Alternative Architectures

COMS W4118

Prof. Kaustubh R. Joshi

krj@cs.columbia.edu

http://www.cs.columbia.edu/~krj/os

References: Operating Systems Concepts (9e), Linux Kernel Development, previous W4118s

Outline

- Singularity OS
  - – Motivation
  - – Software Isolated Processes
  - – Contract based IPC
  - – Kernel Architecture
  - – Benefits
- Summary

Singularity OS

- Microsoft Research OS
  - Developed between 2003-2008
  - Shared source code available (http://research.microsoft.com/en-us/projects/singularity/)
  - Influence on MSFT OSes unknown

- Why is it interesting?
  - Radically different approach to memory isolation
  - Use programming language/compiler techniques rather than paging/segmentation hardware

Motivation

- Revisit basic OS design decisions that have been untouched since the 1970s (UNIX)
  - Memory paging based protection model
  - IPC mechanisms
- Incorporate work on programming languages, compilers, and code verification into core OS architecture
  - Assume higher level tools than assembly language
- Improved robustness/reliability than existing OS designs
  - Security vulnerabilities
  - Failures caused by dynamic code (e.g., extensions, etc.)
  - Unexpected interactions between applications
- Good enough performance

Basic Architectural Ideas

- Software Isolated Processes (SIP)
  – Provide memory isolation purely in software
- Contract-based Channels
  – Allow IPC only through statically verifiable protocols
  – Strict memory ownership (one page one process)
- Manifest-Based Programs
  – Programs declare resource requirements upfront
  – No dynamic code injection/extensions

Paging and Software Isolation

- How does paging work?
  - Don't allow direct memory access
  - Access through a pointer (virtual address)
  - OS controls what pointer points to
  - Maintains mappings such that process A pointers never point to process B memory
- Software isolation idea
  - Enforce pointer control through programming language
  - Don't let programmer change pointer indiscriminately
  - E.g., Java
  - The compiler is the OS?

Memory Safety

- In an unsafe language like C
  - Programmer gets direct control of pointers
  - Can access arbitrary memory (int to ptr cast)
    - `char *ptr = (char *)0x88888888`
  - Can increment/decrement existing pointer
    - `char *dangerous_ptr = ptr + 100000;`
- In language with type/memory safety
  - No "pointer" data type – only references to objects
  - Can't arbitrarily change reference
  - Can't directly cast address to a reference
    - E.g., MyClass c = (MyClass)0x88888888 is not allowed
  - Runtime bounds check ensure array safety

Software Isolated Processes (SIP)

- OS/runtime controls initial pointer assignment
  - Processes are allocated their own memory
  - SIP provided only pointers to its own
  - Safety semantics ensure subsequent isolation
- No need for paging/hardware isolation
  - Kernel/processes in same address space and priv level!
  - All memory visible to all instructions (fast IPC)
  - Every syscall is simply a function call
  - No page table change on context switch
  - Very fast (paper shows significantly improved performance compared to paging)

Compile Time Verification

- Compiler creates bytecode (MSIL or Microsoft Intermediate Language)
- Installer "verifies" bytecode and compiles to native code (e.g., x86)
- Verification ensures
  - SIP doesn't create or modify pointers
  - Don't change type of pointer to circumvent bounds check etc.
  - Don't use uninitialized pointer variables
  - Don't use pointers after SIP relinquishes ownership

Limitations

- But…reality intrudes
  - Only type/memory safe PLs supported
  - No C/C++ code, no assembly snippets
  - What to do about legacy code?
    - Still need some hardware protection
  - Relying on compiler and verifier to be correct
    - Millions of lines of complex code (GCC: 7.3 million LOC)
    - Single bug can destroy safety
    - Need fallback, i.e., hardware protection

Other Paging Features?

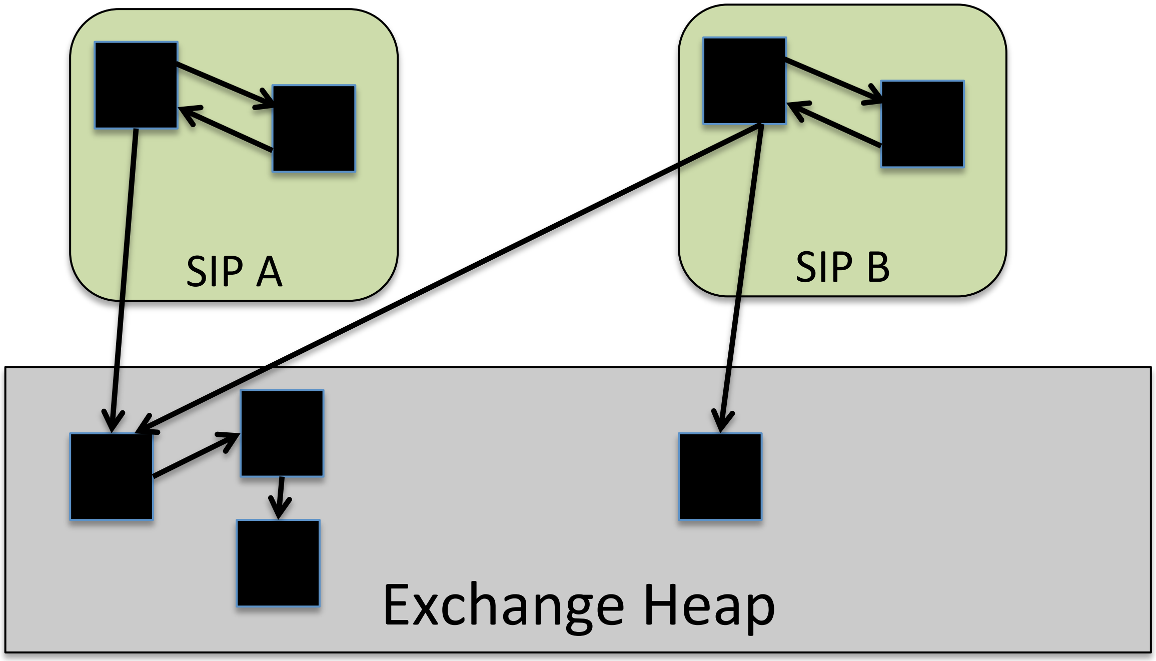
- Illusion of contiguous memory
- Uniform address space
- Freedom from external fragmentation
- Efficient sharing of memory
- Swapping/paging to disk

- What have we gained? Robustness or performance?

Contract Based Channels

- SIPs can communicate only via Contract Based IPC channels
  - Need to be efficient (shared memory)
- Strict control over IPC contents
  - Otherwise SIP may pass any pointer in IPC message
  - Applications must declare protocol before hand
  - Message format, message flow (like we did informally for hw5)
- Ensure memory isolation
  - One SIP can never affect another SIP's memory
  - Makes garbage collection self contained within SIP
- Static verifier checks compliance
  - Does the SIP conform to protocol?

Exchange Heap



- Used for implementing IPC through contract channels
- Enforce single SIP ownership of all pages
  - Verify that SIP doesn't access pointer after sending to another SIP
- Easier garbage collection (no dependency between SIPs)

Manifest Based Programs

- Each program declares manifest up-front
  - Code resources, executable segments
  - Channels, channel contracts, SIP dependencies
  - Hardware resources needed (e.g., ports)
- Disallow dynamic code
  - No loadable modules, dynamic libraries, self-modifying code
  - May have install time extensions
  - Principle: all code must go through same verification process as main program
  - Principle: all safety properties of program must be verified together when it loads

Conclusions

- Does singularity show that…
  - We should get rid of paging/hardware enforcement?
    - No
  - Software isolation provides performance benefits?
    - Yes
  - better robustness is possible than a with a well isolated hardware protected kernel (e.g., microkernels)?
    - Unknown
- Perhaps its utility lies in…
  - Better protecting modules that must exist in a single address space anyway
  - E.g., browser extensions, loadable modules, JVMs etc.
  - Use more explicit communications channels

Course Summary

- OS Architecture
  - Kernels, how kernels are structured
- OS Abstractions
  - Processes, threads, address spaces, files, directories
  - Synchronization
- OS Implementation
  - Interrupts, scheduling, memory management, storage management, filesystems, I/O
  - Both mechanisms and policy
- How a modern OS really works
  - The Linux kernel as modified for Android
  - Saw how theoretical concepts map to reality
  - How to navigate a large codebase
- A flavor of OS research and new designs

Isolation vs. Access Control

- Spent a lot of time on isolation mechanisms
  - How to isolate one application from another
  - CPU (preemptive multitasking),
  - Memory (virtual memory)
  - Disk (filesystems)
  - Network (IPC)
- But how to decide how to use isolation (i.e., policy)?
  - Can a process access a file?
  - Can two processes communicate via IPC?
  - Can a process access an abstraction?
  - Can a process access a resource?
  - The domain of access control policies
  - What are the security implications of various kinds of access control?
  - Ignored in this class – CS4187

Hope it was good for you…

- Learnt a lot
  - You: about operating systems, kernel hacking
  - Me: grading, isolating group conflicts, how much time it takes to teach an OS class ☺
- If you feel excited about systems/OS and did well with the programming assignments
  - Come talk to me about opportunities
  - Consider research/graduate school
  - Talk to the other systems faculty members
- Good luck with the finals!