Filesystem

Implement a simple file system based on a tree of inodes with direct references to data blocks. The interface to the block device that will store your file system, as well as the directory, file, and filesystem operations you must implement, are declared in this header:

fs.h

```
#include <stdlib.h>

enum fs_inode_type {
        FS_INODE_FILE,
        FS_INODE_DIRECTORY,
};

struct block_device {
        size_t block_size;
        size_t block_count;
        void (*read)(struct block_device *, size_t, void *);
        void (*write)(struct block_device *, size_t, const void *);
};

void fs_format(struct block_device *device);
size_t fs_root(struct block_device *device);
void fs_resize(struct block_device *device, size_t inode, size_t size);
size_t fs_read(struct block_device *device, size_t inode, size_t offset,
               size_t size, char *data);
size_t fs_write(struct block_device *device, size_t inode, size_t offset,
                size_t size, const char *data);
size_t fs_find(struct block_device *device, size_t inode, const char *name);
size_t fs_create(struct block_device *device, size_t inode, const char *name,
                 enum fs_inode_type type);
void fs_delete(struct block_device *device, size_t inode, const char *name);
```

A barebones `fs.c` file has been provided that includes block formats and comments about the purpose of each function. A test program that exercises the various library functions has also been provided in `main.c`; build the whole thing into a program called `fs` with this Makefile:

Makefile

```
CFLAGS=-Wall -g

OBJECTS=\
  fs.o \
  main.o

HEADERS=\
  fs.h

fs: $(OBJECTS)
fs.o: fs.c $(HEADERS)
main.o: main.c $(HEADERS)

.PHONY: clean
clean:
        rm -f fs $(OBJECTS)
```

The output should look like this:

```
$ ./fs
Hello, world!
world
```

Obviously, the test cases do not even exercise all of the functions; you are expected to write additional tests to convince yourself that your filesystem works. For instance, to test that `fs_delete` actually deletes recursively, you might create a copy of the test suite with a block device with only seven blocks—for the superblock, root inode, directory data for one child, subdirectory inode, directory data for its child, a file's inode, and a data block—create the subdirectory and file within it with one block of contents, delete the subdirectory, and try to recreate everything, which would only work if all of the blocks were correctly reclaimed onto the free list.

Tips

To allocate a temporary block, but be able to treat it as a particular structure (say, `struct superblock_data`, use something like `struct superblock_data *sb = malloc(bd->block_size)`. Remember to `free` everything you `malloc`!

To compute how many blocks are required to store the contents of a file, use `(inode->size + bd->block_size - 1) / bd->block_size`. To compute which block a particular byte `i` falls into, use `i / bd->block_size`. To compute its offset within that block, use `i % bd->block_size`.

Resources

When you have finished, submit your `fs.c` on CMS.