Team Member

- 109955038 Yuxuan Shui
- 109797586 Haochen Chen

Overview

This is an 80 points project.

Our work might be a little bit unusual because our kernel follows a micro-kernel design. Unlike what was taught in the course, our kernel only does the basic process isolation, address space management and IPC, (almost) everything else is in user space. Including the console and the filesystem (so make sure they are there when you do the test).

Although due to time constraint, lack of experience in micro-kernel design and various other reasons, the result is extremly buggy and feature incomplete. At least we learned a lot, and had fun (at the expense of sleep) in the process.

Filesystem structure

The root directory contains 2 subdirectories, /tarfs for the tarfs, /sata for filesystem on the hard drive. /tarfs is readonly, and /sata is (in theory) read/write.

In case you are not able to run your shell for testing (highly likely), at least you can try the 'cd', 'ls', 'touch' and 'cat' builtin commands in our shell to see the filesystem in motion.

Syscalls

syscalls implemented in libc

Those *syscalls* are implemented in libc as user space functions:

- read, write, open, readdir, writedir, opendir, lseek
- fork, execve (Yep, loading ELF is done in user space)
- execvp: This is implemented but doesn't work, so when you try to run programs, please use absolute path.
- chdir, getcwd (We cheated a little here)

Actual syscalls

These are some of the most important syscalls that are actually implemented in the kernel, with brief explainations:

- get_thread_info: Get the thread info (mainly registers) of current running thread.
- asnew: Create a new sub address space. Can create a COW snapshot of current address space.
- sendpage: Used to transfer memory range between address spaces.
- munmap: Unmap a memory range.
- create_task: Create a new task, take an address space and thread info.
- wait_on, wait_on_port: Wait on file descriptors or a port.
- connect_port, open_port, request, respond, get_response: Used for IPC between processes. This API is TCP-like and stateful. The kernel will try to avoid memory copy as much as possible. I admit this API is not well designed, but in my defense, this is my first try.

What is not implemented

- pipe: We had some code for a pipe server in user space, but sadly we don't have time to finish it.
- sleep: The wait_on syscall takes an argument for timeout, but we are short of time to actually implement that.
- brk: Memory is allocated via sendpage syscall.
- waitpid

Other notes

This kernel generates a lot of logs through serial port. So don't (or do, if you are interested) start qemu with serial port.

Conclusion

This little kernel is extremly fragile, so please test with care and love. Thanks!.