

CSCE 455/855 Distributed Operating Systems

Spring 2001
Steve Goddard

Programming Assignment 2 (PA2), February 14

Due: 6:00pm, Friday, March 2

The focus of this semester's Distributed Operating Systems course is an implementation of the FTFS (Fault Tolerant Distributed File-System). Fault tolerance is achieved through block-level file replication in FTFS. This assignment, however, simplifies that problem by replicating the entire file on all servers rather than portions of the file on different servers. While file-replication is simpler than block replication, this assignment will still require solving some of the complex issues related to block-level file replication in a distributed system.

The assignment is constructed in two parts. In the first part, you will add `mkdir()` and `lseek()` to the simple FTFS API that you implemented in Part I of HW1. In the second part, you will implement the FTFS API such that the file is replicated on n different servers that comprise a FTFS group. File requests to create or modify a file are sent to all servers in the FTFS group. The clients and servers may (but need not) be running on separate machines.

Part I: Extend your FTFS API for a single process

Add the functions `mkdir()` and `lseek()` to your FTFS API that is implemented as C or C++ function calls compiled into one or more `.o` files and linked into a user program to access the FTFS file system. The new functions to be implemented are:

```
/* see man 2 mkdir for details on the operation of this function */
int FtfsMkdir(const char *pathname, mode_t mode);

/* see man 2 lseek for details on the operation of this function */
off_t FtfsLseek(int fildes, off_t offset, int whence);
```

The above two functions should be added to those previously implemented in the API:

```
/* see man 2 open for details on the operation of these functions */
int FtfsOpen(const char *pathname, int flags);
int FtfsOpen(const char *pathname, int flags, mode_t mode);
int FtfsCreate(const char *pathname, mode_t mode);

/* see man 2 read for details on the operation of this function */
size_t FtfsRead(int fd, void *buf, size_t count);

/* see man 2 write for details on the operation of this function */
size_t FtfsWrite(int fd, void *buf, size_t count);

/* see man 2 close for details on the operation of this function */
size_t FtfsClose(int fd);

/* see man 2 unlink for details on the operation of this function */
size_t FtfsUnlink(const char *pathname);
```

Make sure you test this API thoroughly and document your tests. This part is not meant to be difficult. Its purpose is to familiarize you with the `mkdir()` and `lseek()` functions. The next part will also be based on this extended API. However, it will require file replication on multiple servers.

Part II: Multiple File Servers and File Replication

Choose either the socket-based message passing paradigm or the RPC paradigm to implement the FTFS API with full file replication on every server in an FTFS group. You will need to define an FTFS group in which each member (machine) may be both a client and a server, as shown in Figure 1 of Evans' thesis. You should assume the following:

- File descriptors are global. This means that the file descriptor returned from a `FtfsCreate()` or `FtfsOpen()` is recognized by each server when subsequent `FtfsWrite()` calls are made.
- `FtfsRead()` calls may be performed locally.
- `FtfsCreate()`, `FtfsWrite()`, `FtfsUnlink()`, and `FtfsMkdir()` must make updates on all servers in the FTFS group.
- There may exist multiple simultaneous readers and writers per file.
- There are multiple writers, writing to different files, at the same time.
- No faults occur.

Grading Policy for Programs

The programs you hand in should work correctly and be documented. When you hand in your programming assignment, you should include:

1. A program listing containing in-line documentation.
2. A separate (typed) document of approximately two pages describing the overall program design, a verbal description of "how it works" including the basics of what the system is doing underneath, and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made).
3. A separate description of the tests you ran on your program to convince yourself that it is indeed correct. Also describe any cases for which your program is known not to work correctly.
4. A make file that compiles your program(s).

Please hand in your source files for all parts of this project.

The program should be neatly formatted (*i.e.*, easy to read) and structured and documented. Use the handin program to submit your program(s) for grading. This is assignment 2. Your grade will be determined as follows:

```

Program Listing
  works correctly 40%
  in-line documentation 15%
  quality of design 25%
Design Document 15%
Thoroughness of test cases 05%
```