MiniOS CS499 Fall 24

Ethan Jones, Lane Wright, Britton Pearce, Logan Cagle

Agenda

- Lay out the project goals, requirements, and deliverables.
- Demo the OS.
- Give a quick overview on some of the technical internals/achievements.
- Offer some advice to the customer.
- Reflect on the solution that has been delivered.

Project Goals and Requirements

- “Our group will deliver an i386 OS capable of running on QEMU; this can further be separated into three main deliverables: source code, kernel documentation, and a prebuilt .iso file”.
- Input/Output, Data, Security
- Virtual Memory, Interrupts, Processes/Scheduling, Filesystem, System Calls, Userspace Utilities.

Project Goals and Requirements

- “Our group will deliver an i386 OS capable of running on QEMU; this can further be separated into three main deliverables; source code, kernel documentation, and a prebuilt .iso file”.
- Input/Output, Data, Security
- Virtual Memory, Interrupts, Processes/Scheduling, Filesystem, System Calls, Userspace Utilities.

Technical Achievements

- OS Development requires quite a few complex components:
- Bootloader (GRUB), build system (linker, cross-compiler, qemu), GDT, IDT, interrupt handlers, multiple memory allocators, keyboard driver, round robin scheduler, blocking logic, ipc (ringbuffers), system calls, simple paging, terminal output code, libc, hardware interrupt controller configuration, stack management, shell and other programs…
- Some of these are “better” than others, but they’re all there.

Syscalls

- `send`, `recv`, `exit`, `spawn`, `spawn_bg`, `pid`.
- The programmer tailors a structure according to one of the above, triggers a software interrupt, and the OS then handles the syscall.
- `send` sends a message over IPC, etc.
- We’ve produced an accompanying developer document that introduces each syscall and how run programs on MiniOS.

Reflection on Syscalls

- At the end when attempting to debug bidirectional IPC, we realized we had to extend them to include “sender” data.
- Didn’t implement the POSIX interface; prevents porting of real applications.
- Designing this interface well is *extremely important*; whoever purchases the OS will write programs via this interface.

Processes and Scheduling

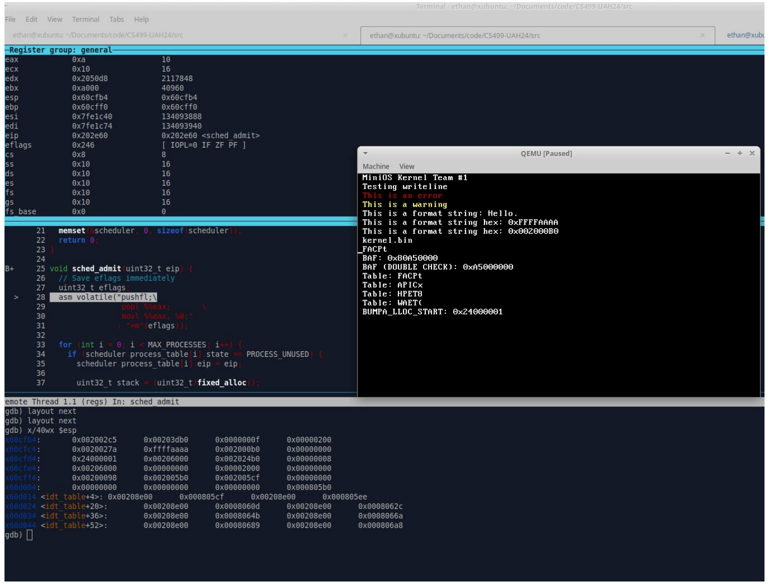
- As show by the demo, we can spawn processes that can block or run unblocked.
- Accomplished with a round-robin scheduler that records what specifically a process is blocking on (whether that be STDIN or waiting on an IPC message).
- Was difficult to debug: non-deterministic, multi-processing, interacts with the underlying hardware, have to reason about the code the compiler produces, etc.

Testing and Debugging

- How to debug non-deterministic, low-level/embedded, scheduling OS code?

Testing and Debugging

- How to debug non-deterministic, low-level/embedded, scheduling OS code?

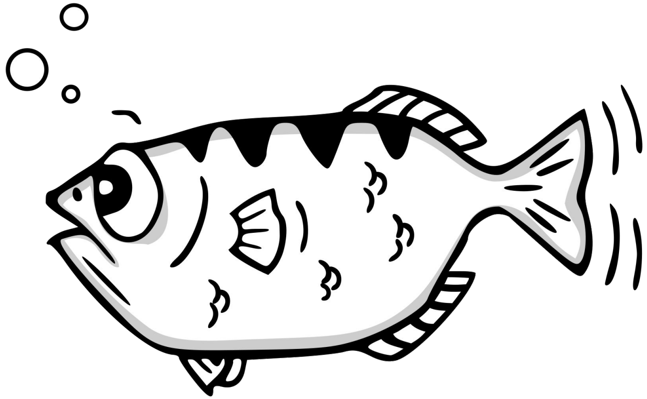

Terminal - ethan@xubuntu: ~/Documents/code/CS499-UAH24/src
File Edit View Terminal Tabs Help
ethan@xubuntu: ~/Documents/code/CS499-UAH24/src
ethan@xubuntu: ~/Documents/code/CS499-UAH24/src
QEMU [Paused]
Machine View

```
Register group: general
eax            0xa                 10
ecx            0x10                16
edx            0x2050d8            2117848
ebx            0xa000              40960
esp            0x60cfb4            0x60cfb4
ebp            0x60cff0            0x60cff0
esi            0x7fe1c40           134093888
edi            0x7fe1c74           134093940
eip            0x202e60            0x202e60 <sched_admit>
eflags         0x246               [ IOPL=0 IF ZF PF ]
cs             0x8                 8
ss             0x10                16
ds             0x10                16
es             0x10                16
fs             0x10                16
gs             0x10                16
fs_base        0x0                 0
```

```
      memset(&scheduler, 0, sizeof(scheduler));
      return 0;
    }

B+  void sched_admit(uint32_t eip) {
      // Save eflags immediately
      uint32_t eflags;
 >    asm volatile("pushfl;\
                    popl %%eax;       \
                    movl %%eax, %0;"
                   : "=m"(eflags));

      for (int i = 0; i < MAX_PROCESSES; i++) {
        if (scheduler.process_table[i].state == PROCESS_UNUSED) {
          scheduler.process_table[i].eip = eip;

          uint32_t stack = (uint32_t)fixed_alloc();
```

```
emote Thread 1.1 (regs) In: sched_admit
gdb) layout next
gdb) layout next
gdb) x/40wx $esp
x60cfb4:       0x002002c5      0x00203db0      0x0000000f      0x00000200
x60cfc4:       0x0020027a      0xffffaaaa      0x002000b0      0x00000000
x60cfd4:       0x24000001      0x00206000      0x002024b0      0x00000008
x60cfe4:       0x00206000      0x00000000      0x00002000      0x00000000
x60cff4:       0x00200098      0x002005b0      0x002005cf      0x00000000
x60d004:       0x00000000      0x00000000      0x00000000      0x000805b0
x60d014 <idt_table+4>: 0x00208e00      0x000805cf      0x00208e00      0x000805ee
x60d024 <idt_table+20>:        0x00208e00      0x0008060d      0x00208e00      0x0008062c
x60d034 <idt_table+36>:        0x00208e00      0x0008064b      0x00208e00      0x0008066a
x60d044 <idt_table+52>:        0x00208e00      0x00080689      0x00208e00      0x000806a8
gdb) 
```

```
MiniOS Kernel Team #1
Testing writeline
This is an error
This is a warning
This is a format string: Hello.
This is a format string hex: 0xFFFFAAAA
This is a format string hex: 0x002000B0
kernel.bin
_FACPt
BAF: 0x80A50000
BAF (DOUBLE CHECK): 0xA5000000
Table: FACPt
Table: APICx
Table: HPET8
Table: WAET(
BUMPA_LLOC_START: 0x24000001
```



Testing and Debugging Cont.

- There are some portions that can be tested: `ringbuffer_t` is tested with unit tests.
- Other “tests” can be seen on the screen on bootup and have to be verified by a user.
- Interacting with the OS is often the easiest way to find and fix multiprocessing bugs.

Testing and Debugging Cont.

- The OS is filled with bugs; as an example, bidirectional IPC is broken.
- These bugs are the result of OS development being an *very* difficult problem.

Recommendations to the Customer

- Don’t use this OS in production! Use our experience and mistakes to inform what you should do next.
- Constrain the requirements: maybe you don’t need dynamic memory allocation, a keyboard driver, or a complex scheduler.
- Buy an embedded OS off the shelf – we’ve got enough knowledge now to maybe extend that.

Reflections on the MiniOS Project

- Debugging these low-level, multiprocessing, hardware-adjacent problems is absolutely hell.
- It’s important to be skilled with your tools: being more comfortable with GDB could have sped up iteration and debugging.
- It’s one thing to learn OS “theory”, proper implementation is another thing entirely.