CS134a: Modern OS (Week 8)

- Two rounds of research
- 1980's: modularity and micro-kernels
  - *Mach (CMU; Rashid, ...)*
  - *Amoeba (Vrije; Tannenbaum, ...)*
  - *X-Kernel (U. Arizona; Peterson, ...)*
  - *V (Stanford; Cheriton, ...)*
- 1990's: minimalism
  - *ExoKernel (MIT; Kaashok, ...)*
  - ***Vino** (Harvard; Seltzer, ...)*
  - *SPIN (U. Washington; Bershad, ...)*

Outline

- Why Extensible Operating Systems.
- The Challenges of Extensibility.
- The VINO Extensible Operating System
- Self Monitoring and Adaptation in VINO.
- Status and Conclusions.

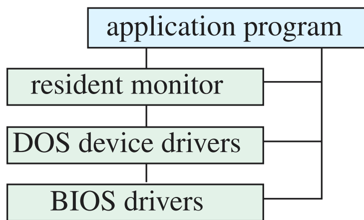
Why Extensible Operating Systems?

- Most needs of applications are similar.
- Not all needs of applications are identical.
- OS provides least-common-denominator services and policies..
- Resource-intensive applications (e.g., databases) avoid the OS.

Extensibility is Common

- Database servers.
- Web browsers.
- Active networks.
- Extensible systems reinvent the "operating system" problem.
  - *Protection.*
  - *Resource management.*
  - *Definition of an abstract interface.*

Example: DOS



DOS

- Minimal functionality
- Extremely flexible
- Add functionality through system calls and device driver hooks
  - *Add buffer cache*
  - *Add filesystems*
- Single address space
- System call redirection lengthens execution path

Mechanisms

- Recompile the kernel
- Linux/Solaris modules
- Exokernel DPF

Issues I

Design Issue	Options		
Mutability	**parameterized** both the code base and conguration are determined at compile time	**recongurable** the code base is determined at compile time, but behavior can be modied at run time	**extensible** both the code base and behavior can be modied at run time.
Location	**user** extensions are implemented at user-level	**kernel** extensions are implemented at kernel-level	
Trust/Failure	**good will** extensions can read/write any portion of the kernel	**software** safe languages or SFI protect kernel from extensions	**hardware** address spaces protect kernel from extensions

Issues II

Design Issue	Options		
Lifetime	**application** extensions persist as long as application runs	**resource** extensions persists as long as the resources to which they are attached	**kernel** extensions persist until kernel reboot when reinitialization takes place
Granularity	**module** entire subsystems are replaced	**limited procedural** a subset of the kernel functions can be replaced	**procedural** any function can be replaced
Arbitration	**prohibition** request fails if there is a conict (e.g. there is a central authority that arbitrates requests)	**split resolution** conicts are avoided by leaving global decisions to the kernel and local decisions to application	**multiplexing** resources are time multiplexed

Mutability

		Code Base	
		xed	**extensible**
Conguration	**static**	Code and behavior xed at compile time. System: Scout Name: **parameterizable**	**Not feasible: this choice implies an extensible code base with static behavior.**
	dynamic	Code is xed, but the behavior can be modied at run time. System: Solaris (scheduling) Name: **recongurable**	Both code and behavior can be modied at runtime. System: SPIN, VINO Name: **extensible**

VINO: avoid Re-solving OS Problem

- Let applications manage own resources.
- Let operating system arbitrate between applications.
- Use operating system authentication.
- Design a single abstract interface.

The OS Perspective

- Keep kernel functionality simple.
- Allow applications to specify complex behavior.
- Prevent applications from damaging each other.
- Provide mechanisms to arbitrate resources among different applications.

The Challenges of Extensibility

- Protection Mechanism.
- Resource Allocation.
- Identifying what to Extend.

Protection Mechanisms

- Two Issues:
  - *Limiting access.*
  - *Controlling behavior.*
- Hardware/VM.
  - *Exo-kernel.*
  - *Microkernel.*
- Language & Atomic Interfaces.
  - *Spin (Modula-3).*
- Software Fault Isolation & Transactions.
  - *VINO.*

Controlling Behavior

- Can application retain state across kernel calls?
- SPIN
  - *Protection provided by safe languages and guards.*
  - *Extensions do not hold locks between calls to the main SPIN kernel.*
  - *Extensions cannot call freely into base SPIN system.*
- VINO
  - *Protection provided by SFI and transactions.*
  - *Extensions (grafts) hold locks/resources across calls to the main VINO kernel.*
  - *Grafts have fairly extensive call interface to main kernel.*

Taxonomy

System	Mutability	Location	Trust/Failure	Lifetime	Granularity
Apertos	extensible	kernel	software	application	procedural
Exokernel	extensible	user	hardware	permanent	procedural
	extensible	kernel	software	application	limited procedural
Mach	extensible	user	hardware	kernel	module
MS-DOS	extensible	kernel	good will	kernel	module
Choices	extensible	kernel	software	kernel	module
Scout	parameterizable	kernel	good will	permanent	procedural
SPIN	extensible	kernel	software	kernel	limited procedural
Solaris	parameterizable	kernel	good will	permanent	module
	recongurable	kernel	good will	application	module
	extensible	kernel	good will	kernel	module
VINO	extensible	kernel	software	resource	limited procedural

Resource Allocation

- Extensions can hog resources.
    - *The while (1) {} extension.*
    - *The while (1) { malloc/new } extension.*
    - *Lock a resource and never unlock it.*
  - *Explicitly.*
  - *Implicitly.*
    - *Call host operations that obtain resources or locks.*
- Kernel still needs to deal with resource allocation.
- Kernel defines policies for setting resource limits.

Identifying What to Extend

- Some applications are well understood.
- Many are not.
- Need a systematic method to identify what to extend and how.
- Not much current research on this topic.
- Last part of this talk presents the VINO approach to solving this problem.

The VINO Extensible Operating System

- Kernel is a toolkit of building blocks.
- Applications can replace policies.
- Applications can add new services.
- Kernel mediates between conflicting policies.

Key Design Points

- Grafts are untrusted, written in C or C++.
- Software fault isolation adds runtime checks.
- Time-outs prevent resource starvation.
- Transaction mechanism protects kernel from misbehaved graft.

Software Fault Isolation

- Existing (clever) SFI tools add runtime overhead of 10%.
- Our (simple) SFI tool adds overhead of 50%-100%.
- Techniques to optimize SFI are similar to standard compiler optimizations.

Transaction Overhead

- Lightweight in-memory transactions.
- Protects shared state by locking.
- Use transaction abort when resource consumption exceeds system limits.
- Unoptimized costs (120MHz Pentium).

Transaction Begin	36 ms
Transaction End	30 ms
Lock/Unlock	33 ms

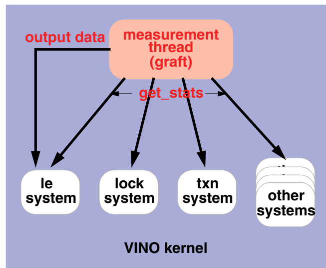
Self Monitoring and Adaptation in VINO

- Now what?
- Hard work determining what to "fix."
- Need to assist application writers in identifying (and fixing) bottlenecks.
- Two prong approach:
  - *Identify problems.*
  - *Suggest solutions.*

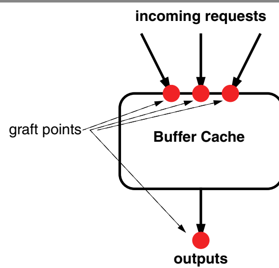
Self-Monitoring in VINO

- Measurement thread periodically collects module statistics.
- Generate detailed profiling information.
- Capture module inputs (traces) and outputs (logs).
- In-situ simulation evaluates competing algorithms and policies.

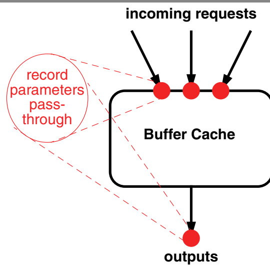
Measurement Thread



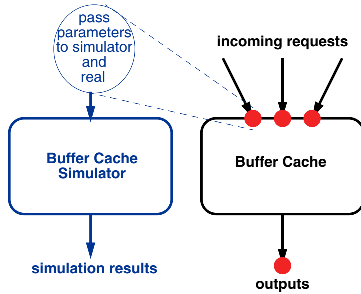
Generating Traces and Logs



Generating Traces and Logs



In-Situ Simulation



What do we do with Data?

- Off-line Analysis
  - *Monitors long-term behavior.*
  - *Identifies common usage profiles.*
  - *Detects uncommon usage.*
  - *Suggests thresholds to online system.*
  - *Conducts feasibility evaluations.*
- Online Analysis
  - *Monitor instantaneous resource utilization.*
  - *Maintain efficiency statistics.*
  - *Detect dangerous conditions.*

Off-line Analysis

- Use data from measurement thread to construct time series usage profile.
- Conduct variance analysis.
- Construct predicted usage profiles.
- Determine resource thresholds from predicted profiles.
- Notify online system of thresholds.
- Evaluate traces and logs; derive new algorithms.
- Simulate new algorithms, in situ.

Online Analysis

- Receive threshold and variance information from off-line system.
- Maintain dynamic statistics about:
  - *Cache hit rates.*
  - *Lock contention.*
  - *Disk queue lengths.*
  - *Load averages.*
  - *Context switch rates.*
- Detect abnormal behavior.
- Dynamically trigger trace generation.
- Trigger adaptation heuristics.

Adaptation Heuristics

- Goal: decrease application latency.
  - *Paging*
  - *Collect page access trace.*
  - *Look for well-known patterns (linear, cyclic, strided).*
  - *Look for page access correlation.*
  - *Install better prefetching algorithm.*
- Disk Wait
  - *Similar process to paging.*
  - *Replace read-ahead for the application(s).*
- CPU Hogs
  - *Examine profile output.*
  - *Recompile kernel modules in application context.*

Adaptation (continued)

- Interrupt Latency
  - *Measure latency between interrupt arrival and delivery to process/thread.*
  - *Look for excessively long intervals or high variance.*
  - *Check (fix) scheduling priorities.*
- Lock Contention
  - *Measure lock wait times.*
  - *Decrease lock granularity on highly contested items.*

Conclusions

- Extensibility is a powerful tool.
- More than just a "performance hack".
  - *Simplifies system monitoring.*
  - *Enables dynamic system tuning.*
  - *Provides potential for better system/application integration.*
- Kernel extensibility should be central paradigm.
  - *Allow extensible applications to take advantage of kernel's extensibility framework.*