Design Document

1. Introduction……1
2. New system call……..1-5
   2.1 System call description……1-3
   2.2 Error Handling……3-4
   2.3 How to implement these system call……4-5
3. Discussion about recover situations……..6
4. Extra Bonus Request……6-7

1. Introduction

In this project, we add some system calls to get inode and zone bitmap to implement directory, inodebitmap and zonebitmap walker.

With the help of these walkers, we add other system calls to break or interrupt bitmap or directory file and then recover them.

2. New system call

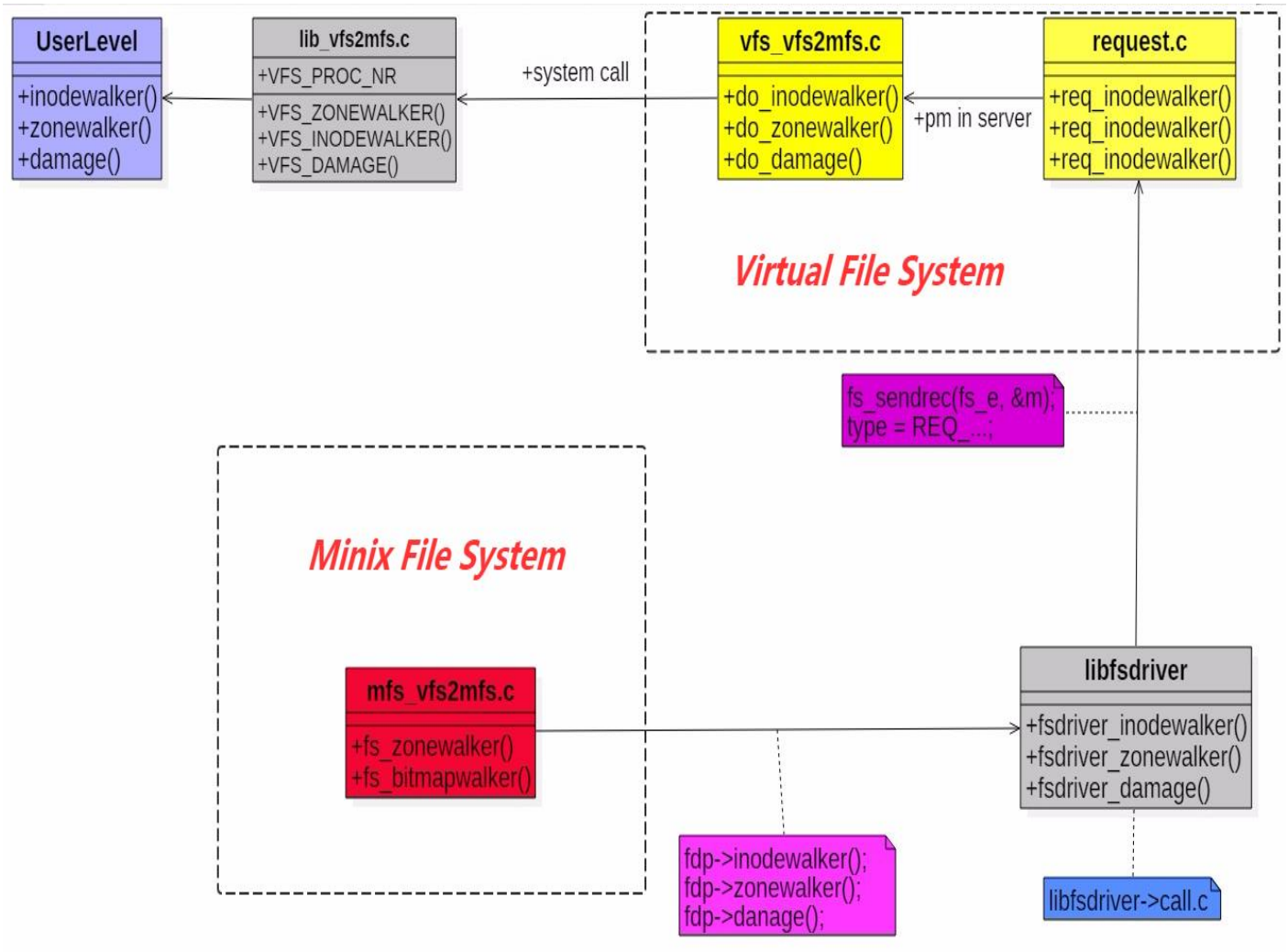
Because we need VFS to invoke MFS's system calls that invoke some specific functions and get inode, zone bitmap, so we add some new system calls.

2.1 system call description

New system calls are created as required. To add these system calls, we add some new files: syscl_mfs.c, lib_vfs2mfs.c, vfs_vfs2mfs.c, And we modify the callnr.h, makefile, proto.c and table.c in /servers/vfs and /fs/mfs; makefile.inc in /lib/libc/sys; call.c, fsdriver.h and table.c in /lib/lib/libfsdriver.h. The detail of each system call is shown in the following:

System call	API used in vfs_vfs2mfs.c	Input	Output	Description	Exception
VFS					
VFS_INODEWALKER	int do_inodewalker()	Void	int	If success, return 0	If block doesn't exist, return -1
VFS_ZONEWALKER	int do_zonewalker()	void	Int	If success, return 0	If zone number overflow, return -2
VFS_RECOVERY	int recovery(int mode)	int mode	Int	Recover inode or zone bitmap or directory file depends on 'mode'. If success, return 0	If mode error, return -3; If recover failed, return -4;
VFS_DAMAGEINODE	int damage_inode(int inode_nb)	int inode_nb (inode number)	Int	If success, return 0	If inode is not used, return -5;
VFS_DAMAGEDIR	int damage_dir(char* path)	char* path	Int	If success, return 0	If path is not exsist, return -5;
VFS_DAMAGEZONE	int damage_zone(int zone_nb)	Int zone_nb (zone number)	Int	If success, return 0	If zone is not used, return -5;
VFS_DAMAGEDIRINODE	int damage_dirinode(char * path)	char* path	Int	If success, return 0	If path is not exesist, return -5;

MFS					
fdr_iwalker	fs_inodewalker()	void	Int	If success, return 0	If block doesn't exist, return -1
fdr_zwalker	fs_zonewalker()	void	Int	If success, return 0	If zone number overflow, return -2



There are other APIs that are mainly aims to get block and bitmap:

```
void init_global();
void get_bitmap(bitchunk_t *bitmap, int type);
void print_bitmap(bitchunk_t *bitmap);
int *get_list_used(bitchunk_t *bitmap, int type);
void lsuper();
void fatal(char *s);
void list_inode(struct inode *inode);
bitchunk_t *alloc_bitmap(int nblk);
void free_bitmap(bitchunk_t *p);
char *alloc(unsigned nelem, unsigned elsize);
char *int2binstr(unsigned int i);
int *get_list_blocks_from_inodes();
int *check_indir(zone_t zno);
int *check_double_indir(zone_t zno);
```

2.2 Error handling

The error code is described in 1.1. And we have one file error_code.h in libc to implement error handler. Generally, if error_code is -5, the service will ask for a request again.

2.3 How to implement these system calls

We need to define system call in callnr.h and implement them in /lib/libc/lib_vfs2mfs.c. Simultaneously we implement API in /vfs/vfs_vfs2mfs.c and /mfs/syscl_mfsf.c.

By now, the user will invoke the system call in VFS, and then, the system call will send request to mfs to get the physics structure in file system. These structures are bitmaps. In this way, we can get directory, inodebitmap and zonebitmap walker.

As for the recover and damaged tools, we set static variable to record the state of inode and zone bitmap.

In inode and zone damaged tools, we will record the bitmap first, and then change their status (from 0 to 1 or from 1 to 0).

In directory file damaged tool and inode of directory damaged tool, we just change the file structure and inode directly. Because we can refer the bitmap.

In recover tool, we will check the recover mode first. If it is inode and zone bitmap recover mode, we will examine whether bitmap record is correct. If so, change the bitmap according to record; If it is directory file recover mode, we will change the file structure refers to its zone bitmap; If it is directory inode recover mode, we will change the inode status refer to inode bitmap.

3 Discussion about recover situations

Part of the superblock is damaged

The inode of a file is damaged.

If inode of a file is damaged, we will check inode bitmap first. Refered to the bitmap, we will know which inode is damaged and how to fix it.

A block (or blocks) allocated to a file is damages

If a block is damaged, we will check zone bitmap first. Refered to the bitmap, we will know which block connected to zone is damaged and how to fix it.

4 Extra bonus request

In this project, we implement a very interesting driver to help us finish the tasks. We refer to the fsdriver added in minix3.4, set a new driver to connect VFS with MFS, and the new driver in our project called fsdriver, too.
The main purpose of both fsdriver is the same: to achieve the bitmap structure's delivery. Because in minix, when we invoke some system call about file system, the VFS's system call will invoked first. However, the bitmap we want to get is defined in MFS. So we need to send a request from VFS to MFS to get them.
And we implement this function as a driver, fsdriver.

In the user mode, we add one call.c in /lib. In this file we can send the request we mentioned before and get the respond as a message. The request will sent when we invoke VFS's specific system call and it will invoke MFS's specific system call.
We think this deserves extra bonus because it is really helpful and special.

Another bonus is that we add inode walker inside directoryWalker which is very convenient.