UNIVERSITY OF COPENHAGEN
Department of Computer Science
Robert Glück glueck@diku.dk
Jost Berthold berthold@diku.dk

March 2, 2012

G-Exercise 4 for "Operating System and Multiprogramming", 2012

Hand in your solution by uploading files in Absalon before Mar.13

Exercise Structure for the Course

Please see the introduction on exercise 1 for information about the exercise structure. Solutions to G-exercises are in general handed in by uploading an archive file (zip or tar.gz) to Absalon. When you upload your solution to Absalon, please use the last names of each group member in the file name and the exercise number as the file name, like so:
`lastname1_lastname2_G3.zip` or `lastname1_lastname2_G3.tar.gz`.
Use simple txt format or pdf for portability when handing in text, and comment your code.

Important changes to the schedule: Please note that, due to timing constraints, submission dates for exercise 4 and 5 are different from the ones before:

- Exercise 5 (the next exercise) has a shorter editing time of only 7 days, and no resubmission. The exercise will be published on Friday, March 9 (as usual), but has to be handed in 8 days later, on Saturday, March 17 (until midnight).
- The deadline for resubmission of exercise 4 (this exercise) is also Saturday, March 17.

This is because the teaching assistants need to correct the exercise until March 20.

About this Exercise

Topics

This exercise is about using and implementing file systems, and uses the Buenos exercise system. The first task is to implement the necessary system calls which allow one to access the virtual file system. In Task 2, you are asked to extend the Buenos-provided file system TFS to a more usable one, *Flat FS*. Task 3 (independent of 2) is to provide a small shell for file system access from the command line.

What to hand in

Please hand in your solution in a single archive file (zip or tar.gz format) which contains:

1. A source tree for Buenos which includes your work on tasks 1 and 2.
2. A directory containing C code and a Makefile for the shell developed in task 3.
3. A report in pdf or txt format.
   Your report should discuss the design decisions you took for your file system modifications (in addition to commenting the C code), and answer the additional questions.

Tasks

1. **System Calls for File Handling**

   System calls `syscall_read` and `syscall_write` have been implemented in an earlier exercise, but they only worked on `stdin` and `stdout`. In the first exercise, these system calls should be extended to work with other file handles as well (`stdin` and `stdout` are just special file handles). Other system calls are added, in order to open, close, create, and delete files.

   (a) Implement the system calls described below (following the Buenos roadmap, section 6.4 and chapter 8). Use only calls to the virtual file system layer in Buenos (declared in `fs/vfs.h` and implemented in `fs/vfs.c`). Do not use TFS functions directly.

   - `int syscall_open(const char *pathname)`
     A call to `syscall_open` prepares the file indicated by `pathname` for reading and writing (read-only mode is not supported). The `pathname` includes a volume name and a file name (for instance `[root]a.out`). Returns a positive number bigger than 2 (serving as the file handle to the open file) when successful, or a negative number on error.
   - `int syscall_close(int filehandle)`
     This system call has the effect to invalidate the file handle passed as a parameter. It cannot be used in file operations any more afterwards.
   - `int syscall_create(const char* pathname, int size)`
     Creates a new file with name `pathname` of size `size`. Returns 0 on success, or a negative value on error.
   - `int syscall_delete(const char *pathname)`
     Deletes the file with name `pathname` from the file system, fails if the file is open. Returns 0 on success, or a negative number on error.
   - `int syscall_seek(int filehandle, int offset)`
     Set the reading or writing position of the open file represented by `filehandle` to the indicated *absolute* offset (in bytes from the beginning). Returns 0 on success, a negative number on error. Seeking beyond the end of the file sets the position to the end and produces an error return value.
   - `int syscall_read(int filehandle, void *buffer, int length)`
     Reads at most `length` bytes from the file identified by `filehandle` (at the current file position) into `buffer`, advancing the file position. Returns the number of bytes actually read (before reaching the end of the file), or a negative value when an error occurred.
   - `int syscall_write(int filehandle, const void *buffer, int length)`
     Writes `length` bytes from `buffer` to the open file identified by `filehandle`, starting at the current position and advancing the position. Returns the number of bytes actually written, or a negative value on error.

   (b) Adapt your user process implementation to work correctly with the modified system calls. At the very least, you will need to manage a list of open files.

2. **Indirect block pointers and file resizing in the Buenos inodes**

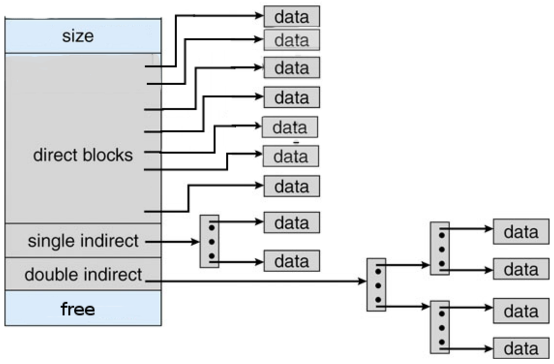
   The file system TFS that comes with Buenos imposes very tight limits on the size of files in the file system. This limitation can be alleviated by using inodes with indirect pointers, in a modified file system *FlatFS*.

   Do not modify TFS, but implement your modifications as a new type of file system. Copy `fs/tfs.h` and `fs/tfs.c` and rename them to `fs/flatfs.[ch]`, and include FlatFS in the list of file systems in `fs/filesystems.c`. In this way, you can continue using the original TFS to copy (small) files to and from the simulated Buenos system using `tfstool`.

   (a) **Using indirect block pointers in the inodes**

   The Buenos TFS inodes contain a maximum of 127 block pointers for a file, which effectively limits the file size to 65 KB.

   i. Implement a solution using indirect block pointers, similar to the one used in Unix inodes [SGG, p.477f]. In a FlatFS inode, the first 7 pointers are direct pointers, followed by one single-indirect pointer, followed by one double-indirect pointer. (The remaining block pointer fields are left blank).

   ii. What is the maximum file size implied by this scheme for the block pointers?

   (b) **Support for file resizing**

   The original TFS does not support file resizing. The file size must be specified when creating a file, and cannot be modified any more. In order to really take advantage of the modifications in 2a, FlatFS should be able to resize files. Having this, it is possible to always create files of length 0 and extend them afterwards.

   In your modified file system from 2a (or in the original TFS, if you did not work on 2a), implement a mechanism which extends a file when a program writes data beyond its original end.

3. Implement a small userland shell, so the Buenos user can explore and modify the file system of a volume directly by typing in commands. Your shell should provide at least the following commands:

`ls`	lists the contents of a volume (1 argument)
`cp`	copies the contents of one file to another (new or overwritten) file (2 arguments)
`show`	writes the contents of a file to stdout (1 argument)
`rm`	deletes a file from the file system if it exists (1 argument)
`touch`	creates a new empty file (1 argument)
`exit`	terminates the shell (no arguments)

   The commands should have the same basic functionality as in a linux shell. However, you don't need to implement any options.