Outline

• Extensible Operating Systems
  – rationale
  – microkernel approaches (L3/L4)
    • what should be in a microkernel?
    • overhead of microkernel operations
  – exokernel approaches (Aegis)
    • philosophy
    • primitive operations

• [Lied95] Liedtke, J., On Micro-Kernel Construction, 15th SOSP

Operating System Structure

• UNIX-like monolithic OSes

• Disadvantages
  – inflexibility (bloated kernels are a consequence)
  – lack of modularity
  • discourages changes to implementations of existing abstractions

Alternative: Extensible Operating Systems

• Collectively referred to as microkernel-based OSes
  – a thin kernel layer provides minimal functionality
  – higher-level abstractions built on top

• Benefits
  – modularity
  – servers are just like any other user process: server malfunction is isolated
  – system is more flexible and tailorable
  • different strategies and APIs can coexist in the same system

• Different from OS flavors
Main Challenge for µKernel OSes: Inefficiency

What Should a µKernel Contain?

Two points on the spectrum
- L3/L4
  - start from a conventional OS and trim down stuff
    - what is the minimal set of abstractions that must be supported?
    - can cross-protection domain interactions be efficiently implemented?
  - Aegis
    - who needs abstractions anyway?
    - µkernel exposes the underlying hardware
      - only manages protection and fair allocation of resources
      - all resource management decisions provided by user-level libraries

L3/L4
- Three factors affect performance of µkernel-based OSes
  - µkernel approach
    - implies need for additional protection-domain crossings
  - concepts implemented by a particular µkernel
    - what abstractions does it support?
    - choice of abstractions influences
      - functionality (e.g., protection must be supported in the µkernel)
  - implementation of the µkernel
    - primitive operations such as context switches, IPC, etc.
- L3/L4 was an exercise in analyzing the contribution of these factors
  - a mature project at GMD (German National Research Center for Information Technology)
    - active since 1989

L3/L4: What Should a µkernel Support?
- Objective: System must support interactive and/or not completely trustworthy applications
  - protection
  - principle of independence
    - a subsystem S can give guarantees independent of S'
  - principle of integrity
    - there must be a way for S1 to address S2 and establish a communication channel that can neither be corrupted nor eavesdropped by S'
- L3/L4's solution
  - address spaces
  - threads and IPC
  - unique identifiers
L3/L4 Concepts

- Address spaces
  - an address space is a mapping which either associates each virtual page to a physical page frame, or marks it inaccessible
  - L3/L4 supports recursive construction of address spaces
    - kernel provides $\sigma_0$ (representation for all physical memory)
    - three operations
      - grant: granted page is removed from granter’s address space and included into the grantee’s address space
      - map: owner of an address space can map any of its pages into another address space
        » pages can be accessed in both address spaces
      - flush: owner of an address space can revoke any mappings of its pages
        » flush affects all pages received directly or indirectly
- I/O devices are supported by associating an address space
  - works for both memory-mapped I/O and I/O ports
- Advantage: Memory managers and pagers can be completely implemented at the user-level

L3/L4 Address Spaces

- Use of map/grant/flush

L3/L4 Concepts (contd.)

- Threads and IPC
  - a thread is an activity executing inside an address space
    - set of registers, stack pointer, state information, current address space
  - $\mu$kernel supports threads
    - to ensure protection (by controlling changes of address spaces)
    - to enable preemption (implies support for context switching)
  - IPC is also supported by the $\mu$kernel
    - messages between threads
    - implicitly enforces an agreement
      - sender decides to send information (determines its contents)
      - receiver determines whether it is willing to receive messages and is free to interpret the received message
    - thus, IPC forms basis for the ‘principle of independence’

- Hardware interrupts are just IPC messages
  - user process is scheduled immediately

L3/L4 Concepts (contd.)

- Unique identifiers
  - $\mu$kernel has to generate IDs for threads, communication channels, etc.

- A handful of concepts yield tremendous flexibility
  (see paper for details)
  - memory manager, pager
  - multimedia resource allocation
  - device drivers
  - everything except processor architecture, first-level TLB and caches

- L4/Linux
  - SOSP’97
  - a complete Linux server on top of L4
**µkernel Costs**

- Measurements on a 50 MHz i486

- Switching overhead
  - kernel-user switches: mode change (~107 cycles) + overhead (~15 cycles)
  - better hardware support can reduce this to ~20 cycles
  - address-space switches: very architecture dependent
    - tagged TLB architectures do not need a TLB flush
    - otherwise, require a TLB flush
    - performance tricks on the Pentium, PowerPC
    - can utilize segment registers to multiplex portions of the user address space
  - overhead: 9 cycles per TLB miss, 15 cycles for a segment switch
  - thread switches and IPC
    - fast/restricted implementations such as LRPC
    - system call, argument copy, stack and address space switch
    - L3 can do an IPC in ~250 cycles (10µs)
    - 100 cycles is trap overhead

- Memory effects
  - previous studies have shown that microkernels result in more system Icache and Dcache misses
  - detailed analysis in paper
    - user misses about the same
      - system structure does not affect the user-level code
    - system misses different primarily due to capacity misses
      - actually, fewer conflict misses in microkernels
      - higher capacity misses is a sign of bad design: no need for it to be that high

- Non-portability
  - to get full performance from µkernels need to reimplement it for each architecture
    - paper contains a good discussion of differences between i486 and Pentium, as well as between the MIPS R2000 and i486
    - so, Windows NT approach of a HAL is not good for performance!

**An Alternate Approach: Exokernels**

- Philosophy: Separation of protection from management
  - a µkernel should provide the lowest-level interface
    - fair allocation of the hardware resources
      - e.g., processors, TLB, caches, memory, framebuffers, disk
    - protection for this allocation
      - expose allocation, expose names, visible revocation
    - all management policies are contained in user-level library OSes

- Benefits: Efficiency, flexibility, and customizability

- Differs from
  - virtual machines:
    - does not provide each application its own view of the hardware
    - just exposes existing hardware: no overhead of supporting the VM
  - L3/L4-like µkernels
    - does not provide any high-level abstractions

**Exokernel Mechanisms**

Three mechanisms to export hardware resources

- Secure bindings
  - applications can securely bind to machine resources and handle events
    - resources: memory, CPU, disk memory, TLB, addressing context identifiers
    - events: interrupts, exceptions, cross-domain calls
    - protection mechanisms that decouple authorization from actual use
      - authorization check only at bind time
      - use requires a capability that can be verified quickly
        - can use hardware (e.g., a TLB entry) or software (packet filters)
        - downloading code can also improve performance (Application Specific Handlers)
          - more about this in the next lecture
    - kernel can protect resources without understanding them
Exokernel Mechanisms (contd.)

- Visible resource revocation
  - e.g., take back a frame, but leave it up to the application to decide which page gets swapped out
  - use of physical names
    - can be exploited by the application to factor in hardware features

- Abort protocol
  - resource revocation from unresponsive applications
  - application must provide a list of resources which can be used to back up repossessed resources
    - e.g., names and capabilities of disk blocks to back up physical memory pages

Performance of Exokernel Primitives

- Very aggressive implementation technology
  - PCT (protected control transfer)
    - ~20 cycles on MIPS R2000
    - restricted IPC for ~50 cycles
    - dynamic compilation of ASHs

- Primitives are really fast
- Flexibility can be exploited for performance
  - custom file systems (for Web servers and databases)
  - custom page management (for distributed shared memory systems)

- Big unknown
  - performance and programmability of large-scale applications
    - will it be expensive to support the needed abstractions?