

# CMSC412

dun dun dun...

Project 0/1

# Administrivia

---

- <http://www.cs.umd.edu/class/fall2008/cmssc412>
- CSI 1121; Mon, Wed 11:00-11:50
- Me: Aaron Schulman ([schulman@cs.umd.edu](mailto:schulman@cs.umd.edu))

# Why are we here?

---

- To get you started on the project and answer your questions.
- Give you background material.
- Show you how the concepts you learn in lecture apply to GeekOS.
- Come with questions.

# Why are operating systems such a big deal?



# Semester project

---

- Read the source
  - It is heavily commented.
- Post your questions on the forum.
- Come to recitation with questions.
- These projects are challenging, **but fun**

---

# Start Early

That's what they all say

# Setup build environment

---

- Setup instructions are on the web page:
  - QEMU, Compilers, Debuggers
    - Linux, Mac and Linuxlab
    - Cygwin
  - GeekOS build

# GeekOS

## emulation environment

---

GeekOS

QEMU (Hardware Emulator)

Linux/Mac

Real Hardware

# Project 0

---

- Go home, order **DP Dough**, start project
- Project requirements:
  - Resource restrictions on GeekOS processes:
    - # of active processes
    - # of syscalls by a single process

# Project 0

---

- The OS is split into two *user-level* and *kernel-level*.
- The two levels are connected by the *system call* boundary.

# System calls

- Software interrupt
  - The only interrupt callable from user level `idt.c Init_IDT`
  - `SYSCALL_INT: 0x90`
- Operation: `syscall.h; syscall.c; libc/process.c`
  - Put arguments in registers on user side; raise INT.
  - Recover them on kernel side.
  - Call the appropriate `Sys_XXX`.
  - Return result/error code in appropriate register.
- Use `g_CurrentThread` for information about who raised it

# Thread system

---

- Each thread is a `Kernel_Thread` object: `kthread.h`
- Current thread: `g_CurrentThread` global
- User mode threads
  - `Kernel_Thread` objects with a populated `User_Context`
- Transferring from user mode to kernel mode: `syscall`
- Kernel vs user memory
  - Distinct views: one from the user and one from the kernel.
  - Kernel needs to access user memory (but not vice versa!)
  - Use `Copy_From_User/Copy_To_User`

# The system queues

---

- Thread\_Queue structure
- Run queue:
  - Threads which are ready to run, but not currently running.
  - GeekOS has a single run queue for now...
- Wait queues:
  - Threads that are waiting for a specific event or on a specific device; e.g. Keyboard IO, network IO, other threads:
  - `geekos/kthread.c Join()`
  - Spend 2 mins: follow the `Get_Key` syscall to see how the thread eventually gets to the keyboard wait queue

# Interrupts

- Types:
  - Illegal operations: **result in kills**
  - Faults: page faults etc: **not of concern right now**
  - Hardware interrupts.
  - Software interrupts (traps): *syscall int*
- Interrupt handlers
  - **src/geekos/int.c**
  - On completion control returns back to the thread that was interrupted.

# Interrupts

- When you don't want to receive them:
  - When you are modifying global data structures; queues etc.
  - When you want to make some operation atomic.
  - `Disable_Interrupts()` / `Enable_Interrupts()`:
    - `include/geekos/int.h`
    - Use caution
    - `Enable_Interrupts()` when atomic operation finished
    - See places where this has been done: e.g. `src/geekos/user.c Attach_User_Context()` and `src/geekos/kthread.c Reaper()`
  - `Begin_Int_Atomic()` / `End_Int_Atomic()` - `include/geekos/int.h`
    - Oblivious way of saving and restoring interrupt state.