CS39002 Operating Systems Laboratory
Spring 2026

Lab Assignment: 10
Date posted: 10-Apr-2026

In-memory Simulation of the FAT File System

In this assignment, you simulate a file system in a 64 MB chunk of the main memory. We call this memory chunk the *virtual disk* (VD). The virtual disk is assumed to consist of 1 KB blocks, that is, VD contains exactly 64 MB / 1 KB = 2^{16} = 65536 blocks, numbered 0, 1, 2, … , 65535, so a block number can be specified by 16 bits. In order to simulate real-life situations, let us use unsigned int (32 bits) to number the blocks. The virtual disk is organized as follows.

Block 0 stores the total number of blocks, the number of free blocks, and the first-block number of the root directory. We plan to work with 65536 blocks. If your system does not permit 64 MB segments, go for smaller virtual disks (make the size an integral number of MB's, at least 1 MB). In case you need to use a VD smaller than 64 MB, you may use a free-block bitmap table and a FAT as per your need (you may also use the default settings as explained below). You may as well go for a larger (than 64 GB) VD. But then, you need more than 8 + 256 blocks to store all the information about your disk. Block 0 is needed to store the relevant details.

Blocks 1 – 8 store the information of which blocks are free and which are not. There are 2^{16} blocks in total in VD. We store the free/allocated information as a bitmap, so we need to use 2^{16} bits = 2^{13} bytes = 2^3 = 8 blocks for the bitmap.

Blocks 9 – 264 store the FAT (that is, the table of next-block pointers). We use 32 bits (4 bytes) for a block number, that is, we can store 1 KB / 4 byte = 256 pointers per block, so we need 2^{16} / 256 = 256 blocks to store the FAT.

Block 265 is the starting (first) block of the root directory.

Blocks 266 – 65535 store user files and directories (including extensions of the root directory).

We store integers and bits in the raw format (not as numeric strings). For example, an unsigned integer can be stored in 32 bits (or 4 bytes). A string representation of an unsigned int may take as many as 10 characters (in decimal) or 8 characters (in hexadecimal). This is unacceptable. In order to read raw data, you need to use the `memcpy()` function declared in `<string.h>`. This function has the following prototype.

```
memcpy(dest_pointer, src_pointer, no_of_bytes);
```

The memory addresses `dest_pointer`, … , `dest_pointer + non_of_bytes - 1` should be disjoint from the memory addresses `src_pointer`, … , `src_pointer + non_of_bytes - 1`. If you want to read an integer `t` in the raw format from the virtual disk position `p`, make the following call. Assume that VD is an array of characters.

```
memcpy(&t, VD + p, 4);
```

The reverse transfer can be done as follows.

```
memcpy(VD + p, &t, 4);
```

If you want to read or write a structure `mystruct` from or to the position `p` of VD, use the following.

```
memcpy(&structvar, VD + p, sizeof(mystruct));
memcpy(VD + p, &structvar, sizeof(mystruct));
```

Only names of files and directories are stored as literal strings. All other items (the contents of Block 0, the FAT, and the metadata associated files and directories) should be stored and accessed in the raw format. Moreover, the bitmap should be stored as a sequence of bits. You can set/clear each bit using bit-wise operations on words (or characters).

Storing directories

A directory is stored as a linear (unsorted) list of the metadata of the files and directories it contains. For simplicity, the metadata consists of only the following items.

```
struct metadata {
    char type;
    char name[23];
    unsigned int size;
    unsigned int firstblock;
};
```

Each `metadata` structure stores information of a single file or subdirectory. The field `type` is `'f'` for a file, or `'d'` for a directory. We store the `name` of the file or subdirectory as a 23-byte null-terminated string, that is, each name can be at most 22 characters long. We assume that file/directory names consist only of printable ASCII characters, but do not contain white spaces, `/`, or the back-quote character. For a regular file, `size` is the number of bytes in that file. For a subdirectory, it is the number of entries in that directory. Finally, `firstblock` stores the number of the first block storing the file or the subdirectory.

The size (width) of the `metadata` structure is 32 bytes, that is, each block can accommodate entries for only 1024 / 32 = 32 files and/or subdirectories. If the number of entries is larger, then a second free block is used, and a pointer to this second block is stored in the FAT section of the disk. If the number of entries exceeds 64, then a third block is needed, and so on. The links are always to be found from the FAT, and are never stored in the individual blocks.

Each directory always contains two subdirectories `.` and `..`. The first `metadata` entry of a directory stores information about itself (`.`). In particular, the number of entries in that directory is stored in the `size` field of this entry. The second `metadata` entry of the directory is a pointer to the parent (`..`) of that directory. The size of this entry can be 0, because the actual size of the parent directory is stored in the first `metadata` entry of the parent itself, and can be found by following the `firstblock` pointer in the entry for `..`. This is also true for all subdirectory entries. Note that the size of a directory (even if empty) is at least 2.

Storing files

A file is stored character by character in (an integral number of) blocks. The number of the first block storing a file can be obtained from the corresponding `metadata` entry of the directory that hosts the file. Subsequent blocks (if any) storing the file are linked. These links can be found from the FAT section of the disk.

In order to specify a file or directory, you can use a relative path or an absolute path. Relative paths are computed with respect to the current directory. Some examples: `myfile.txt`, `.././readme` or `assignments/LA10/makefile` or `./assignments/LA10/makefile`, or `../OS/assignments/LA10/makefile`. For directories, we may have an optional trailing `/` character like: `.`, `./`, `..`, `../`, `../OS`, `../OS/`, `assignments/LA10`, `assignments/LA10/`, `./assignments/LA10`, `./assignments/LA10/`.

File or directory names with an absolute path start with `/`, and are to be computed with respect to the root directory. Examples: `/`, `/courses/OS/readme.txt`, `/courses/OS`, `/courses/OS/`, `/courses/OS/../ALGO/syllabus.txt`.

The disk operations to be supported

For simplicity, the file system will support only the following operations.

`md` or `mkdir`: Create a new directory. The name of the directory can be specified by a relative or an absolute path.

```
md OS/assignments/LA10
md /courses/OS/slides
md ../lib
```

`cd` or `chdir`: Change to a relatively or absolutely specified directory. If no argument is specified, jump to the root directory, that is, `cd` is the same as `cd /`.

```
cd ..
cd OS/assignments/
cd ../courses/algolab/programs
```

`dir`: Print only the names of all the files and subdirectories for the current directory. This does not take any argument.

`ls`: Print the detailed contents of the specified directory (if no directory is specified, the current directory is implied). The listing includes the types, names, sizes, and first blocks of all the `metadata` entries stored in the directory block(s). You can also supply a file name as the argument. In that case, only the details of the specified file will be printed. It suffices for `dir` and `ls` to print the contents of a directory in the sequence of creation, that is, you do not need to sort the list.

```
ls
ls ../
ls /courses/lab
ls ./courses/lab/os/assignments/
ls ./courses/lab/os/assignments/A15/makefile
```

`cp` or `copy`: Three types of file copying are allowed: hard disk (HD) to virtual disk (VD), VD to HD, and VD to VD. Files/directories in HD or VD can be specified using relative or absolute path names. In order to disambiguate which types of files we are talking about, we use a backward quote (`) before every file name residing on HD. Any source or destination from or to HD must be a (quoted) file name (directories are not permitted). Any source in VD must likewise be a file name. The destination in VD may be either a file name or a directory name. In the first case (destination is a file name in VD), the source is copied to the specified file. In the second case (destination is a directory in VD), a file of the same name as the source is copied to the destination directory. If the destination file is already existing, the old version must be overwritten. Otherwise, a new file is to be created.

```
cp `../makefile ./
cp progs/LA15/makefile `makefile
cp data.txt tmp/data/d.txt
cp data.txt tmp/data/
```

`prn` or `type`: Print a file (not a directory) on VD to the terminal. Again absolute or relative file names may be supplied. Printing makes sense only for text files.

What do you have to code

Write three separate C/C++ files for doing the following jobs. Also write a makefile, and if needed, other header or source files.

Disk manager

The disk manager creates or removes a 64 MB shared-memory segment. This segment is used as the virtual disk VD. You should be able to use VD from any terminal, and it should remain in the main memory until the disk manager explicitly removes it. The disk manager runs in two modes: creation mode and removal mode.

In the creation mode, the disk manager not only creates the shared-memory segment (with a reproducible key obtained by `ftok`()) but also initializes the memory to stand for an empty FAT-formatted disk. In particular, it will initialize Block 0 to store the total number of blocks, the total number of free blocks, and the block number of the root directory. It will then store the initial bitmap in Blocks 1 through 8 (all blocks are initially free except 0 through 265), and the initial FAT (all next pointers are NULL, that is, 0) in Blocks 9 through 264. Finally, it will create an empty root directory (with only the two entries `.` and `..`) in Block 265. Assume that the parent of the root directory is the root directory itself.

In the removal mode, the disk manager removes the shared-memory segment it created for hosting VD.

Since multiple processes can access (read from and write to) the virtual file system (a shared-memory segment), a mutex should guard against simultaneous accesses to the virtual disk. For simplicity, you do not have to use any mutex (semaphore). In this assignment, the disk is accessed only by manually entered user commands which are processed quickly, so you may assume that two such commands never run at the same time. The emphasis of this assignment is on file-system implementation (not synchronization).

Disk Utilities

Write a file `diskutils.c(pp)` to implement disk-related operations. Write it as if it will form a library (although you are not required to generate a `.a` or `.so` archive from it). Unless really needed, libraries do not declare global variables. In this case, you declare only the following global variables.

```
unsigned char *D;           /* A pointer to the shared-memory segment storing VD */
unsigned int NBLOCKS,       /* The number of blocks on VD */
             NFREEBLOCKS,   /* The number of free blocks on VD */
             RBN;           /* The first-block number of the root directory */
```

This file does not need to keep track of the current working directories of user processes. It makes no sense to do that since multiple shells may access the disk from different terminals, and have their individual current working directories, so there is no concept of a global current working directory.

Write the following functions.

`joindisk()`: A user process gets attached to the shared-memory segment VD by this call. It also initializes the global variables `D`, `NBLOCKS`, `NFREEBLOCKS`, and `RBN`.

`leavedisk()`: A user process gets detached from VD by making this call.

`getfreeblock()`: A free-block number is to be returned by this function. In order to have a feeling of non-contiguous allocation, return a free-block number randomly. The bitmap table stored in Blocks $1-8$ should be consulted to check whether a randomly generated block number corresponds to a free block. If not, another random block number is generated and checked, and so on.

`freeblock()`: Free an allocated block. This is needed during overwriting of an existing file.

Each process using VD maintains its copy of the global variable `NFREEBLOCKS`. However, since multiple processes can update the count by a `getfreeblock()` or `freeblock()` call in two running sessions, the local copy of the count may fail to store the correct information. That means that every time a block is allocated or freed, it is necessary to read the shared count `NFREEBLOCKS` from Block 0, update it, and write it back to Block 0. In reality, a mutex is needed to guard against multiple accesses at the same time. As stated earlier, assume that the user never makes multiple requests together, and you may skip this protection.

Then, implement the functions for disk access discussed earlier. This includes functions for directory listing, directory creation, file copying, and file printing. The change-directory command cannot be implemented here, because for reasons mentioned above, there is no global concept of a current directory. Therefore each individual process using the disk-related functions must send its own current working directory as a parameter. This parameter is needed by the functions to locate relative addresses specified as file or directory names.

Some other functions frequently used by the disk-related functions may also be implemented, like a function to check the existence of files and directories, a function to obtain the `metadata` of an existing file or directory, and so on. You are free to use your own helper functions.

A shell program (foosh)

This is the user-level program that implements the commands mentioned in the section Disk operations to be supported. It is a command interpreter that keeps on reading lines of commands from the user, splits each line into arguments, and takes necessary actions. The special command `exit` or `quit` terminates the shell.

The shell starts by joining the disk (thereby initializing its global variables `D`, `NBLOCKS`, `NFREEBLOCKS`, and `RBN`). It also keeps track of two information about the current working directory of that shell. The first item is `CBN` (the number of the first block of the directory). The second item is a string (`CWD`) showing the absolute path of the current directory starting from the root of the virtual file system. This string is to be printed at every prompt of the interpreter loop. Because `CBN` and `CWD` are maintained by the shell, implementing `cd` (or `chdir`) is the duty of the shell program. Write a function to this effect. This is what happens in reality. In Linux, try `whereis mkdir`, `whereis ls`, or `whereis cp`, and you will get an executable file. But `whereis cd` will print no executable file.

For all other commands, call appropriate routines from the diskutils library. Of course, you do not need to compile the disk utilities to a static or dynamic library. You instead `#include "diskutils.c"` in the program `foosh.c`(`pp`) for the shell.

Before exiting, make sure that the shell leaves from the shared-memory segment.

What to submit

You need to submit the following files.

- The code `diskmanager.c`(`pp`) for creating, initializing, and removing the virtual disk. This source file will contain a `main()` function.
- The file `diskutils.c`(`pp`) to implement the disk utilities. This file will contain no `main()` function.
- The shell program `foosh.c`(`pp`). This will compile to an executable `foosh` (the Foo version of `sh`), so this source file too must contain a `main()` function.
- Any other header or source files that you may find convenient to separate as a logical module.
- A makefile with targets `diskmanager`, `foosh`, and `clean`. The file should also contain a target `all` (or `compile`) as the first target to build both `diskmanager` and `foosh`.

Club together all the above files in a single tar/tgz/zip archive, and submit only that archive.

Sample Output

First, run `diskmanager` in the create mode in a shell. Then run `./foosh` in one terminal. To start with (during the first invocation of `foosh`), we have an empty FAT-formatted disk. We have the following session. Here, `$` is the prompt of your `bash`, and `[foosh] VD:...>` is the prompt of `foosh`.

```
$ ./foosh
+++ Number of blocks = 65536
+++ Number of free blocks = 65270
+++ First block of the root directory = 265
[foosh] VD:> md course
[foosh] VD:> md course/os
[foosh] VD:> md course/os/theory
[foosh] VD:> md course/os/lab
[foosh] VD:> ls
Total 3 entries
--------------------------------------------------------------------------
TYPE                        NAME            SIZE          FIRST BLOCK
--------------------------------------------------------------------------
 d                           ./               3          265
 d                           ../              3          265
 d                       course/              3          61452
--------------------------------------------------------------------------
[foosh] VD:> cd course
[foosh] VD:/course> ls
Total 3 entries
--------------------------------------------------------------------------
TYPE                        NAME            SIZE          FIRST BLOCK
--------------------------------------------------------------------------
 d                           ./               3          61452
 d                           ../              3          265
 d                           os/              4          45463
--------------------------------------------------------------------------
[foosh] VD:/course> cd os
[foosh] VD:/course/os> ls
Total 4 entries
--------------------------------------------------------------------------
TYPE                        NAME            SIZE          FIRST BLOCK
--------------------------------------------------------------------------
 d                           ./               4          45463
 d                           ../              3          61452
 d                       theory/              2          64034
 d                          lab/              2          51894
--------------------------------------------------------------------------
[foosh] VD:/course/os> cd lab
[foosh] VD:/course/os/lab> md sample
[foosh] VD:/course/os/lab> ls
Total 3 entries
--------------------------------------------------------------------------
TYPE                        NAME            SIZE          FIRST BLOCK
--------------------------------------------------------------------------
 d                           ./               3          51894
 d                           ../              4          45463
 d                       sample/              2          12563
--------------------------------------------------------------------------
[foosh] VD:/course/os/lab> md /course/os/lab/sample/RWP
[foosh] VD:/course/os/lab> cd os
*** Error: unable to change to directory os
[foosh] VD:/course/os/lab> cd sample
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/fork1.c .
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/fork2.c .
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/forkn.c .
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/forkarray.c .
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/forkexec.c .
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/execlp.c .
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/execvp.c .
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/signal.c .
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/pipe.c .
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/dup.c .
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/shm.c .
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/sem.c .
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/pthread.c .
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/RWP/dbmgr.c RWP
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/RWP/makefile RWP/
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/RWP/reader.c RWP
[foosh] VD:/course/os/lab/sample> cp `/home/foobar/osl/SampleProgs/RWP/writer.c RWP/
[foosh] VD:/course/os/lab/sample> dir
                    ./                          ../                        RWP/                    fork1.c
                fork2.c                     forkn.c                  forkarray.c                  forkexec.c
               execlp.c                    execvp.c                     signal.c                      pipe.c
                  dup.c                       shm.c                        sem.c                    pthread.c
[foosh] VD:/course/os/lab/sample> ls RWP
Total 6 entries
--------------------------------------------------------------------------
TYPE                        NAME            SIZE          FIRST BLOCK
--------------------------------------------------------------------------
 d                           ./               6          27912
 d                           ../             16          12563
 f                       dbmgr.c           1999          55751
 f                      makefile            226          15992
 f                      reader.c           1575          14467
 f                      writer.c           1667          15950
--------------------------------------------------------------------------
[foosh] VD:/course/os/lab/sample> cd RWP
[foosh] VD:/course/os/lab/sample/RWP> dir
                    ./                          ../                    dbmgr.c                    makefile
                reader.c                    writer.c
[foosh] VD:/course/os/lab/sample/RWP> ls /
Total 3 entries
--------------------------------------------------------------------------
TYPE                        NAME            SIZE          FIRST BLOCK
--------------------------------------------------------------------------
 d                           ./               3          265
 d                           ../              3          265
 d                       course/              3          61452
```

```
------------------------------------------------------------------------
[foosh] VD:/course/os/lab/sample/RWP> cd
[foosh] VD:> dir
                    ./                      ../                 course/
[foosh] VD:> exit
+++ Number of free blocks = 65215
$
```

Now, run `./foosh` again. You will be able to see the directories and files created by the first invocation of `foosh`.

```
$ ./foosh
+++ Number of blocks = 65536
+++ Number of free blocks = 65215
+++ First block of the root directory = 265
[foosh] VD:> ls
Total 3 entries
------------------------------------------------------------------------
TYPE                        NAME            SIZE          FIRST BLOCK
------------------------------------------------------------------------
 d                           ./               3          265
 d                          ../               3          265
 d                      course/               3          61452
------------------------------------------------------------------------
[foosh] VD:> cd os/lab
*** Error: unable to change to directory os/lab
[foosh] VD:> cd course/os/lab
[foosh] VD:/course/os/lab> cp `~/os/pthread.pdf .
*** Error: Unable to read input file ~/os/pthread.pdf
[foosh] VD:/course/os/lab> cp `/home/foobar/os/pthread.pdf PTHREAD.pdf
[foosh] VD:/course/os/lab> ls
Total 4 entries
------------------------------------------------------------------------
TYPE                        NAME            SIZE          FIRST BLOCK
------------------------------------------------------------------------
 d                           ./               4          51894
 d                          ../               4          45463
 d                      sample/              16          12563
 f                  PTHREAD.pdf          836763          27863
------------------------------------------------------------------------
[foosh] VD:/course/os/lab> prn sample/fork1.c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main ()
{
   int n, mypid, parpid;

   printf("Parent: n = "); scanf("%d", &n);

   /* Child creation */
   if (fork()) { /* Parent process */
      mypid = getpid();
      parpid = getppid();
      printf("Parent: PID = %u, PPID = %u...\n", mypid, parpid);
   } else {      /* Child process */
      mypid = getpid();
      parpid = getppid();
      printf("Child : PID = %u, PPID = %u...\n", mypid, parpid);
      n = n * n;
   }

   printf("Process PID = %u: n = %d\n", mypid, n);

   exit(0);
}
```

When the second session is still running, open another terminal, and run `foosh` there too.

```
$ ./foosh

+++ Number of blocks = 65536
+++ Number of free blocks = 64397
+++ First block of the root directory = 265
[foosh] VD:> ls
Total 3 entries
------------------------------------------------------------------------
TYPE                        NAME            SIZE          FIRST BLOCK
------------------------------------------------------------------------
 d                           ./               3          265
 d                          ../               3          265
 d                      course/               3          61452
------------------------------------------------------------------------
[foosh] VD:> md data
[foosh] VD:> cd data
[foosh] VD:/data> md aa
[foosh] VD:/data> md ab
[foosh] VD:/data> md ac
. . .
[foosh] VD:/data> md ff
[foosh] VD:/data> ls
Total 38 entries
------------------------------------------------------------------------
TYPE                        NAME            SIZE          FIRST BLOCK
------------------------------------------------------------------------
 d                           ./              38          57288
 d                          ../               4          265
 d                          aa/               2          60305
 d                          ab/               2          4512
 d                          ac/               2          3498
 d                          ad/               2          42729
 d                          ae/               2          45327
 d                          af/               2          11479
```

```
 d                          ba/              2          32146
 d                          bb/              2          1840
 d                          bc/              2          10401
 d                          bd/              2          28444
 d                          be/              2          4302
 d                          bf/              2          26001
 d                          ca/              2          51804
 d                          cb/              2          55246
 d                          cc/              2          9511
 d                          cd/              2          48562
 d                          ce/              2          61640
 d                          cf/              2          19904
 d                          da/              2          21218
 d                          db/              2          59730
 d                          dc/              2          16493
 d                          dd/              2          2890
 d                          de/              2          62384
 d                          df/              2          8551
 d                          ea/              2          43421
 d                          eb/              2          54186
 d                          ec/              2          35428
 d                          ed/              2          17075
 d                          ee/              2          31471
 d                          ef/              2          29683
 d                          fa/              2          8827
 d                          fb/              2          34195
 d                          fc/              2          12325
 d                          fd/              2          3433
 d                          fe/              2          13986
 d                          ff/              2          23804
-------------------------------------------------------------------------
[foosh] VD:/data> ls ff/
Total 2 entries
-------------------------------------------------------------------------
TYPE                       NAME             SIZE          FIRST BLOCK
-------------------------------------------------------------------------
 d                          ./               2          23804
 d                          ../             38          57288
-------------------------------------------------------------------------
[foosh] VD:/data>
```

Now let the earlier shell (still running) write a file to the `ff` subdirectory.

```
[foosh] VD:/data> cd
[foosh] VD:> cd data
[foosh] VD:/data> dir
                     ./                         ../                        aa/                        ab/
                    ac/                        ad/                        ae/                        af/
                    ba/                        bb/                        bc/                        bd/
                    be/                        bf/                        ca/                        cb/
                    cc/                        cd/                        ce/                        cf/
                    da/                        db/                        dc/                        dd/
                    de/                        df/                        ea/                        eb/
                    ec/                        ed/                        ee/                        ef/
                    fa/                        fb/                        fc/                        fd/
                    fe/                        ff/
[foosh] VD:/data> cp `/home/foobar/os/tut/contiguous.jpg ff/
[foosh] VD:/data> cp `/home/foobar/os/tut/five_lines.jpg ff/
[foosh] VD:/data> cp `/home/foobar/os/tut/Dekker.jpg ff/
[foosh] VD:/data> cp `/home/foobar/os/tut/RAG.jpg ff/
[foosh] VD:/data>
```

The other running `foosh` can see the changes, and it exits.

```
[foosh] VD:/data> ls ff/
Total 6 entries
-------------------------------------------------------------------------
TYPE                       NAME             SIZE          FIRST BLOCK
-------------------------------------------------------------------------
 d                          ./               6          23804
 d                          ../             38          57288
 f                 contiguous.jpg        38817          55730
 f                five_lines.jpg       437148          4818
 f                    Dekker.jpg       128355          14756
 f                       RAG.jpg        92194          21879
-------------------------------------------------------------------------
[foosh] VD:/data> exit
+++ Number of free blocks = 63677
$
```

Let the running `foosh` do the following experiment. Let it read the executable `foosh` from HD to VD, and save it back to HD with the name `barsh`.

```
[foosh] VD:/data> cp `foosh /
[foosh] VD:/data> cd ./.././..
[foosh] VD:> ls
Total 5 entries
-------------------------------------------------------------------------
TYPE                       NAME             SIZE          FIRST BLOCK
-------------------------------------------------------------------------
 d                          ./               5          265
 d                          ../              5          265
 d                      course/              3          61452
 d                        data/             38          57288
 f                        foosh          26464          34072
-------------------------------------------------------------------------
[foosh] VD:> cp foosh course/os/foobarsh
[foosh] VD:> ls course/os/foobarsh
53766 f      26464    course/os/foobarsh
[foosh] VD:> cp course/os/foobarsh `barsh
[foosh] VD:> exit
+++ Number of free blocks = 63625
$
```

Now, try the following from your terminal shell (`bash`).

```
$ chmod 755 barsh
$ ./barsh
```

It should run like `foosh`. Nice efforts… So good!

Then run `diskmanager` in the removal mode, and your virtual file system vanishes. All efforts wasted… So bad!