Project

1. Understanding Relevant Code

Refer to operating system (os.html) and the links given in reference section of each page.

LInk to the book

2. Design

There are two ways in which immediate files can be implemented in the minix operating system -
Static : In this approach, the maximum file size is specified at creation and you can't change the type of file once created. In case the immediate file size exceeds the specified size, it will report an error. This is can be implemented by creating a new file system.
Dynamic : A file is created as an immediate file and if size exceeds specified size, it becomes a regular file. To implement this approach, we need to make changes in the existing file system.
Implementation in Dynamic Approach: To include immediate file system in an existing file system, the following sections are involved - *Data structures representing files Virtual File System Message Formats Minix File System* : in the funtions of the regular file system of minix, there is almost no change in create, open and close functions. The changes in read, write, delete are made so that if the file is immediate (which can be deduced by comparing size), then there is the content is to be accessed from the inode.

3. Tutorials for Basic Understanding

1. http://homepages.cs.ncl.ac.uk/nick.cook/csc2025/minix/ (http://homepages.cs.ncl.ac.uk/nick.cook/csc2025/minix/)
2. http://homepages.cs.ncl.ac.uk/nick.cook/csc2025/minix/familiarisation.html (http://homepages.cs.ncl.ac.uk/nick.cook/csc2025/minix/familiarisation.html)
3. https://diuf.unifr.ch/pai/education/2004_2005/courses/OS/04_Minix_01 (https://diuf.unifr.ch/pai/education/2004_2005/courses/OS/04_Minix_01)

4. Installation and setting up the system

- Follow this (http://sudevambadi.me/MinixMajor/ref/install.html) link to install minix in virtual box
- Vi editor can be used to edit the code but if you need any other editior you can install it using `pkgin` or `pkgin_cd` . Refer the link above
- If installation is to be done by `pkgin_cd` , you must install virtual box extensions from here (https://www.virtualbox.org/wiki/Downloads)
- You might need to edit the `rc.conf` , if `pkgin_cd` is not working.
- To transfer your code to eclipse (most preferred), follow this (http://wiki.minix3.org/doku.php?id=developersguide:eclipsetutorial) link

5. Algorithm

```
1: procedure FS_READWRITE_IMMED
2:     is_immediate = 0 (0 if regular, 1 if immediate)
3:     mode = inode.i_mode (regualar or immediate)
4:     pos = req_msg.LSEEK_POS (lseek position)
5:     nrbytes = req_msg.NBYTES (number if bytes to be read or written)
6:     rw_flag = req_msg.m_type (READING or WRITING)
7:     f_size = inode.f_size
8:     immed_buff[] = "" (temporary array)
9:     if mode == I_IMMEDIATE then
10:        if rw_flag == WRITING then
11:            if (f_size + nrbytes) >MAX_IMMEDSIZE then
12:                if (pos + nrbytes) <MAX_IMMEDSIZE then
13:                    i_immediate = 1
14:                else
15:                    /** Shift from Immediate to regular **/
16:                    Copy the content of zone to immed_buff array
17:                    Mark all zones as empty_zone
18:                    Change inode.size to zero
19:                    Change update_time of inode
20:                    Mark the inode dirty
21:                    Request a new block, bp
22:                    Copy the immed_buff content to bp.data field
23:                    Mark the block, bp dirty
24:                    Update pos and f_size
25:                    inode.i_mode = REGULAR
26:                    is_immediate = 0 // file is no more immediate
27:                end if
28:            else
29:                is_immediate = 1
30:            end if
31:        else
32:            /** reading no change required **/
33:            is_immediate = 1
34:        end if
35:    end if
36:    if is_immediate == 1 then // the file is still immediate
37:        Calculate the zone_position in the disk
38:        Call system_read or system_write function with zone_pos as argument
39:    end if
40: end procedure
```

6. Final Implementation - Coding

File System Basic Structure

The `/usr/src/servers/mfs` directory contains the source code for FS in minx3.2 operating system. Some of the important files in mfs are `main.c, in- ode.h, open.c, write.c, buf.h, super.h, super.c` etc. The main function of each file in mfs are listed below:

- buf.h - Defines the block cache. It contains a union named fsdata u with following attributes:
    - b data[MAX BLOCK SIZE], a character array, containing ordinary user data.
    -
    - b dir[NR DIR ENTRIES(MAX BLOCK SIZE)] - directory block
    - b bitmap[[FS BITMAP CHUNKS(MAX BLOCK SIZE)]] - bitmap block
    - direct and indirect inode blocks
- cache.c - FS has a buffer cache to reduce the number of disk accesses. File contains 9 procedures, few of them are listed below.
    - get block - Fetch a block for reading/writing.
    - put block - Return a block
    - rw block - Transfer block between disk and cache
    - free zone - If file is deleted, free the zone
- const.h - Defines constants, like flags, table size that will be used in the file system. Few constants are, IN CLEAN, IN DIRTY, ATIME, CTIME etc
- fs.h - Master header for FS, includes all header files needed by the MFS source files
- glo.h - Defines all the global variables. Few examples of global variables are fs m in, fs m out, err code, fs dev, user path[PATH MAX] etc.
- inode.h - Contains the stucucture for inode and the inode table as in- ode[NR INODES]
- inode.c - Contains functions which manages the inode table. The func- tions are get inode(), put inode(), rw inode(), alloc inode() etc
- main.c - Contains the main routiene of the file system. The main loop does three activities
    - get work(&fs_m_in) - Gets a new work
    - Processes the work i.e. selects the fuction to be called to complete the work and calls it from the table of function pointers.
    - reply(src, &fs_m_out) - Sends a reply
- open.c - Contains the codes for six system calls:open, close, mknode, mkdir, close, lseek
- proto.h - Lists all function prototypes for all functions used in MFS
- read.c - All functions that are used for reading or writing are present in read.c. Some of the functions include, fs readwrite, rw chunk, read map etc.
- super.h - Contains the superblock table. Super block holds information about inode bitmap, zone bitmaps, inodes etc.
- super.c - Handles the superblock table and other related data structures like zone bitmap, inode bitmap etc. Major functions in this file are: al- loc bit(), free bit(), get super(), read super() etc.
- table.c - Contains the table that map system call numbers onto the rou- tines.
- write.c - Contains files that are not in read.c but are necessary for writing in a file. Most important functions in write.c are write map, clear zone, new block and zero block

Data Structures Involved

Inode

we can see that there are 7 zones of 4 bytes each, an indirect zone of size 4 bytes, a double indirect node of size 4 bytes and an unused space of 4 bytes. Each zone points to a disk block where actual data gets stored. According to definition of immediate files we had to find a space in the inode where we can store the immediate files. These zones are apt place to store the immediate files because no data which is critical to the file is being affected. We calcualted maximum size of immediate files as 40 bytes by adding up sizes of all zones, indirect zones and unused space.

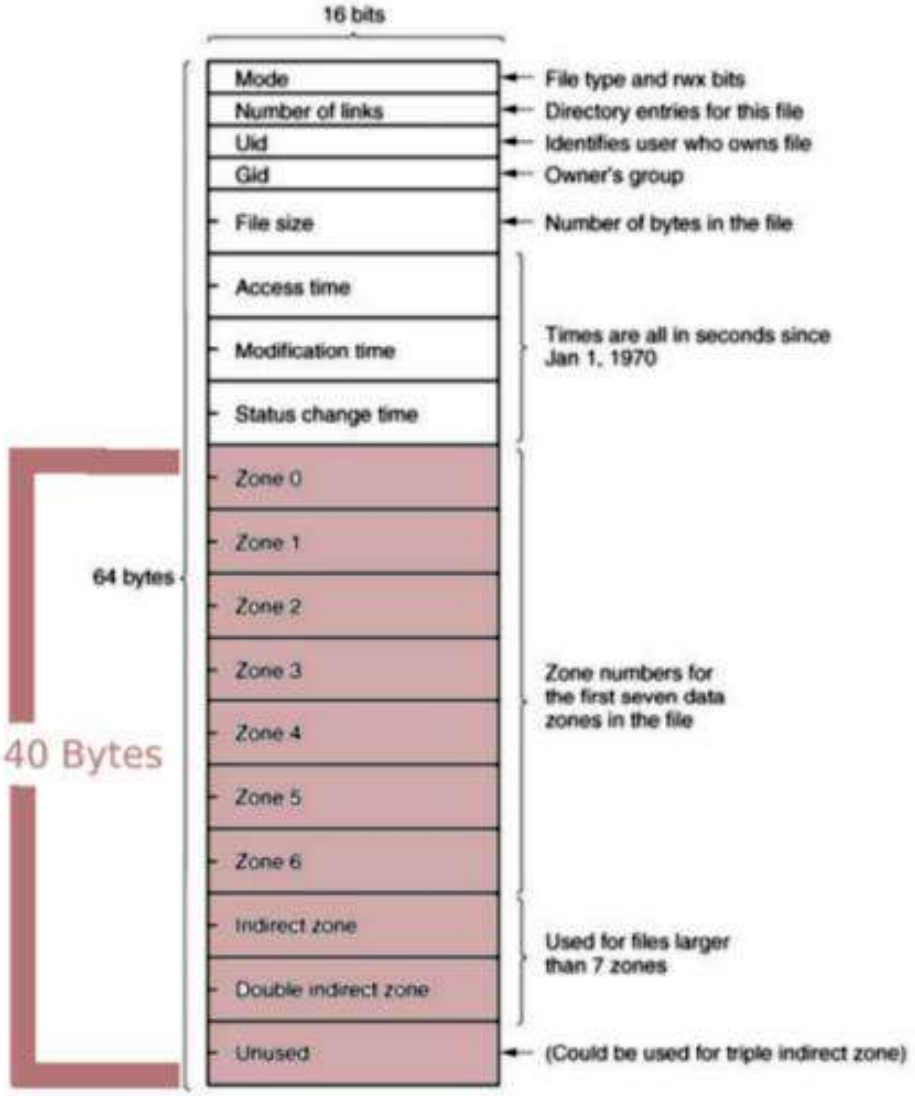

Figure 3.1: Inode Structure

```
max size = zn sz X 7 + ud sz + idt sz + ddt sz
max size = 4 X 7 + 4 + 4 + 4 = 40bytes
```

buffer or block cache

This is a union of different types of blocks in the disk. Eg. normal data block, directory block, inode block, bitmap block etc. The design of buffer or block cache is given below.

```
union fsdata_u {
  /* ordinary user data */
  char b__data[_MAX_BLOCK_SIZE];
  /* directory block */
  struct direct b__dir[NR_DIR_ENTRIES(_MAX_BLOCK_SIZE)];
  /* V1 indirect block */
  zone1_t b__v1_ind[V1_INDIRECTS];
  /* V2 indirect block */
  zone_t b__v2_ind[V2_INDIRECTS(_MAX_BLOCK_SIZE)];
  /* V1 inode block */
  d1_inode b__v1_ino[V1_INODES_PER_BLOCK];
  /* V2 inode block */
  d2_inode b__v2_ino[V2_INODES_PER_BLOCK(_MAX_BLOCK_SIZE)];
  /* bit map block */
  bitchunk_t b__bitmap[FS_BITMAP_CHUNKS(_MAX_BLOCK_SIZE)];
};
```

b_data array is used to cache the data which is stored in the disk block, all the modifications by the user are done here and then the data is written back to the disk. b data(b) is a macro which returns the pointer to the first byte of b data array.

message

```
typedef struct {
  int m_source;         // message source
  int m_type;           // message type
  union {
        mess_1 m_m1;
        mess_2 m_m2;
        mess_3 m_m3;
        mess_4 m_m4;
        mess_5 m_m5;
        mess_7 m_m7;
        mess_8 m_m8;
  } m_u;
} message;
```

mess 1, mess 2, mess 3, . . . are different message types. In MFS, global variables, fs m in and fs m out, of type message are used to send and recieve messages from various servers like VFS. Following system calls are used for message passing: echo, notify, sendrec, receive, send.

Files Involved

List of all files in which codes are added or deleted.

- /src/include/const.h - A new flag I IMMEDIATE is created to sup- port immediate files
- src/sys/lib/libsa/minixfs3.h - A new flag I IMMEDIATE is created to support immediate files
- /src/servers/vfs/open.c - When O CREATE flag is set. Set the file mode as immediate instead of regular. It will have size zero.

```
  /* In function common_open */
if (oflags & O_CREAT) {
// we have removed I_REGULAR mode
      omode = I_IMMEDIATE | (omode & ALL_PERMS & fp->fp_umask)
  ;
vp = new_node(&resolve, oflags, omode);
r = err_code;
if (r == OK) exist = FALSE; /* file created */
else if (r != EEXIST) {   /* other error */
  if (vp) unlock_vnode(vp);
  unlock_filp(filp);
  return(r);
}
```

- src/sys/sys/stat.h - Following additions are done in this file:
  - S_IFIMMED - macro was defined, similar to regular files

    ```
    #define S_IFIMMED 0130000
    ```
  - S_IFIMMED - macro redifined for easier usage, similar to regular files.

    ```
    #define S_IFIMMED S_IFIMMED
    ```
  - S_ISIMMED(m) - macro defined which checks whether a files is immediate or not (similar to regular files)

    ```
    #define S_ISIMMED(m) ( ( (m) & S_IFMT ) == S_IFIMMED )
    ```
  - There are 114 more files where we have located S ISREG(m) and added S ISIMMED(m) also in the code, because regular files and immediate files have same functionalities except that immediate files are stored in inode and regular files are stored in disk blocks pointed by zones in the inode. Few of the files are listed below:
    - /servers/vfs/select.c
    - /servers/vfs/link.c
    - /servers/vfs/read.c
    - /commands/grep/mmfile.c ... etc
  - /src/severs/mfs/read.c - Major changes/addition for implementation of immediate file system is done in this file, in the fs readwrite() function

```
/** in File read.c **/
/** start **/
cum_io = 0;
char immed_buff[41];
if ((rip->i_mode & I_TYPE) == I_IMMEDIATE) {
  int is_immediate;
  int i;
  if (rw_flag == WRITING) {
   if ((f_size + nrbytes) > 40) {
```

```
     if (position == 0 && nrbytes <= 40) {
      is_immediate = 1;
     } else {
      register struct buf *bp;
      for (i = 0; i < f_size; i++) {
        immed_buff[i] = *((char *) rip->i_zone)+i);
      }

      for (i = 0; i < V2_NR_TZONES; i++) {
        rip->i_zone[i] = NO_ZONE;
      }
      rip->i_size = 0;
      rip->i_update = ATIME | CTIME | MTIME;
      IN_MARKDIRTY(rip);

      bp = new_block(rip, (off_t) 0);

      if (bp == NULL)
        panic("error");

      for (i = 0; i < f_size; i++) {
        b_data(bp)[i] = immed_buff[i];
      }

      MARKDIRTY(bp);
      put_block(bp, PARTIAL_DATA_BLOCK);

      // same as after rw_chunk is called
      position += f_size;
      f_size = rip->i_size;
      rip->i_mode = I_REGULAR;
      is_immediate = 0;
     }
```

```
  } else {
    is_immediate = 1;
  }
}
 if (is_immediate == 1) {
   if (rw_flag == READING) {
     r = sys_safecopyto(VFS_PROC_NR, gid,
     (vir_bytes) cum_io,
     (vir_bytes)(rip->i_zone + position),
     (size_t) nrbytes);
   } else {
     r = sys_safecopyfrom(VFS_PROC_NR, gid,
     (vir_bytes) cum_io,
     (vir_bytes)(rip->i_zone + position),
     (size_t) nrbytes);
     IN_MARKDIRTY(rip);
   }

   if (r == OK) {
     cum_io += nrbytes;
     position += nrbytes;
     nrbytes = 0;
   }
   for (int i = 0; i < f_size; i++) {
     immed_buff[i] = *((char *) rip->i_zone)+i);
   }
   printf("immedbuf: %s\n", immed_buff);
 }
}
/** end **/
```

References

- Paper on immediate files (http://dare.ubvu.vu.nl/bitstream/handle/1871/2604/11033.pdf)
- Operating Systems: Design And Implementation 3rd Edition by Andrew Tanenbaum
- Major Project Report (report.pdf)