TMDE Documentation

Project Overview:

The project's goal was to implement a minimal disk emulator on the Linux environment using the C programming language. The goal was to transform the theoretical knowledge in low-level systems into practical experience. Although the features it supports are minimal, I was able to advance my skills and knowledge in disk emulators.

Learning and Design Process:

My journey in low-level systems began over six months ago, when I read a low-level programming book written by Igor Zhirkov, where I learned the assembly language, and a lot of information regarding this area. Now you might think that I copy-pasted the TMFS documentation here, and I would say that you are partially right. But here we diverge a bit. For a filesystem to run, you need a disk emulator and what's better than building your own? (borrowing one online). Correct, and that's why I built my own :).

Filesystem requirements:

1- Must use exactly 512-byte blocks in all operations.
2- Must use "dinit", "dmount", and "dunmount" to initialise, mount, and unmount the disk.
3- The maximum file size is limited to ~32MB.
4- Must follow a block design.
5- Must use the error codes provided in "os.h".

Current Progress and Modifications:

The disk emulator supports the following features:

. Printing information about the drive.
. Initialising the drive.

TMDE Documentation

. Mounting the drive.
. Unmounting the drive.

Function-Level Documentation:

The functions:

. dshow: This function prints information about the drive.

Parameters:

. A pointer to the disk we want to print information about.

Functionality:

. First, we check if there is a disk or not.
. Then we print the drive number, file descriptor, and the number of blocks that are represented through “driveNo”, “fd”, and “blocks” fields of “dd” respectively.

Returns:

void.

. dinit: This function initialises the disk drive and the filesystem is mounted here.

Parameters:

None.

Functionality:

. We start by assigning 0 to mount.
. Then we loop through the drives in our system. Mounting them, and unmounting them. But in between, a filesystem is mounted.
. Finally, we return.

Returns:

void

. dunmount: This function unmounts the disk drive.

Parameters:

. A pointer to the disk we want to unmount.

Functionality:

. First, we check if there is a disk or not.
. Then we close the file descriptor field of “dd”.

. After that, we assign the result of ANDing the binary NOT of the drive number field of “dd” and the mounted global variable to “x”. Which in turn will have the value 0. So, we assign “x” to “mounted”, free “dd” and return.

Returns:

void.

. dmount: This function mounts the disk drive.

Parameters:

. The drive number of the disk that we want to mount.

Functionality:

. We start by checking: If the drive number is valid or not. If it is, we continue. Else, we return 0. And if ANDing the “mounted” and the “driveNo” variables is equal to zero or not. If it is, we continue. Else, we return 0.

. Then we assign the size of the “sdisk” structure to size. And the return value of calling “malloc” with “size” as its argument to “dd”. Of which we then check if zero or not. If it is, the mallocing operation failed and we return 0. Else, we continue by zeroing out “dd”.

. After that, we concatenate “BasePath” and “drivNo” by calling “strnum”, and assign the return value to “file”.

. Moving on, we assign the return value of opening “file” with read and write permissions to the variable “tmp”. Of which we then check if it is smaller than 3 or not (because -1 is for when its an error. And 0,1, and 2 are reserved for the standard input, output, and error respectively). If it is, we free “dd” and return 0.

. Then, we assign “tmp” to the “fd” field of “dd”. And the return value of calling “fstat” (to get the file size in blocks) to “tmp”. Of which we then check if its

true or not OR if the “st_blocks” (which is equal to our block size) field of “sbuf” is equal to zero or not. If the result of the OR operation is false, we continue. Else, we close the file descriptor of “dd”, free “dd”, and return 0.

. After that , we assign the “st_blocks” field of “sbuf” minus one to the “blocks” field of “dd”, the “driveNo” to the “driveNo” field of “dd”, and the OR between “driveNo” and “mounted” to mounted.

. Finally, we return “ dd.

Returns:

A pointer to the disk mounted.

.zero: Zeroes out bytes of the memory region pointed to by the first parameter by iterating through its bytes and zeroing them one by one.