Rock’n Roll

Mathis Bouverot-Dupuis

TABLE OF CONTENTS

Project Outline

My initial goals :

- Simple OS (one-man project) : 32 bits, uniprocessor, not POSIX compliant.
- Runs on QEMU.
- Understand how components work together, keep every component simple and use basic (slow) data structures.

A few stories (don’t do the same mistakes as me) :

- Implement scheduling before anything else (filesystem...).
- Optimizations.
- Gdb with QEMU.

Bootloader

My own two-stage bootloader (GRUB too hard).
First stage : 512 byte binary loaded by the BIOS.
16-bit real mode, uses BIOS interrupts for I/O.

1. Initializes the machine (disk, etc.).
2. Finds the file containing the second stage on disk.
3. Loads the second stage (flat binary).

Second stage :

1. Enters 32-bit protected mode.
2. Gets more machine info (memory map, filesystem info, etc.).
3. Loads the kernel (as an ELF binary).

Threads

Thread abstraction :

- Code and execution context (registers, scheduling state).
- Scheduler : switches between threads.
- Operations : create(), exit(), join(), yield()

Scheduling :

- Switching/creating threads is a mess (evil stack manipulation).
- Simplest possible scheduler : no priorities.

Synchronization

Simple case : uniprocessor.
Two primitives : locks and events.

1. Locks : spinlocks and queuelocks (mutex).
2. Events : condition variables.

Can be used from kernel and user space (through system calls).
Also semaphores : never really understood them.

Process

- Manages resources (files, locks, etc) for a group of threads.
- Tree structure.
- Threads in same process share resources and address space.
- Operations : fork(), exec(), exit(), wait() (shell).
- Fork : loads ELF binaries (e.g. cd/ls).
- Fork/exec/exit with multiple threads : life is hard (e.g. critical sections).

Filesystem

Only one filesystem : EXT2.

- Initial disk image is also EXT2.
- Kernel manages access (system calls to read/create/listdir/etc).
- Parsing the EXT2 format : lots of work (at least easy disk image creation).
- No file synchronization (yet).
- Buffered block I/O : still a mystery for me.

Memory management

Paging : very basic.

- Every process has its own address space.
- Kernel at 3GB+, user space below.
- Frames are never swapped/freed.
- Uses the memory map created by the bootloader (theoretically at least).

Heap : also basic (linked lists of memory blocks).

Interrupts/drivers

External interrupts (keyboard, disk, etc) : much harder than they seem.

- Coordinate interrupts with threads (shared memory) ? (can't use locks, can't just disable interrupts).
- If functionality used in interrupts : have to clear IF when using it. (heap ? scheduler ?)
- Should have implemented a generic deferred interrupt mechanism.