# Operating Systems

## Week 2 Recitation: The system call

Paul Krzyzanowski

Rutgers University

Spring 2015

# System calls

- System calls are an operating system's API
  - The set of functions that the operating system exposes to processes
- If you want to the OS to do something, you tell it via a system call
- Examples

| Windows | Linux |
|---|---|
| *NtOpenFile* | *open* |
| *NtReadFile* | *read* |
| *NtCreateProcess* | *fork* |
| *NtGetCurrentProcessorNumber* | *getpid* |

See http://j00ru.vexillium.org/ntapi/ for a list of Windows system calls

See http://linux-documentation.com/en/man/man2/ for a list of Linux system calls

# What are system calls used for?

- Anything to do with:
  - Accessing devices
  - Accessing files
  - Requesting memory
  - Setting/changing access permissions
  - Communicating with other processes
  - Stopping/starting processes
  - Setting a timer
- You need a system call to:
  - Open a file
  - Get data from the network
  - Kill a process
- You ***do not*** need a system call to:
  - Replace data in a string
  - Create an object (instance of a class)
  - Call a function

# System calls are made via traps

- System calls request operating system services
- Operating system code executes with the processor running in *kernel* (also known as *supervisor* or *privileged*) mode
  - Privileged mode gives the CPU the rights to:
    - Execute special instructions
      (change interrupt masks, set hardware timers, halt the processor)
    - Access specific registers (e.g., private stack pointer)
    - Change the memory map
    - Access regions of memory that have been restricted for kernel access only
    - Access the processor's I/O ports (if the architecture has them)
- A trap takes has one parameter: index into an Interrupt Vector Table
  - The table is in memory that only the kernel can access
  - All addresses in the table go to well-defined entry points in the OS

# Making system calls programmer-friendly

- System calls are made to look like function calls
- As a programmer, you do not want to
  - copy parameters into some special place
  - know the system call number
  - invoke a software interrupt
  - figure out how to copy any return data back
- System call library
  - A user-level library that is linked with your program
  - Provides a functional interface to system calls
  - Handles the work of passing parameters and getting results

# System calls

## Entry
## Trap to system call handler

- Save state
- Verify parameters are in a valid address
- Copy them to kernel address space
- Call the function that implements the system call
  - If the function cannot be satisfied immediately then
    - Put process on a ***blocked*** list
    - ***Context switch*** to let another *ready* process run

## Return from system call or interrupt

- Check for signals to the process
  - Call the appropriate handler if signal is not ignored
- Check if another process should run
  - Context switch to let the other process run
  - Put our process on a ***ready*** list
- Calculate time spent in the call for profiling/accounting
- Restore user process state
- Return from interrupt

# The End