CS 3411 Project 2: I/O

Due Date: Feb. 16 (Wednesday), 11:00pm
Collaboration Policy: Empty Hands

Motivation

The UNIX file abstraction makes it difficult to remove data from a file and to insert data, except at the current end of file. By this abstraction, when data is written to a file anywhere except at the end, existing data is overwritten. In this project, you will write routines that allow a user to insert or remove data from a file through a single function. This project will give you experience with routine use of the system call interface for file I/O.

Requirements

You will write two routines: `insert`, which inserts data at specified location within a file, and `extract` which removes data from a specified location and returns the removed data to the caller. A detailed specification for each routine follows.

`int insert(int fd, void *buf, size_t bytes, size_t offset)`
Inserts the contents of input buffer at the specified location within the file given by `fd`. All data originally in the file survives the insertion. The parameter `buf` points to a buffer from which the data to be written should be taken. The parameter `bytes` contains the number of bytes that should be written. When executed successfully, the file size will be increased by `bytes` bytes. The parameter `offset` indicates the offset at which the write should begin. (An offset of zero indicates the first byte written becomes the first byte of the file.) A negative offset or one that is beyond the end of the file is an error. The return value is the number of bytes actually written, or a negative value on error.

`int delete(int fd, size_t bytes, size_t offset)`
Deletes data at the specified location. The parameter `bytes` contains the number of bytes that should be deleted. When executed successfully, the file size will be reduced by `bytes` bytes. The parameter `offset` indicates the offset at which the delete should begin. (An offset of zero indicates the first byte deleted is the first byte of the file.) A negative offset or one that is beyond the end of the file is an error. The return value is the number of bytes actually deleted, or a negative value on error.

`int extract(int fd, void *buf, size_t bytes, size_t offset)`
Deletes data at the specified location and writes it to the specified buffer. The parameter `bytes` contains the number of bytes that should be deleted. When executed successfully, the file size will be reduced by `bytes` bytes. The parameter `offset` indicates the offset at which the delete should begin. (An offset of zero indicates the first byte deleted is the first byte of the file.). A copy of the deleted data is written to the location given by `buf`. A negative offset or one that is beyond the end of the file is an error. The return value is the number of bytes actually deleted, or a negative value on error.

Error Handling

Assume that `fd` is the return value from an earlier successful call to open. Your routines should return an error if the file specified by `fd` cannot be read or written as required by the specified operation. Note that calls to `read` and `write` will detect this error and return a value that can be propagated to the user. Do not open files within the `insert` and `extract` routines.

It is not an error if the specified operation results in a **successful** insertion or removal of fewer than the specified number of bytes. The value returned to the user should accurately reflect the number of bytes inserted or removed

successfully.

Assume that `buf` has been allocated previously. It is not required to detect whether the input value for `buf` is valid. Do not allocate memory within these routines that is not also deallocated within the routine.

It is an error if the `bytes` parameter to `delete` or `extract` is larger than the size of the file.

Errors from system calls made within these routines should result in an error return from your library routine.

You do not need to consider the effects of other processes operating on this file concurrently with these routines.

Consider the time it takes to execute a system call as you write these routines. Do not process data character by character.

Notes

- Offset 0 precedes the first byte of the file. Offset 1 follows the first byte of the file.
- The size of a file can be modified using the `truncate` and `ftruncate` system calls. These are described in section two of the manual pages.
- If a file has 10 bytes, an insert at offset 10 (at current end of file) is successful. An insert at offset 11 is an error.
- You must use the system call file interface. Use of the stdio library (fopen, fread, fwrite etc.) **is not allowed** and will result in a grade of 0. If you think you have to use a routine that is not part of the system call interface, get in touch with me before using the routine.

Tests

Your submission must include at least one test program named `testops.c` that invokes each routine. You may of course want to create multiple test programs.

Makefile

Each routine should be implemented in a separate file, `insert()` in `insert.c`, `delete()` in `delete.c`, and `extract()` in `extract.c`.

Also generate a header file named `ufsext.h` that a user of your routines will include in his code. The header will contain function declarations and any other definitions required to use your routines. The grader will include this header in his test source code and link against your libraries (described below) in order to create the test executables.

Write a test program called `testops.c` that exercises all of these routines.

Submit a makefile that builds both `insert.o`, `delete.o` and `extract.o` from their corresponding C files. You makefile should also build test program `testops`. The command `make ops` should create the `insert.o`, `delete.o` and `extract.o`. The command `make tests` should compile all your test code. The makefile default should be to create all the object files and test code. The command `make clean` should remove all object code and executables.

For convenience, please keep all your code in the same directory so that the grader can find it easily.

Submission

The files described above should be collected into a tar file named `ufsext.tgz`. Typing the commands: `gtar xvf ufsext.tgz; make;` should produce the libraries and executables named above. See the teaching assistant if you are unsure how to create the required tar file.

Pay close attention to the subroutine and file names given in this specification, as well as the specified order of parameters. If the grader cannot link a test against your routines, or a test fails because your implementation deviates from this specification in some obvious way, the test grade may be zero. These requirements are likely to be strictly enforced due to the class size.

The submission is due Wednesday, Feb. 16 at 11pm. Include a single README gives the number of slip days that you used for the project and that describes each file in your project directory. Your README should briefly describe the tests that you developed.

Collaboration

Empty hands discussions are allowed for this project.