Project 1

Murder, Subversion, and Spying

in GeekOS
• Project 0 due Thursday, 11:59PM
• Homework 1 due **tomorrow** in class
• Submit instructions for Project 0 **later today**
Overview

• Augment GeekOS to include:
  • **Background processes**
  • **An ability to kill processes:**
    • Asynchronously
    • From another process
  • **An ability to view status of active processes**
Questions

• How are processes implemented in GeekOS?

• How do processes use system calls to request kernel services?
Address Space Protection

- Protects against processes accessing:
  - Another processes memory
  - Kernel memory

- Logical addresses used
  - Kernel controls what memory a process can access
  - An interrupt is issued if the process attempts to access memory outside of its logical address
Address Space Protection in GeekOS

- User processes’ address spaces don’t overlap
- Kernel “sees” all address spaces
Address Space Protection in GeekOS

- Allow for the use of *relative* memory references
- Relative to the base of the *current* memory segment
- Linker must know where parts of the program will be w/ regards to the start of the executable image in memory.
Segmentation Principles

Each user program has memory segments for code, data, stack, etc...

Segment Descriptor

Kernel Address = User Address + Base

Gives user processes the **illusion** they have their own world that starts at 0
X86 Segmentation in GeekOS

- Segment Descriptor
- Base address
- Limit address
- Privilege Level
- Descriptors are stored in descriptor tables
- Two types of descriptor tables
Descriptor Tables

- **Local Descriptor Table (LDT)**
  - Stores the segment descriptors for *each* user process.
  - One per process

- **Global Descriptor Table (GDT)**
  - Stores information for *all* of the process
  - For each user process, a descriptor in the memory containing the corresponding LDT
X86 Segmentation in GeekOS

GDT

Global Descriptor Table: holds LDT descriptors for user processes

LDT

Local Descriptor Table: holds segment descriptors for user processes
X86 Segmentation in GeekOS (implementation)
X86 Segmentation

- Intel docs, fig. 3-1 and 3-5
- Yes, you should download a copy
User Processes in GeekOS
Lifetime of an User Process

- Shell spawns user processes using `Spawn_With_Path`
  [(see src/user/shell.c)](src/user/shell.c)
- User processes termination
  - Normally - via `Exit`, called automatically when `main()` finishes
  - Killed - via `Sys_Kill` which you will implement
- Parent processes can wait for their children using `Wait`
System Calls Review

• Program may need to access memory
  • i.e. for I/O, may need to access video memory outside of the processes’ segment

• OS provides a series of System Calls
  • Routines that carry out some operation for the user process that calls it.

• In GeekOS, INT90
System Calls Review

- INT90
- put args in registers on user side
- recover them on kernel side
- call Sys_XXX accordingly
  - Sys_Null, Sys_Exit (src/geekos/sycall.c)
- return result/error code
- Use g_CurrentThread to get info about current thread
Background Processes

- Shell
- `src/user/shell.c`
- Modify code to handle forking process
  - Parse commands and scan for `&`
  - If `&` detected, spawn in background, don’t `Wait()`
  - If `&` not detected, Spawn normally, do `Wait()`
- `Sys_Spawn()`
- `src/geekos/syscall.c`
- need to consider ‘spawn in background argument’
Killing Background Processes

- **Kernel:** `Sys_Kill()`
- `src/geekos/syscall.c`
- Get the PID of the victim process
- Lookup the victim’s `kernel_thread` (see `Lookup_Thread` in `src/geekos/kthread.c`)
- Dequeue thread from all queues, and ‘kill’ it

- **User**
  - Add `src/user/kill.c` for testing
  - Add `kill.c` to `USER_C_SRCS` in `build/Makefile` to create an user program
Printing the process table

- **Kernel:** `Sys_PS()`
- Return information about current processes
- `src/geekos/synccall.c`
- Prepare an struct `Process_Info` array in kernel space
- Walk all threads: `s_allThreadList` in `src/geekos/kthread.c`, fill out the above array
- Copy array into user space: `Copy_To_User()`

**User**

- Add the ‘ps’ user program: `src/user/ps.c`, see req #2
- Hit ‘ps’, ‘man ps’ in Linux to get an idea