Administrivia

- CSIC 2118, MW 9:00-9:50 / 10:00-10:50
- John Silberholz (josilber@umd.edu)
- Calvin Grunewald (cgrunewa@gmail.com)
- TAs
  - Bo Han (bohan@cs.umd.edu)
  - Vasilieos Lekakas (lex@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

- Parallel to lecture
- Lots of code
  - Heavily commented
  - Read it!
- Use the forum
  - Quickest way to get questions answered
- Come to recitation with questions
- These projects are challenging, but fun
Best advice for success

- Start early
  - Seriously
  - No, for real
Getting Started

- Getting the source
  - `svn co https://svn.cs.umd.edu/repos/geekos/project0`
  - Uname: “sv-geekosro”; passwd: “”
- Setup instructions are on the website:
  - QEMU, Compilers, Debuggers
    - Linux, Mac, and Linuxlab
    - Cygwin
  - GeekOS build
GeekOS Emulation Environment

- GeekOS
- QEMU (Hardware emulator)
- Linux/Mac
- Real Hardware
Project 0

• Project requirements
  – Resource restrictions on GeekOS processes:
    • # of active processes
    • # of syscalls by a single process
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 args in registers on user side; raise INT
  - Recover them on kernel side
  - Call the appropriate `Sys_xxxx`
  - Return result/error code in appropriate register

- **Use** `g_CurrentThread` for information about who raised it
Threads

- Each thread is a Kernel_Thread object: kthread.h
- Current thread: g_CurrentThread (global)
- User mode threads
  - Kernel_Thread with a populated User_Context
- Transfer from user to kernel mode: syscall
- Kernel vs. user memory
  - One from user view and one from 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 one run queue (for now.....)

- **Wait Queues:**
  - Threads that are waiting for a specific event or on a specific device (keyboard/network/other threads)
  - `geekos/kthread.c Join()`
  - Follow Get_Key syscall to see how the thread gets in the keyboard wait queue
Interrupts

- **Types:**
  - Illegal operations (result in kills)
  - Faults (like page faults; not of concern now)
  - Hardware interrupts
  - Software interrupts (traps): syscall int

- **Interrupt handlers**
  - `geekos/int.c`
  - On completion control returns to the thread that was interrupted
Interrupts

- When you don't want them:
  - When modifying global structures, queues, etc.
  - When you want an atomic operation
  - Disable_Interrupts() / Enable_Interrupts()
    - include/geekos/int.h
    - Use caution (interrupt state dependent)
    - Enable_Interrupts() when finished
    - See examples: src/geekos/user.c: Attach_User_Context()
- Begin_Int_Atomic() / End_Int_Atomic
  - Oblivious way of saving and restoring interrupt state