Project 3 Overview

Overview

- In this project you will create a filesystem
- The filesystem will support:
  - Undo logging
  - Redo logging
  - Crash recovery

gtfs API

```
gtfs_t* gtfs_init(std::string directory, int verbose_flag);
int gtfs_clean(gtfs_t *gtfs);
int gtfs_clean_n_bytes(gtfs_t *gtfs, int bytes);

file_t* gtfs_open_file(gtfs_t* gtfs, std::string filename, int file_length);
int gtfs_close_file(gtfs_t* gtfs, file_t* fl);
int gtfs_remove_file(gtfs_t* gtfs, file_t* fl);

char* gtfs_read_file(gtfs_t* gtfs, file_t* fl, int offset, int length);
write_t* gtfs_write_file(gtfs_t* gtfs, file_t* fl, int offset, int length, const char* data);
int gtfs_sync_write_file(write_t* write_id);
int gtfs_sync_write_file_n_bytes(write_t* write_id, int bytes);
int gtfs_abort_write_file(write_t* write_id);
```

Initialization and on disk log management

File creation/removal

Reading and Writing data

Write Visibility Scenarios

A process can read its own writes, without syncing

```
int main(){
        gtfs_init(dir_abs_path)
        gtfs_open_file(“file1.txt”, length)
        gtfs_write_file(“hello1”, offset, n_bytes)
        data * = gtfs_read_file(offset, n_bytes)
        assert(strcmp(data, “hello1”) == 0)
        return 0
}
```

A process can abort a write, and read a version of the file as if that write didn’t happen

```
int main(){
        gtfs_init(dir_abs_path)
        gtfs_open_file(“file1.txt”, length)
        wr1 = gtfs_write_file(“hello1”, offset, n_bytes)
        wr2 = gtfs_write_file(“hello2”, offset, n_bytes) //overwrite
        data * = gtfs_read_file(offset, n_bytes)
        assert(strcmp(data, “hello2”) == 0)
        gtfs_abort _write(wr2)
        data * = gtfs_read_file(offset, n_bytes)
        assert(strcmp(data, “hello1”) == 0) //read the first write at this offset
        return 0
}
```

A process’s write only becomes visible to other processes, if the write is sync’d.

```
//process 1

int main(){

        gtfs_init(dir_abs_path)

        gtfs_open_file(“file1.txt”, length)

        wr1= gtfs_write_file(“hello1”, offset, n_bytes)

        gtfs_sync_write_file(wr1) //added to on disk log



        return 0

}
```

A process's write only becomes visible to other processes, if the write is sync'd.

```
//process 1

int main(){

        gtfs_init(dir_abs_path)

        gtfs_open_file("file1.txt", length)

        wr1= gtfs_write_file("hello1", offset, n_bytes)

        gtfs_sync_write_file(wr1) //added to on disk log


        return 0

}
```

```
//process 1 has exited, process 2 now

int main(){

        gtfs_init(dir_abs_path)

        gtfs_open_file("file1.txt", length)

        data * = gtfs_read_file(offset, n_bytes)

        assert(strcmp(data, "hello1") == 0)

        return 0

}
```

A process’s write that isn’t sync’d will not be visible from a second process

```
//process 1
int main(){
        gtfs_init(dir_abs_path)
        gtfs_open_file(“file1.txt”, length)
        wr1= gtfs_write_file(“hello1”, offset, n_bytes)

        //we did not call gtfs_sync_write
        return 0
}
```

A process’s write that isn’t sync’d will not be visible from a second process

```
//process 1
int main(){
        gtfs_init(dir_abs_path)
        gtfs_open_file(“file1.txt”, length)
        wr1= gtfs_write_file(“hello1”, offset, n_bytes)

        //we did not call gtfs_sync_write
        return 0
}
```

```
//process 1 has exited, process 2 now
int main(){
        gtfs_init(dir_abs_path)
        gtfs_open_file(“file1.txt”, length)
        data * = gtfs_read_file(offset, n_bytes)
        assert(strcmp(data, “”) == 0) //uninitialized reads return “”. See API spec
        return 0
}
```

Log management on disk / in memory

Doing a clean(), should apply the on disk log to the file, and truncate the log size to 0

```
total 4
-rwxrwxr-x 1 daniel daniel 100 Mar  9 10:38 additional1.txt
-rwxrwxr-x 1 daniel daniel  28 Mar  9 10:38 additional1.txt.log
total 4
-rwxrwxr-x 1 daniel daniel 100 Mar  9 10:38 additional1.txt
-rwxrwxr-x 1 daniel daniel   0 Mar  9 10:38 additional1.txt.log
```

gtfs_clean() //applies and truncates log
Reflected in file size

Test Cases

- The starter code comes with test cases to check for expected filesystem behavior
  - Write to file, then read that data back from the same process
  - Write to file, then overwrite that write, abort the latter write, then read the first write from the same process
  - Write to file from one process:
    - Then sync that write, and read back the same data from a second process
    - Then abort that write, and check that the data cannot be read from a second process
  - Checks for on-disk log appending: syncing writes should increase the on disk size of redo-log
  - Checks for log compaction: cleaning the on-disk log should reduce their size (either to 0, or by some partial amount if using the clean_n_bytes() api)

Other Notes

- You can assume that only one process at a time will have the file system open
  - That is, when a process calls ‘gtfs_init’ you can assume as a precondition that no other running process has issued this API call
- Uninitialized reads that are in file bounds return “”
  - Not the best interface design, but the tests assume this