Ernesto Soltero
Max Roth
Nick Jenis

Manual Pages (for syscalls)

Name: exit - terminate the calling process
Usage: exit
Description: exit deallocates the stacks space and deallocates the pcb for the given process. If this was the running process, dispatch a new one. exit returns void

Name: spawnp - create a new process running the specified program
Usage: spawnp void (*entry)(void), prio
Description: spawnp creates a new process based off the given one. This new process inherits the running process's id and sets that to its parent pid. If the creation is successful, spawnp returns the pid of the created process otherwise spawnp returns -1.

Name: sleep - put the current process to sleep for some length of time
Usage: sleep time_in_ms
Description: sleep takes the given time in ms and moves the currently running process to the sleep queue. If the given time is 0, sleep just preempts the process.

Name: read - read bytes from the specified input channel
Usage: read int fd, void *buf, int count
Description: read reads up to the count number of bytes or however many are currently available, whichever is smaller. The file descriptor can either be the the console or serial terminal. read returns the number of characters read.

Name: write - write bytes to the specified output channel
Usage: write int fd, void *buf, int count
Description: write sends a count number of bytes from the buffer to the given file descriptor. The file descriptor can either be the serial terminal or the console. If count is 0, write sends the entire NULL-terminated buffer. write returns the number of characters written.

Name: get_process_info - retrieve information about a process
Usage: get_process_info int code, int16_t pid
Description: get_process_info returns details of the pcb for a given process or the currently running process if the given pid is 0. get_process_info can return pid, ppid, state, wakeup, prio, quantum, or the default quantum. If the given pid or code could not be found, get_process_info returns -1.

Name: get_system_info - retrieve information about the system
Usage: get_system_info int code

Description: get_system_info returns system details based on the given code. get_system_info can return details on the system time, number of processes, and max number of processes. If the code is not found, get_system_info returns -1.

Name: create_file - creates a file in the file system
Usage: create_file char* filename
Description: create_file attempts to create a new file in the file system with the given name. If a file with this name already exists, create_file returns 1. For success, create_file returns 0

Name: delete_file - deletes a file from the file system
Usage: delete_file char* filename
Description: delete_file attempts to delete a file from the file system. If this operation is successful, delete_file returns 0, 1 for failure.

Name: read_file - read from a file in the file system
Usage: read_file char* filename
Description: read_file attempts to read from a file in the file system. If the file exists, read_file returns a pointer a string of the data. If the file doesn't exist, 0 is returned.

Name: write_file - write to a file in the file system
Usage: write_file char* filename, uint8_t size, char* buf, int doAppend
Description: write_file attempts to write to a file in the file system. If the file is found, write_file overwrites the existing data in the file of length size. If the parameter doAppend is set, write_file appends data to the back of the file. If this operation is successful, write_file returns 0, or 1 for failure.

Name: list_files - lists the files in the file system
Usage: list_files
Description: list_files scans through the file system based on the current directory and returns a pointer to a string of the file names.

Name: cd - changes the current directory in the file system
Usage: cd char* new_directory
Description: cd attempts to change the working directory to the new one given. If this operation is successful, cd returns 0, or 1 for failure

Name: bogus - catch all for incorrect syscalls
Usage:
Description: This is a fake syscall which operates as a catch all for incorrect syscalls.

Internals Description

Module - Shell
Description: The shell operates as a medium level process, running from the start of the OS to the end. The shell uses a primary read loop which waits on the user to enter a command and parameters from the terminal. The shell is designed to only allow 1 space in between parameters. To perform a command with a longer or multiple spaced parameter, such as a write_file, the shell uses a single quote ‘ to start and stop the longer parameter.
Once a command has been entered and parsed, the shell matches that with a list of built in commands. If a match is found, the shell executes the syscall for the specific command. The use of syscalls allows the user to interact with the file system and other system modules. If the command is not found, the shell simply loops to begin again.
The shell can also spawn new processes if that is desired.

Module - USB driver
Description: The usb driver is only part of a usb protocol (driver/host controller/usb core). Most of the work of the driver was in the header file because the whole system derives from the header file because it is highly dependent on the data structures provided by it. The usb protocol calls for very specific structures which include a urb (data block that gets passed), endpoints, configuration, interface, device, and descriptors. I lot of the work was in getting the data types correct in our structs. In our driver implementation there is functionality to write/read to files to our filesystem on the usb storage device. These devices take in a file and a buffer and input these into a urb and pass this to our usb message passing function as defined by the usb protocol. There is also functions to deal with registering/deregistering the driver and probing the driver so figuring out when a usb is connected/disconnected. Other than that our filesystem would be written to the usb and the only thing that would have to be modified is the starting point of where we would write/read files too.

Module - Simple File System (SFS)
Description: A simple implementation of a file system that allows for basic reading and writing of files in a directory structure. The file system works using three basic structures: one for a data block, one for an entry, and one representing the file table. The file table is the main structure for the file system. Everything in the file system is relative to the table, so SFS will work anywhere the file table structure is placed. For this project, it is placed at 1 gigabyte in RAM, but it can be instantiated anywhere the computer has read/write access to. The file table is made up of an array of all of the data block structures, an array of entry structures, a string representing the current working directory, and an integer representing the next data block open to write to. The entry structure either represents a file or a directory entry. It is composed of a type variable, indicating if the entry is a file or directory entry, an integer representing the first data block of data, an integer indicating the size of the file, and an array of characters (often cast to a string) representing the filename. The filename is especially important to the functionality of our filesystem, since it is one of the driving forces of the directory structure. The full filename is actually a file path, and is only accessible by its

shortened name when combined with the working directory path in methods to recreate the true filename, representative of the file path. This implementation can have a lot go wrong, however the filesystem is filled with logic and error checking to make sure this simple implementation works perfectly, appearing similar to other directory implementations while maintaining extreme simplicity. Finally, the data block structure is made up 512 bytes to store data, as well as a pointer to another data block that contains further data if a file is larger than 512 bytes, creating a linked list structure to store larger file's data. All operations are performed in separate methods executed by system processes (or their equivalent user processes when called from the shell). The main methods of note revolve around file manipulation, namely create, read, write, and delete.

Experiences

Good things:
Teamwork
Expanded our knowledge in low level systems

Bad things:
Underestimating the complexity of the project. Even implementing a simple feature was a lot harder due to the low level of this project.

When we started the project we wanted to do projects that would interact with each other so we decided on first doing a file system. Then we discussed between doing a floppy disk driver or a usb driver to store the files and we decided on doing a usb driver since no one uses floppy disks in the modern world. We started off by making a shell so we could interact with the system. Then we all started working on our file system and by the midpoint review we had some functionality working for the file system but we hadn't started on the usb driver so we split up. After a couple of days of research we found out how complicated a usb driver actually is and we also learned that we wouldn't be able to test it without also making a host controller driver and a hub driver. Nevertheless we felt like it was too late in the project to scrap it and start working on a floppy disk so we decided to go through with our original plan since we had already done a lot of research. So we made a lot of progress in the filesystem and some in the driver. The main work of the driver was getting the data structures for the usb 2.0 protocol correctly. All drivers that implement the usb protocol would derive from a single header file so its important to get the correct data types. We got a lot of functionality done for the file system and made good headway in our driver but we can't test the usb driver because we don't have the other components, so we can only assume what we have so far works because of the copious amount of research we did in getting the data types correct. So if we were to change anything, we would choose to implement a floppy driver over a usb since it would of been less complicated since we would interact with the PCI directly over the usb protocol. Overall we feel that we learned a lot about low level systems;although we didn't completely finish our project we know exactly what direction we have to take in order to complete our initial goals.