What is an Operating System?
The Big Fuzzy Picture

Michael E. Locasto

Department of Computer Science
UofC
CPSC 457

September 19, 2014
Last week, we spoke about some of the primary responsibilities and concepts involved in operating systems. Last session, you learned a bit about OS history (to give us some context). Today, we will consider the “big picture” of OS structure and try to define some key terms.
Recap your knowledge of x86 architecture:
“A Tiny Guide to Programming in 32-bit x86 Assembly Language”

A great exposition of how to speak machine language directly to the OS:
“A Whirlwind Tour on Creating Really Teensy ELF Executables for Linux”

MOS 1.3: Computer Hardware Review
MOS 1.6: System Calls
MOS 1.7: Operating System Structure
Recap / Connections

Why?

(a) give you an overview of some commands from the command line
(b) give you an idea of the rich variety of commands (from ps to sftp)
(c) demonstrate the use of the Unix manual (man(1) command)
(d) use it to drive a discussion of processes (ps, pstree, top)
(e) observe the transformation of code to binary (+ reinterpretation)
(f) observe the coarse–level detail of an ELF file (i.e., sections)
(g) use strace(1) to run a program
(h) give concrete examples of the “stuff” that application programs are
(top of diagram)

Observe:

The operating system does not execute source code or programs; rather, it manages processes. The ELF is part of a contract between the compiler and the Operating System. This observation leads us to ask: so how do programs become processes? (We have seen this already)
Obtain an intuition for *what* an operating system kernel is and what its roles and defining characteristics are by placing it in the context of managing the execution of processes on “top” of some hardware.
What is the difference between an operating system and a shell?

What is the difference between an operating system and an OS distribution?

What is the difference between an operating system and an OS kernel?
What main problem do operating systems solve?

→ Manage time and space multiplexed resources

→ Mediate hardware access for application programs

→ Load programs and manage execution
(a) a pile of source code (LXR)
(b) a program running directly on the hardware
(c) a resource manager (multiplex resources, hardware)
(d) an execution target (abstraction layer, API)
(e) privilege? supervisor? (LDK page 4)
(f) userland / kernel split
Sketching time.

Claim: in order to understand what an OS is, we need to understand its context and the environment it operates in (this is driven by the 2-3 problems it solves)
Monolithic kernel (e.g., Linux): a single address space
Layering (e.g., THE, MULTICS)
Microkernel: small supervisor, most functionality placed into user–level processes
Client-server: a distinction without a difference
virtual machines: VMM, hypervisors
exokernel: VMM-like construct that partitions system resources (i.e., a namespace manager)
Next session, we will examine the intimate relationship between the underlying hardware platform and what the OS does and how it does it.

How does a kernel enforce these divisions?

What is the necessary architectural support for basic operating system functionality?

How do processes ask the operating system to do stuff?