Project 1

...because we want to do more than count system calls and spawned processes
Administrivia

• Project 0 due tomorrow, September 17, 11:59 PM
• Homework 3 due September 24
• Office Hour change
  – Project TAs will be holding office hours MW 1:00 PM – 2:00 PM
Overview

• After completing project 1, GeekOS will be able to:
  – Spawn background processes
  – Kill processes
    • Asynchronously
    • From another process
  – Provide status of active processes
    • Similar to the ps unix command
But First... Some 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 space
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 with regards to the start of the executable in memory
Segmentation Principles

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

Segment Descriptor

- size
- base
- privilegeLevel
- ...

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 processes
  – Contains a LDT segment descriptor for each user process
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
• http://www.intel.com/products/processor/manuals
User Processes in GeekOS
Lifetime of an User Process

• Shell spawns user processes using `Spawn_With_Path` (see `src/user/shell.c`)

• User process 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

- Processes may need to access protected system resources
  - E.g., 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
  - Execute in ring 0 – kernel mode (as opposed to ring 3, user mode)
System Calls Review

– In GeekOS, int 0x90
– int 0x90
  • Put arguments in registers on user side
  • Recover the arguments on kernel side
  • Call Sys_xxx accordingly (Sys_Null, Sys_Exit) (src/geekos/syscall.c)
  • Return result or error code
– Use g_currentThread to get information about the current thread
Background Processes

• Changes to the shell (src/user/shell.c)
  – Modify code to handle forking processes
    • Parse commands and scan for &
    – If & detected, spawn in background, don’t \texttt{Wait}()
    – If & not detected, spawn normally, do \texttt{Wait}()
  
• \texttt{Sys\_Spawn}() (src/geekos/syscall.c)
  – Need to consider ‘spawn in background argument’

• Input
  – Background process and any child of a background process cannot receive input from the keyboard
Killing Background Processes

• Kernel: Sys_Kill() (src/geekos/syscall1.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
    • Run queue, join queues, device queues, etc.
  – The currently running thread can kill itself
  – Handle killing zombies, FG processes, BG processes, Wait()ing processes, processes spawned by background processes, etc.
Killing Background Processes

• User
  – Add src/user/kill.c for testing
  – Add kill.c to USER_C_SRCS in build/Makefile.common to create an user program
Printing the Process Table

• Kernel: Sys_PS() (sys/geekos/syscall.c)

  – Return information about current processes
  – Prepare a struct Process_Info array in kernel space
  – Examine all threads: s_allThreadList in src/geekos/kthread.c, and fill out the above array
  – Copy array into user space: Copy_To_User()
Printing the Process Table

• User
  – Add the ‘ps’ user program: `src/user/ps.c`
  – Run ‘ps’, ‘man ps’ in Linux to get an idea of what we are looking for