design.txt

DESIGN DOCUMENT

Won Ko	woko@ucsc.edu
Cesar Casil	ccasil@ucsc.edu
Allston Mickey	amickey@ucsc.edu
Vinod Krishnamurthy	vkkrishn@ucsc.edu

GOAL

The goal for this assignment is to implement a FAT-based file system in user space by using the FUSE system framework. This will give us experience on extending an operating system using user code. It will also give us an idea of how we can quickly implement a new file system along with learning about all of the different types that are being used today!

ASSUMPTIONS

About the FAT file system:

- Originally designed for small hard drives (floppy disks, thumb drives, and SD cards)
- Single list of blocks in use
- Root directory “file” starts in block 0
    - Other files have first block number in directory entry

FUSE: file system intercepts calls and lets a user-space program handle them

Block Sizes:
From textbook: “ Having a large block size means that every file, even a 1-byte file, ties up an entire cylinder. It also means that small files waste a large amount of disk space...Thus if the allocation unit is too large, we waste space; if it is too small, we waste time.”
Larger blocks are faster (transfer more data per seek) but less efficient (waste space).

DESIGN

A file system controls how data is stored and retrieved; without it, information would be placed in what could be described as a bucket of data with no way to tell where one piece of information stops and the next begins.

We are implementing the File Allocation Table (FAT) file system by using the FUSE system framework. By using FUSE we are able to layer the FAT file system on top of the one currently being used in FreeBSD.

To test our FAT file system, we will pick two different block sizes:

Case 1: 512B

Case 2: 8KB

The block size we select will impact our maximum file size, maximum file system size, and the performance of the file system.

PSEUDOCODE

<u>fuse_fat.c</u>

func get_block_type = returns whether the block is a super block, FAT block, or data block based on its index

```
// set up super block
func initialize_super_block (block super_block){
    Set write position to pointer at beginning of super_block
    copy magic number into super_block at write position
    Offset the write position by 32 bits
    Copy the number of total blocks into super_block at write position
    Offset the write position by 32 bits
    Copy the number of fat blocks into super_block at write position
    Offset the write position by 32 bits
    Copy the block size into super_block at write position
    Offset the write position by 32 bits
    Copy the starting root block index into super_block at write position
}

// set up FAT blocks
func initialize_fat () {
    FOR block in FAT blocks
        FOR entry in block
            IF block entry corresponds to ROOT_BLOCK
                Set entry value in FAT to -2
            IF block entry corresponds to super block or FAT block
                Set entry value in FAT to -1
            ELSE
                Set entry value in FAT to 0
}

// initialize disk, sets up super block and fat blocks
func initialize_disk() {
    FOR every block in system
        Allocate space for block
    initialize_super_block()
```

```
        initialize_fat()
}

// parses char array of 64 bytes into directory struct
func get_directory (char bytes[64]){
        Set write position to beginner of bytes array
        Allocate memory for directory struct

        Copy first 24 bytes from char array to filename (24 bytes)
        Copy first 8 bytes from char array to creation time (8 bytes)
        Copy first 8 bytes from char array to modification time (8 bytes)
        Copy first 8 bytes from char array to access time (8 bytes)
        Copy first 4 bytes from char array to file length (4 bytes)
        Copy first 4 bytes from char array to start block (4 bytes)
        Copy first 4 bytes from char array to flags (4 bytes)
        Copy first 4 bytes from char array to unused (4 bytes)
}

// write directory contents into char array of 64 bytes
func write_directory (directory d, char bytes[64]) {
        Copy filename from d to first 24 bytes of char array
        Copy creation time from d to next 8 bytes of char array
        Copy modification time from d to next 8 bytes of char array
        Copy access time from d to next 8 bytes of char array
        Copy file length from d to next 4 bytes of char array
        Copy start block from d to next 4 bytes of char array
        Copy flags from d to next 4 bytes of char array
        Copy unused from d to next 4 bytes of char array
}

// searches given block to see if it has filename
// returns directory entry
func search_block (block start_block, char filename) {
        FOR directory entries in data block
                Dir d = get_directory(entry)
                IF (d->file_name and filename match)
                        RETURN d
        RETURN NULL
}

// recursively searches the blocks for the matching filename
// returns directory entry
func find_file_in_block (block start_block, char path){
        Token = first token in path
        Rest_of_path = rest of path string without token
        Dir d = search_block (start_block, token)
        IF (d == NULL) return NULL
        IF (rest_of_path != NULL)
                RETURN find_file_in_block(blocks[d->start_block], rest_of_path)
        RETURN d
}

// looks in the FAT for the link to the next block FROM the input index
// returns value
func get_fat (int32_t index) {
        FAT_block_num = the FAT block index belongs to
        Byte_offset = the offset within the FAT block where index is located
        RETURN blocks[FAT_block_num]->bytes[byte_offset]
}

// sets value in FAT at specific index
func set_fat (int32_t index, int32_t value) {
        FAT_block_num = the FAT block index belongs to
        Byte_offset = the offset within the FAT block where index is located
        blocks[FAT_block_num]->bytes[byte_offset] = value
}

// looks for file starting from root block
// returns directory entry
func find_file (char path) {
        FOR valid block in FAT
                Dir d  = find_file_in_block(valid block, path)
                IF (d != NULL) RETURN d
        RETURN NULL
}

// searches for file in system
// fills buffer with stat info about file
func fat_getattr (path, buffer) {
        IF path is root directory
                buffer->mode = S_IFDIR
                buffer->nlink = 2
        ELSE
                Dir d = find_file(path)
                IF (d == NULL) RETURN -ENOENT
                IF (d is a directory)
                buffer->mode = S_IFDIR
                buffer->nlink = 2
            ELSE
                buffer->mode = S_IFREG
                buffer->nlink = 1
            buffer->st_size = d->file_length
            buffer->access_time = d->access_time
            buffer->mod_time = d->mod_time
            buffer->creat_time = d->creat_time
        buffer->mode = ALL PERMISSIONS
        RETURN 0
}

// goes through every directory entry in a block and adds it to an array of entries
func get_dir_entries (block b) {
        dir_entries;
        FOR every entry in block
            dir_entries[index] = entry
        RETURN dir_entries
}

// finds directory, and fills buffer with stats for all files within that directory
func fat_readdir (path, buffer, filler, offset, fi) {
        Dir d
        IF path is not ROOT
            D = find_file(path)
            IF d is NULL RETURN NULL
            filler(buf, “.”, NULL, 0)
            Filler(buf, “..”, NULL, 0)
            Start_block_idx = d->start_block
        ELSE
            Start_block_idx = ROOT

        FOR each FAT block
            Dir_entries = get_dir_entries(block)
            FOR each dir_entry in dir_entries
                IF (d is a directory)
                    buffer->mode = S_IFDIR
                    buffer->nlink = 2
                ELSE
                    buffer->mode = S_IFREG
                    buffer->nlink = 1
                    buffer->st_size = d->file_length
                    buffer->access_time = d->access_time
                    buffer->mod_time = d->mod_time
                    buffer->creat_time = d->creat_time
                    buffer->mode = ALL PERMISSIONS
                Filler(buf, d->file_name, st, 0)
            free(dir_entries)
        RETURN 0
}

// checks existence and permissions of a given path
func fat_open (path, file_info) {
        IF path is ROOT return 0
        Dir d = find_file(path)
        IF d == NULL RETURN -ENOENT
        RETURN 0
}

// scans the FAT for a free block
func get_free_fat_block_idx() {
        Block_idx = 0
        FOR each FAT block
            FOR each entry in FAT block
                IF entry points to a free block RETURN block_idx
                Block_idx++
        RETURN -1
}

// reads specified size of bytes into buffer
func fat_read (path, buf, size, offset, file_info) {
        Dir d = find_file(path)
        IF d == NULL RETURN -ENOENT
        Write_position = buf
        int Bytes_to_read
        FOR block in FAT
            IF size is greater than BLOCK_SIZE
                Bytes_to_read = BLOCK_SIZE
            ELSE
                Bytes_to_read = size
            Write_position += bytes_to_read
            Size -= bytes_to_read
            IF size <= 0 BREAK
        RETURN 0
}
```

```
// finds empty directory entry given parent block
func find_empty_dir (block parent_block) {
    FOR entry in data block
        my_dir = entry
        FOR bytes in entry
            IF bytes != 0x0000000
                BREAK
        RETURN my_dir
    RETURN NULL
}

// make a directory at the specified path
func fat_mkdir (path, mode) {
    Dir d = find_file(path)
    Int slash = 0
    IF d == NULL
        FOR each char in path
            IF char is a “/”
                Slash = char index
        STRNCPY(parsedpath, path, slash*sizeof(char))
        IF parsedpath == NULL
            Parent_block_idx = ROOT
        ELSE
            Dir parent_entry = find_file(parsedpath)
            Parent_block_idx = parent_entry->start_block

        Dir new_dir = malloc(sizeof(dir))
        STRCPY(file_name, basename(path))
        STRCPY(new_dir->file_name, file_name)

        new_dir->time = time(NULL)

        Int free_block_idx_for_contents = get_free_fat_block_idx()
        new_dir->start_block = free_block_idx_for_contents

        set_fat(free_block_idx_for_contents, LAST_BLOCK_IN_FILE)

        // set up flags here

        Char write_position = get_space_for_directory(parent_block_idx)
        write_directory(new_dir, write_position)
    Return 0
}

// finds space where a directory can be written without overwriting
func get_space_for_directory (int block_idx) {
    Char write_position = find_empty_dir(blocks[block_idx])
    IF write_position != NULL
        RETURN write_position

    Int next_block_idx = get_fat(block_idx)
    IF next_block_idx == NO_ALLOC_BLOCK RETURN NULL
    ELSE IF next_block_idx is LAST_BLOCK_IN_FILE or FREE_BLOCK
        Int free_idx = get_free_fat_block_idx()
        set_fat(block_idx, free_idx)
        set_fat(free_idx, LAST_BLOCK_IN_FILE)
        RETURN blocks[free_idx]->bytes[0]
    Return get_space_for_directory (next_block_idx)
}

// renames file at path to specified new name
func fat_rename (path, newname) {
    Dir d = find_file(path)
    IF d is NOT a directory
        STRCPY(d->file_name, newname)
    RETURN 0
}
```