/

#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/unistd.h>
#include <linux/ext3_fs.h>

#define BLOCKS_PER_GROUP 32768
#define BLOCK_SIZE 4096
#define BLOCK_GROUP_NO 1
```

```
#define BOOT_REC_SIZE 1024
#define MAXBUF 1024
#define SBSIZE 256
int main(int argc, char *argv[])
{
struct ext3_super_block sb ;
int fd, i ;
long long unsigned offset, position ;
char buff[MAXBUF], *cp ;


if(argc < 2) {
printf("Device Path name missing\n") ;
exit(0) ;
}
fd = open(argv[1], O_RDONLY, 0) ;;
printf("File Descriptor = %d\n", fd) ;
if(BLOCK_GROUP_NO){
offset = (long long unsigned)
BLOCKS_PER_GROUP*BLOCK_SIZE*BLOCK_GROUP_NO ;
       position = llseek(fd, offset, 0) ;
}
else position = llseek(fd, BOOT_REC_SIZE, 0);
printf("Head position = %llu\n", position) ;
read(fd, buff, MAXBUF) ;

cp = (char *) &sb ;
for(i=0; i<SBSIZE; i++) cp[i] = buff[i] ;

printf("Total Inodes: %u\n", sb.s_inodes_count) ;
printf("\tInodes Size: %u\n", sb.s_inode_size) ;
printf("Total Blocks: %u\n", sb.s_blocks_count) ;
printf("\tFree Blocks: %u\n", sb.s_free_blocks_count) ;
printf("Free Inodes: %u\n", sb.s_free_inodes_count) ;
printf("\tFirst Data Block: %u\n", sb.s_first_data_block) ;
printf("Block Size: %u\n", sb.s_log_block_size) ;
printf("\tBlocks/Group: %u\n", sb.s_blocks_per_group) ;
printf("Inodes/Group: %u\n", sb.s_inodes_per_group) ;
printf("\tFirst non-reserve Inode: %u\n", sb.s_first_ino) ;
printf("Block group no.: %u\n", sb.s_block_group_nr) ;

return 0 ;
}
```

The output on my machine is

```
# a.out /dev/hda3
File Descriptor = 3
```

```
Head position = 1024
Total Inodes: 1281696
        Inodes Size: 128
Total Blocks: 2560359
        Free Blocks: 1667658
Free Inodes: 1128903
        First Data Block: 0
Block Size: 2
        Blocks/Group: 32768
Inodes/Group: 16224
        First non-reserve Inode: 11
Block group no.: 0
#
```

A bitmap block is of size *4096 bytes = 32768 bits*. Hence there are *32768 blocks per group*.

(f) Following code shows the group descriptor entries. The structure for the group descriptor is `struct ext3_group_desc`.

```
/**********************************************
 * readGroupDesc.c                            *
 * Reads the group descripter from a block    *
 * group.                                     *
 * Pass the path name of the device as a      *
 * command line argument                      *
 * *******************************************/
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/unistd.h>
#include <linux/ext3_fs.h>

#define BLOCKS_PER_GROUP 32768
#define BLOCK_SIZE 4096
#define BLOCK_GROUP_NO 0
#define MAXBUF 4096
#define GDSIZE 32
int main(int argc, char *argv[])
{
struct ext3_group_desc gd ;
int fd, i, j ;
char buff[MAXBUF], *cp ;
long long unsigned int offset, position ;

if( argc < 2) {
```

```
printf("Device Path name missing\n") ;
exit(0) ;
}
fd = open(argv[1], O_RDONLY, 0) ;
printf("File Descriptor = %d\n", fd) ;
offset = ((long long unsigned int)
BLOCK_GROUP_NO*BLOCKS_PER_GROUP + 1)*BLOCK_SIZE ;
position = llseek(fd, offset, SEEK_CUR) ;
printf("Head position = %llu\n", position) ;
read(fd, buff, MAXBUF) ;

cp = (char *) &gd ;
for(i=0; i<5; ++i) {
int k = i*GDSIZE ;
printf("Block group - %d\n", i) ;
for(j=0; j<GDSIZE; j++) cp[j] = buff[j+k] ;

printf("\tBlock no of group bitmap: %u\n", gd.bg_block_bitmap) ;
printf("\tBlock no group inode bitmap: %u\n",
gd.bg_inode_bitmap) ;
printf("\tBlock no of group inode table: %u\n",
gd.bg_inode_table) ;
printf("\tNumber of free blocks in group: %u\n",
         gd.bg_free_blocks_count) ;
printf("\tNumber of free inodes in group: %u\n",
gd.bg_free_inodes_count) ;
printf("\tNumber of directories in group: %u\n",
gd.bg_used_dirs_count) ;
}
return 0 ;
}
```

The output on my machine is

```
# a.out /dev/hda3
File Descriptor = 3
Head position = 4096
Block group - 0
        Block no of group bitmap: 2
        Block no group inode bitmap: 3
        Block no of group inode table: 4
        Number of free blocks in group: 0
        Number of free inodes in group: 16208
        Number of directories in group: 2
Block group - 1
        Block no of group bitmap: 32770
        Block no group inode bitmap: 32771
        Block no of group inode table: 32772
```

```
        Number of free blocks in group: 15586
        Number of free inodes in group: 14342
        Number of directories in group: 101
Block group - 2
.......................
Block group - 4
        Block no of group bitmap: 131072
        Block no group inode bitmap: 131073
        Block no of group inode table: 131076
        Number of free blocks in group: 31933
        Number of free inodes in group: 8503
        Number of directories in group: 8
#
```

(g) On my machine -

```
# ls -lai / | less
2 drwxr-xr-x   22 root root 4096 Sep 19 04:26 .
2 drwxr-xr-x   22 root root 4096 Sep 19 04:26 ..
```

Though the inode number for the root is shown to be 2, it is actually 1. We look at the inode table using the following program. The Ext3 disk inode structure is defined in `ext3_fs.h`. It is *ext3_inode*.

```
/***********************************************
 * readInodeTable.c                            *
 * Reads the inode tables from a block         *
 * group.                                      *
 * Pass the path name of the device as a       *
 * command line argument                       *
 * *********************************************/
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/unistd.h>
#include <linux/ext3_fs.h>

#define BLOCKS_PER_GROUP 32768
#define BLOCK_SIZE 4096
#define BLOCK_GROUP_NO 0
#define MAXBUF 4096
#define INODE_SIZE 128
int main(int argc, char *argv[])
{
struct ext3_inode inode ;
int fd, i, j ;
```

```
char buff[MAXBUF], *cp ;
long long unsigned offset, position ;

if( argc < 2) {
printf("Device Path name missing\n") ;
exit(0) ;
}
fd = open(argv[1], O_RDONLY, 0) ;
printf("File Descriptor = %d\n", fd) ;
offset = ((long long unsigned)
BLOCK_GROUP_NO*BLOCKS_PER_GROUP + 4)*BLOCK_SIZE ;
position = llseek(fd, offset, 0) ;
printf("Head position = %llu\n", position) ;
read(fd, buff, MAXBUF) ;

cp = (char *) &inode ;
for(i=0; i<2; ++i) {
int k = i*INODE_SIZE, l = 0 ;
printf("Inode - %d\n", i) ;
for(j=0; j<INODE_SIZE; j++) cp[j] = buff[j+k] ;

printf("\tFile type & access rights: %x\n", inode.i_mode) ;
printf("\tFile length: %u bytes\n", inode.i_size) ;
printf("\tHard links count: %u\n", inode.i_links_count) ;
printf("\tData block count: %u\n", inode.i_blocks) ;
printf("\tPointers to data blocks:\n") ;
while(inode.i_block[l] != 0)
printf("\t\t%u\n", inode.i_block[l++]) ;

}
return 0 ;
}
```

The output on my machine is

```
# a.out /dev/hda3
File Descriptor = 3
Head position = 16384
Inode - 0
        File type & access rights: 0
        File length: 0 bytes
        Hard links count: 0
        Data block count: 0
        Pointers to data blocks:
Inode - 1
        File type & access rights: 41ed
        File length: 4096 bytes
        Hard links count: 22
```

```
        Data block count: 8
        Pointers to data blocks:
                511
#
```

Note that

i. The file type and access rights **0x41ed** is `drwxr-xr-x`.

0100	000	111	101	101

ii. The size is 4096 bytes $= 8 \times 512$ (sector size).

iii. Number of hard link count is 22.

iv. The logical block number is 511.

(h) Let us look into the data block corresponding to the root directory. The directory entry structure is defined in `ext3_fs.h`, and it is `ext3_dir_entry_2`. The following code accesses the data block.

```
/*********************************************
 * readRootDirData.c                         *
 * Reads the root directory data.            *
 * Pass the path name of the device as a     *
 * command line argument                     *
 * ******************************************/
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/unistd.h>
#include <linux/ext3_fs.h>

#define ROOT_DATA_BLOCK_NO 511
#define BLOCK_SIZE 4096
#define MAXBUF 4096
#define DIR_ENT_SIZE 264
#define INT_SIZE  4

int main(int argc, char *argv[])
{
struct ext3_dir_entry_2 dirEnt ;
int fd, j, entIndex ;
char buff[MAXBUF], *cp ;
long long unsigned offset, position ;
short unsigned int *sip ;

if( argc < 2) {
printf("Device Path name missing\n") ;
exit(0) ;
```

```
}

fd = open(argv[1], O_RDONLY, 0) ;
printf("File Descriptor = %d\n", fd) ;
offset = (long long unsigned) ROOT_DATA_BLOCK_NO*BLOCK_SIZE ;
position = llseek(fd, offset, 0) ;
printf("Head position = %llu\n", position) ;
read(fd, buff, MAXBUF) ;

entIndex = 0 ;
cp = (char *) &dirEnt ;
printf("   inode  rec.length  name.length  filename\n") ;
while(1) {
sip = (short unsigned int *) (buff + entIndex + INT_SIZE) ;
        if(*sip == 0 || *sip > DIR_ENT_SIZE) break ;
         for(j=0; j<*sip; j++) cp[j] = buff[j+entIndex] ;
cp[j] = '\0' ;
entIndex += *sip ;
printf("%8u ....... %3hu ........ %3hhu  %s\n",
dirEnt.inode, dirEnt.rec_len, dirEnt.name_len,
dirEnt.name) ;
}
return 0 ;
}
```

(i) In my machine there is a *symbolic link* `alphaBeta` with the inode number **15**. We access the inode data structure for $15 - 1 = 14$. We get the following output.

```
# cc readSymLinkInode.c
# a.out /dev/hda3
File Descriptor = 3
Head position = 16384
Inode - 15
        File type & access rights: a1ff
        File length: 16 bytes
        Hard links count: 1
        Data block count: 0
        Pointers to data blocks:
                0x756e696c
                0x63672d78
                0x61742e63
                0x7a672e72
#
```

The 'a' in the file type indicates that it is a symbolic link. Look at the pointer data in hex (it is a little-endian system) -

6c 69 6e 75 78 2d 67 63 63 2e 74 61 72 2e 67 7a

This is the ASCII code for "`linux-gcc.tar.gz`". The inode itself stores the target file name of the symbolic link.

(j) There is a regular file `iTab.c` in my root directory

```
22 -rw-r--r-- 1 root root 756 Nov  9  2004 /iTab.c
```

The metadata fron the inode $22 - 1 = 21$ is -

```
# cc -Wall readRegFileInode.c
# a.out /dev/hda3
File Descriptor = 3
Head position = 16384
Inode - 22
        File type & access rights: 81a4
        File length: 756 bytes
        Hard links count: 1
        Data block count: 8
        Pointers to data blocks:
                11856
#
```

The exact file size is stored in the inode. There is no special EOF mark. We take an ASCII dump of the raw data block

```
# cc -Wall readDataBloc.c
# a.out /dev/hda3
File Descriptor = 3
Head position = 48562176
ASCII dump of the data block:

23 69 6E 63 6C 75 64 65 20 3C 66 63 6E 74 6C 2E 68 3E A 23
69 6E 63 6C 75 64 65 20 3C 6C 69 6E 75 78 2F 65 78 74 33 5F
66 73 2E 68 3E A A A 23 64 65 66 69 6E 65 20 42 4C 4F 43
4B 5F 53 49 5A 45 20 34 30 39 36 A 23 64 65 66
...............
5F 62 6C 6F 63 6B 5B 69 5D 29 20 3B A 7D 9 A 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

The corresponding file is

```
#include <fcntl.h>
#include <linux/ext3_fs.h>
...................
   printf("Data Block %u: %u\n", i, in.i_block[i]) ;
}
```

6. Creating Ext2 or Ext3 file system:

   (a) The command `mke2fs` creates a Ext2 file system. The same command with '`-j`' as the option creates a journal for Ext3 file system.

   (b) Different parameters have their default values and also can be specified as command line arguments e.g. the block size, number of inodes etc.

   (c) The essential steps are

      i. Initialize the superblock and the group descriptors.

      ii. Reserve space for the superblock, group descriptor, inode table, inode bit map and data block bit map in each group.

      iii. Initialize the bit maps.

      iv. Create the `root` directory and the `lost+found` directory (lost and found defective blocks). Update the bit maps.

   (d) Create an Ext2 file system on a floppy and mount it.

```
# mke2fs /dev/fd0
mke2fs 1.27 (8-Mar-2002)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
184 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
1 block group
8192 blocks per group, 8192 fragments per group
184 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 22 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
#
```

   (e) We mount the floppy disc and create a file -

```
# mkdir mnt
# mount /dev/fd0 ./mnt
# cd mnt
# ls -lai
total 17
      2 drwxr-xr-x  3 root     root   1024 Sep 24 23:41 .
  17837 drwxr-xr-x 24 goutam   users  4096 Sep 24 23:36 ..
     11 drwx------  2 root     root   12288 Sep 24 23:32 lost+found
# vi test.c
# cc -Wall test.c
```

```
# ls -lai
total 30
      2 drwxr-xr-x   3 root    root    1024 Sep 24 23:43 .
  17837 drwxr-xr-x  24 goutam  users   4096 Sep 24 23:36 ..
     12 -rwxr-xr-x   1 root    root   11377 Sep 24 23:43 a.out
     11 drwx------   2 root    root   12288 Sep 24 23:32 lost+found
     13 -rw-r--r--   1 root    root      89 Sep 24 23:43 test.c
# a.out
On the mounted file system
#
```

(f) Un-mounting the file system -

```
# umount /dev/fd0
# cd mnt
# ls
#
```

7. Some more code

```
/***********************************************
 * readSymLinkInode.c                          *
 * Reads the inode of a symbolic link          *
 * Pass the path name of the device as a       *
 * command line argument                       *
 * *********************************************/
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/unistd.h>
#include <linux/ext3_fs.h>

#define BLOCKS_PER_GROUP 32768
#define BLOCK_SIZE 4096
#define BLOCK_GROUP_NO 0
#define MAXBUF 4096
#define INODE_SIZE 128
#define SYM_LINK_INODE_NO 14
int main(int argc, char *argv[])
{
struct ext3_inode inode ;
int fd, j, k, l = 0 ;
char buff[MAXBUF], *cp ;
long long unsigned offset, position ;
```

```
if( argc < 2) {
printf("Device Path name missing\n") ;
exit(0) ;
}
fd = open(argv[1], O_RDONLY, 0) ;
printf("File Descriptor = %d\n", fd) ;
offset = ((long long unsigned)
BLOCK_GROUP_NO*BLOCKS_PER_GROUP + 4)*BLOCK_SIZE ;
position = llseek(fd, offset, 0) ;
printf("Head position = %llu\n", position) ;
read(fd, buff, MAXBUF) ;

cp = (char *) &inode ;
k = SYM_LINK_INODE_NO*INODE_SIZE ;
printf("Inode - %d\n", SYM_LINK_INODE_NO+1) ;
for(j=0; j<INODE_SIZE; j++) cp[j] = buff[j+k] ;

printf("\tFile type & access rights: %x\n", inode.i_mode) ;
printf("\tFile length: %u bytes\n", inode.i_size) ;
printf("\tHard links count: %u\n", inode.i_links_count) ;
printf("\tData block count: %u\n", inode.i_blocks) ;
printf("\tPointers to data blocks:\n") ;
while(inode.i_block[l] != 0)
printf("\t\t0x%x\n", inode.i_block[l++]) ;

return 0 ;
}

/**********************************************
 * readRegFileInode.c                         *
 * Reads the inode of a regular file          *
 * Pass the path name of the device as a      *
 * command line argument                      *
 * *******************************************/
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/unistd.h>
#include <linux/ext3_fs.h>

#define BLOCKS_PER_GROUP 32768
#define BLOCK_SIZE 4096
#define BLOCK_GROUP_NO 0
#define MAXBUF 4096
```

```
#define INODE_SIZE 128
#define INODE_NO 21
int main(int argc, char *argv[])
{
struct ext3_inode inode ;
int fd, j, k, l = 0 ;
char buff[MAXBUF], *cp ;
long long unsigned offset, position ;

if( argc < 2) {
printf("Device Path name missing\n") ;
exit(0) ;
}
fd = open(argv[1], O_RDONLY, 0) ;
printf("File Descriptor = %d\n", fd) ;
offset = ((long long unsigned)
BLOCK_GROUP_NO*BLOCKS_PER_GROUP + 4)*BLOCK_SIZE ;
position = llseek(fd, offset, 0) ;
printf("Head position = %llu\n", position) ;
read(fd, buff, MAXBUF) ;

cp = (char *) &inode ;
k = INODE_NO*INODE_SIZE ;
printf("Inode - %d\n", INODE_NO+1) ;
for(j=0; j<INODE_SIZE; j++) cp[j] = buff[j+k] ;

printf("\tFile type & access rights: %x\n", inode.i_mode) ;
printf("\tFile length: %u bytes\n", inode.i_size) ;
printf("\tHard links count: %u\n", inode.i_links_count) ;
printf("\tData block count: %u\n", inode.i_blocks) ;
printf("\tPointers to data blocks:\n") ;
while(inode.i_block[l] != 0)
printf("\t\t%u\n", inode.i_block[l++]) ;

return 0 ;
}

/***********************************************
 * Reads a data block of the Ext3 file         *
 * readDataBlock.c                             *
 * The command line argument is the device     *
 * path name e.g. /dev/hda3                    *
 * *********************************************/
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>
#include <unistd.h>

#define BLOCK_SIZE 4096
#define BLOCK_NUMBER  11856
#define MAXBUF 4096

int main(int argc, char *argv[])
{
  int fd, i ;
  unsigned char buff[MAXBUF] ;
long long unsigned offset, position ;

  if(argc < 2) {
printf("Device Path name missing\n") ;
exit(0) ;
 }
 fd = open(argv[1], O_RDONLY, 0) ;;
 printf("File Descriptor = %d\n", fd) ;
 offset = (long long unsigned) BLOCK_NUMBER*BLOCK_SIZE ;
 position = llseek(fd, offset, SEEK_CUR) ;
         printf("Head position = %llu\n", position) ;

 read(fd, buff, MAXBUF) ;

 printf("ASCII dump of the data block:\n") ;
  for(i=0; i<MAXBUF; ++i) {
     if(i%20 == 0) printf("\n") ;
     printf("%X ", buff[i]) ;
  }
  return 0 ;
}
```

References

[1] Bovet D. P. & Cesati M., *Understanding the Linux Kernel*, SPD O'Reilly, ISBN 81 7366 589 3, 2004.

[2] Bach M. J., *The Design of the Unix Operating System*, PHI, 0 87692 516 6, 1988.