Structural variants
Monolithic, Mikrokernel and Exokernel
5.12.2005
OS kernel: a definition

- Quick and dirty definition:
  - mandatory part of the OS
  - common to all applications
- Responsible for
  - security
  - managability
  - enables a uniform execution environment
OS-kernel: a definition

• An operating system kernel provides
  • virtualization of hardware resources
  • isolation of individual tasks and applications
  • common platform-independent abstractions
  • secure access to hardware devices
  • access control and policy enforcement
Virtualization

- Present logical resources not physical entities
  - manage isolated access by multiple users
  - lock physical entities at the kernel level
  - expose “virtual” resources based on the capabilities of the physical devices
- hide the fact that physical resources are scarce and may only be controlled by a single owner
- multiplex access to hardware resources
- enforce access conditions and permissions
Virtualization: CPU

- Virtualization of CPU resources
  - Execution is continuous
    - starting of the current program counter
    - controlled by the program’s instructions
    - restricted to the current address space
  - However
    - different applications will execute in different spaces
    - a context switch requires heightened privileges
      - potential security problem!
Virtualization: CPU

• Provide multiple “virtual” CPUs
  • use time-division to provide individual timeslices
  • periodically take a scheduling decision
  • execute the context switch from a trusted environment to avoid privilege escalation

• OS kernel implementation
  • provides exception (and IRQ) handlers
  • runs in supervisor mode
Virtualization: CPU

- Implementation details
  - a task list with
    - memory context
    - execution context: area for “register spillage”
  - an exception handler for a periodic (timer) interrupt
  - a scheduling algorithm
- Every process seems a dedicated “virtual” CPU
Virtualization: I/O

- Ethernet
  - Background
    - Dedicated, serial transmission medium
    - Transmits frames from memory buffers to the transmission medium
  - Problem
    - Central instance needed
      - to classify incoming packets
      - to distribute packets to individual applications
Abstraction

- Abstraction
- Definition
  - reduce and manage complexity
  - factor out details to focus on few concepts
- Example: block-devices in UNIX
  - Provision of a common interface
  - Hides the details of various devices and media
    e.g.: Flash media vs. a hard-disk
OS kernel variants

- Three major strategies to implement an operating system kernel
  - Monolithic kernels
  - Microkernel systems
    - Pure microkernel designs
    - Hybrid microkernel designs
  - Exokernel approach
- All have different advantages and drawbacks
- There is no single "silver bullet"
Monolithic kernels

- The OS kernel is a single (monolithic) structure
  - executing entirely in supervisor mode
    - process management
    - memory management
    - drivers for hardware
  - provides a set of system calls to interface with operating system services
  - may use modules to optimize resource utilization
Monolithic kernels

• Typical examples
  • Traditional UNIX kernels (System V and BSD flavours)
  • FreeBSD
  • Solaris
• Linux (see www.kernel.org for source code)
Monolithic kernels

• Advantages
  • Efficiently using function calls between kernel modules
  • Requires a low number of context switches

• Drawbacks
  • Limitations in robustness
  • No fault or privilege isolation

• History
  • In widespread use before MMUs became common
Monolithic kernels

• Summary
  • easy to design and develop
  • hard to evolve without risking growing pains
Microkernel designs

• Design goals
  • use a very simple abstraction over hardware
    • thread management
    • address spaces
    • interprocess communication
  • reduce the functionality in supervisor mode
    • move specific drivers to user-space
    • isolate individual network services in a separate context
Microkernel designs

- Examples
  - AIX
  - AmigaOS
  - Amoeba
  - Chorus
  - Mach
  - Minix (remember Tannenbaum vs. Torvalds?)
  - QNX
  - Symbian OS
Microkernel designs

• Overall architecture
  • OS consists of a kernel and “servers”
  • “Servers” provide high-level functionality
    • external pager
    • device drivers
  • communication is performed using IPC/RPC
    • potential performance problem
    • context switches are expensive
Using an external pager

- A page-fault is handled by an external pager
- The RPC to the pager appears to originate from the faulting process
Fast IPC

• Issues in IPC
  • context switch overheads
  • additional processing for policy enforcement

• The L4 approach
  • Reduce additional processing
    • Policy enforcement is not part of the kernel
    • Visibility provides security at the kernel level
  • Reduce context switch overheads
    • No intermediate switch to kernel mode
Benefit of microkernels

• Essential to trusted computing concepts
  • Provides a small, controlled operating system core
  • Controls the hardware and resources
    • Holds initial ownership of all resources
    • Delegates individual resources to trusted processes
  • Isolates privileges and resources
  • Can provide service level guarantees
L4: a microkernel

• Small number of mechanisms
  • Address spaces
    • Creating address spaces
    • Mapping pages into address spaces
    • Revoking mappings
  • Scheduling
    • Yielding the current time-slice to a specific thread
  • Interprocess communication
    • Performing message transfer to specific threads
L4: a microkernel

- Small number of mechanisms (continued)
  - IPC protocols for specific functions
    - external pager (pagefault handling)
    - interrupt notification
    - preemption notification
    - exception notification
L4: Security

- Selective address space manipulation (system partitioning) provides as a framework for security
- Memory permissions reflect process permissions
Microkernel designs

• Summary
  • Hard to design the component interfaces
  • Provides long-term security and reliability benefits
Hybrid kernels

• The other approach to optimizing IPC
  • Optimize performance, reduce latency
  • Don’t focus on IPC

• Moves non-essential functionality into the kernel
  • typically the pager
  • frequently other OS services

• Examples
  • MacOS X: XNU
  • Windows NT and later
Exokernels

• A special case of microkernels
  • Reduction of mechanisms in supervisor space
  • Moves all abstractions to user-space
    • Limited to virtualization
    • Exposes the hardware to user-space programs
• Currently available as research prototypes only
Exokernel design

- Kernel mechanisms
  - similar to those of L4 in many respects
    - control of processor access
    - control of address spaces
  - different in controlling all functionality from the kernel
    - primitives for different devices and device classes
      e.g., block devices, network devices, ...
  - not designed for user-space device drivers
Recommended Reading

- Engler D.R., Kaashock M.F., O’Toole J. Jr
  Exokernel: An Operating System Architecture for Application-Level Resource Management
- Engler D.R., Kaashock M.F.
  Exterminate all Operating System Abstractions
- Engler D.R., Kaashock M.F., O’Toole J. Jr
  The Operating System as a Secure Programmable Machine
Recommended Reading

- Bershad B. N.
  The Increasing Irrelevance of IPC Performance for Microkernel-Based Operating Systems
- Rashid R., et al.
  Mach: A System Software Kernel
Recommended Reading

- Liedtke J.
  On μ-Kernel Construction
- Liedtke J.
  Improving IPC by Kernel Design
  The Performance of μ-Kernel-based systems
Recommended Reading

• Härtig H., Hohmuth M., Wolter J.
  Taming Linux

• LeVasseur J., Uhlig V., Stoess J., Götz S.
  Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines