CS 5204
Operating Systems
Lecture 5

Godmar Back

Announcements

• (Due to a lack of announced late policy)
  Deadline for Project extended from Oct 2 to Oct 3, 11:59pm.
• No extensions (zero-score) beyond that point.
• Awaiting your project/survey paper revisions – note: waiting for confirmation is not a reason to delay working on the project.

Plan for Today

• Brief overview of kernel systems research in 1980-today
  – Create context for Exokernel and HiStar papers

A (somewhat selective) History of OS/Kernels

• 70s: Unix (AT&T and BSD)
  – Plus many architecture-specific OSes
• 80s: Mach (followed Accent)
  – Pioneered microkernel idea
• 90s:
  – Tanenbaum vs Torvalds debate
  – Engineering of Windows NT & contemporary Unixes
  – In Parallel: decade of “Extensible OS” Research
• 00s:
  – Virtual machine monitors
  – Virtual machines

The Red Line

• Separates user mode & kernel mode
  Chanton 1994

• Q: what should go above the red line, and what should stay below

Microkernels
Microkernels (II)

- Idea of “OS personality”
  - Provided on top of μ-Kernel using servers
- Noteworthy projects:
  - L4 microkernel [L4]
    - Focus on making message-passing fast
  - Fluke
- Industry:
  - NT, OSX – started with μ-Kernel approach, moved away – mostly for performance reasons

The Torvalds/Tanenbaum Debate

- Should you develop a new OS as a monolithic kernel in 1992?
- Questions [Torvalds/Tanenbaum]
  - Microkernel vs. monolithic
    - How important is OS structure?
  - Portability
    - What defines portability?
  - Free Software
    - How important is the development process?

Extensible OS Research in 90s

- Idea: Extensibility, extensibility, extensibility
- Number of projects explored different approaches
  - Exokernel (MIT)
  - SPIN (UW)
  - VINO (Harvard)
  - Fluke/OSKit (Utah)
  - Scout (Arizona)
  - L4 (Liedtke/IBM)
  - Nemesis (U Cambridge)

Exokernel: exterminate all OS abstractions

- The defining tragedy of the operating systems community has been the definition of an operating system as software that both multiplexes and abstracts physical resources. The view that the OS should abstract the hardware is based on the assumption that it is possible both to define abstractions that are appropriate for all areas and to implement them to perform efficiently in all situations. We believe that the fallacy of this quixotic goal is self-evident, and that the OS problems of the last two decades can be traced back to it. The solution we propose is simple: complete elimination of OS abstractions by lowering the operating system interface to the hardware level.

SPIN

- Protection is a software issue
  - First use of a type-safe language to build a kernel (Modula-3)
  - Idea: applications can download code into the kernel safely
SPIN (cont’d)

• How to manage extensions
  – What if they run for too long?
  – How to unload extensions?
  – How to manage their resources (memory, etc.)?
• How to foresee what extensions might want to extend?
  – Size of exposed interface
• How to arbitrate among extensions?
• Type-safe language issues
  – garbage collection
  – language domain crossings (M3 to C and vice versa)

Language-based OS

• If protection is a software issue, why not do away with hardware protection altogether?
• Inferno OS system by Bell Labs
  – Limbo: typesafe language
  – DIS: virtual machine
• Early aspirations of Java
  – A Java Operating System written in Java

Java Operating Systems

• Two views:
  – A fully-fledged operating system written in Java that runs Java applications.
    • Sun’s first JavaOS, Squawk JVM
  – A JVM that provides an OS-like environment for Java applications. The so enriched JVM can run on top of any OS.
    • J-Kernel, MVM, KaffeOS

VINO

• Idea: extensions (“grafts”) are “safe overrides” for functionality at (C-) function level
• Use a transaction mechanism to recover from misbehaved extensions
  – Full transaction mechanism inside the kernel
  – Based on monitoring load/stores from C++ programs

Fluke

• Microkernels meet recursive virtual machines
• Idea: processes could create whatever abstractions they wished for their descendants

OSKit

• Idea: build OS you want from components
IO-Lite/Flash-Lite

- Opposing View: Druschel et al HotOS’97
  - Extensible Kernels are Leading OS Research Astray
- Extensibility mostly busy solving problems it created
- Instead, develop, test, decide, and integrate into commodity OS
- Case in point: IO-Lite & Flash server
  - Zero copy I/O system & event-based webserver
  - Applications control I/O buffer inside kernel

Discussion

- What did we learn from extensible OS research?
  - Is extensibility as important as claimed?
  - If not, what is?
- Is developing entire new OS still feasible today in an academic environment?
  - Is doing so necessary for exciting systems research?