ForthOS Filesystem

Introduction

This section assumes you have the system source installed on a ForthOS partition, and that you're booted back into ForthOS. You're now ready to use the ForthOS filesystem to access this data.

Overview

In classic Forth, your disk devices appear as simple, contiguous runs of block numbers; each block was 1024 bytes in size. If you wanted a particular piece of data, you remembered that, say, 100 was the location for your source code.

When writing source code in classic Forth, your source fits in a single screen organized as 64 columns by 16 rows (thus, 1024 bytes of source). However, Forth source code was not generally commented in this screen; instead, at a fixed offset above your source screen number was a shadow screen. The editor could flip from source to shadow at a keystroke, so your comments tended to "shadow" the shape of the source you were commenting.

Unlike most POSIX-oid Forth systems, ForthOS is still block based, and still uses shadow screens for commenting source code. However, to reflect the relentless march of technology, ForthOS blocks are now 4096 bytes instead of 1024, with a source screen size of 2000 bytes (80 columns by 25 lines). The source code "shadow" screen is therefore on the same block, in the next 2000 bytes of the screen. The last 96 bytes of a source block are reserved for the ForthOS filesystem (which will be described shortly).

Basic use of the filesystem

A filesystem is rooted at a particular block number; initially, you'll be using the source code you installed at block 10000. First you set this block as the root of the filesystem, which also causes a quick check to be done to verify the block actually appears to hold a filesystem:

```
10000 fs.root!
```

(If you get an error, you've somehow managed to not put the source filesystem blocks into the right place on your disk.)

In addition to setting the top of the filesystem, it also sets your current working directory to the top. You can see what's there with:

```
fs.ls
```

You should see both "src" and "kernel" directories in the top of this directory. You can move down to a directory with:

```
fs.cd kernel
```

You move back to the top with:

```
fs.cd/
```

Accessing files

In the fs.ls output, you might have noticed that the "block number" of both files and directories is included in the listed output. This is the literal block number, and is the "real" way that any contents is accessed in ForthOS. The filesystem merely automates the lookup of this number, but you're free to supply it yourself. For instance, if you wanted to look at the first block of source for the kernel, your command sequence could be:

```
fs.cd/
fs.cd kernel
fs.open base list
```

But you could just as properly access it with:

```
10002 list
```

Thus, you can use the ForthOS filesystem to organize source files and directories, but its underlying action is always to simply push the block number on the stack. "list" takes that block number, as does the editor.

Creating files and directories

In most UNIX-like systems, files are almost like leaves, fluttering in the wind with their storage coming and going, shrinking and growing. The ForthOS filesystem is much more like an old-school IBM mainframe system; files and directories are created with the expectation that they'll be staying around. When you create either a file or a directory, you specify the maximum size. The storage is carved out right there and then, and thereafter "file size" is merely a notation local to the file or directory itself.

On the minus side, the filesystem is spectacularly ill-suited for use in applications where the number and size of files is unknown and varies widely. If you're going to undertake such an application under ForthOS, you're much better off to use a big chunk of blocks (which can all live happily under a filesystem "file", if you don't mind leaving the last 96 bytes of each block to the filesystem). On top of that chunk you can implement whatever fancy allocation management you need.

On the plus side, a ForthOS filesystem is not a web of complex data structures, with the web changing constantly as files and directories come and go. Typically, when you edit a source block, you'll modify the source and write it back into place--no filesystem metadata is touched. Even when you, say, grow the file by a block, you're only touching a few, simple data structures--all local to that specific file. Because of this, the filesystem is very rugged, and very resistant to corruption. This contiguous allocation regime also helps with performance--if you're processing a particular file, you are accessing blocks which physically exist all in a row on the disk.

Creating a directory

Let's say that you're going to create a directory for your own private work. In the ForthOS filesystem, a directory is for all intents and purposes its own mini-filesystem. Thus, when you create a subdirectory, you are really taking a chunk of storage from the parent directory, and formatting it as a sub-filesystem within which files and (sub-)subdirectories can be created (taking, in their turn, storage from this directory).

Actually creating a directory is easy enough:

```
fs.cd/
200 fs.mkdir private
```

This creates a directory with 200 blocks; one block is used to hold the directory contents, the rest are "free" and available for use in creating files and directories within. If you follow this with "fs.ls" you'll see you've created a directory with the name "private" in the filesystem.

Create a file

When you're in a particular directory, you create a file with:

```
fs.cd private   10 fs.creat myFile
```

This takes 10 blocks from the free storage of the containing directory, and creates a file with an initial size of 1 block, and 9 blocks to be used as needed to increase the size of the file. The starting block number is, of course, recorded in the directory entry. But it is also passed back on the stack, as you'll often want to invoke the editor on the newly created file. Thus "20 fs.creat myFile v" creates "myFile" and launches the full-screen editor on this file.

Loading source from a file

If you have edited the block(s) of your file, you probably now want to load that source.

```
fs.load myFile
```

This loads, in order, all the blocks of the file. If the file has only its initial one block, that one block will be loaded. However, if you've grown the file to three blocks, fs.load will load the three blocks successively. Internally, fs.load turns into a call to forth.thru with the appropriate block numbers.