John Witt
Anastasia Nesterova

FIle System Project Write Up

.h files:

pdosfilrsys.h

This file includes multiple type definitions in the form of structs. These include DIR_ENTRY, this holds information about a file in the directory, DIRECTORY_BLOCK which holds information about whether a block is a directory, an int representing the next entry and also an array that holds a DIR_ENTRY array for the block. This file also initializes a FAT_BLOCK and a DATA_BLOCK both initialized as arrays so that these structures can hold information within the block. This file also utilizes a union for the 3 different block types under the overarching DICK_BLOCK name. In addition, this file defines a file-descriptor type called PDOS_FILE that holds information about a file. We also define MYFS as a file path.

pdos.h

Here we have prototypes for our different pdos functions. Including _pdos_open_fs, _pdos_close_fs, _pdos_write_block and _pdos_read_block.

.c files:

pdos_mkdisk.c

Here we create a shared memory object using shm_open with the name MYFS (actually declared as a file path). We then truncate the size of the shared memory to the size passed into the function. In this way we create the disk storage for the in-memory file system.

pdos_mkfs.c

First of all we implement a few helper functions here for opening and closing the fs and also reading and writing to a block. For open we use mmap, for close munmap, for reads and writes we use memcpy. We also have functions for getting and setting the state of a block (-1, 0 or a short 4-1023) in the fat table. In the function we format the file system, populating the disk with the ID string block, the two fat blocks and the directory block.

pdos_open.c

This file/function opens the file system and looks for the name of the file in the directory block. If the file is found and it the mode passed in is write, the contents of the file is passed into the buffer and the fat and directory blocks are updated to show that there is no data in the blocks. The file then returned with mode write and pos 0. If the file is not found it is created unless the mode passed in is read or the number of directories (blocks) available is maxed out. If the file is created, populate properties of PDOS_FILE and find on which block it will start and update the fat and directory blocks.

pdos_fgetc.c

If file mode is w, exit. If it is ok to read we check whether or not we are accessing a point past the end of the file. We do this by calculating our offset from the beginning of the file and comparing this to the

filelength. If we are past the end of the file we return -1. If we are within the filelength but at the end of a block/buffer (pos = BLOCK_SIZE) we load next buffer. Then we return the byte at pos in the file.

pdos_fputc.c

If file mode is r, exit. If the file buffer has space for the byte, add byte to buffer. If the buffer has no more space we need to add a new block to the file (file size increases). We update filelength and filemodtime in the corresponding directory entry. We then write the full buffer to the disk. We then must search for a new free block using the fat table, if there isn't any space we throw an error. If there is space and there is no next block stored in fat we modify the fat table to reflect the file adding a block. If there already was a nextblock stored we update our file buffer to be that of the next block. then finally we add the byte to the new buffer which corresponds to the new block.

pdos_close.c

If file has write permission we need update size and write buffer to disk. If the buffer is for the last block of the file we need to update size. We then free the file.

pdos_dir.c

For this file we use a for loop to store all of the filenames in the directory block in an array of file names and then we return that array.

pdos_mkdir.c

This file creates a new sub directory in the root directory. First we make sure the name isnt too long. We then check the directory entries to see if any of the entries are directories with the same name. We don't create a new directory if this is the case. If this is not the case and there is enough space on disk and there are still available directory entries we search for a free block. We update the fat table to show that there is now a directory corresponding to the block. We also update the directory block's directory entry, updating this particular entry as a directory.