CSE 221 Spring 2000 Wrapup

- Review of papers
- Course evaluations
- Paper feedback
Congratulations

- Tough course workload
- 34 papers in 10 weeks
- Regardless of what you learn about OS, hopefully you’re better at reading research papers
Quick Review

- Quick review of the papers that we read
- Highlight the main points, what the papers tend to be remembered for
THE and RC 4000

- Dijkstra, “THE”
  - Abstraction: sequential processes
  - Structure: layered
  - Sync/Comm: semaphores
- Brinch Hansen, “Nucleus (RC 4000)”
  - Abstraction: concurrent processes
  - Structure: “nucleus”, supports multiple OSes concurrently (early microkernel)
  - Sync/Comm: message passing
- Note semaphore (shared mem)/message passing (separate mem) theme
TENEX and Hydra

- Bobrow, “TENEX”
  - Time-sharing (interaction highest priority)
  - Abstraction: “virtual machine” (contemporary process abst.)
  - Structure: kernel, user-level (monolithic)
  - Rich VM semantics
    - Paged VM (BBN Pager), sharing via paging h/w, copy-on-write

- Hydra
  - Structure: “Nucleus” approach (prims for OS subsystems)
  - Separation of mechanism and policy (remember SPIN, Exo)
    - Protection is a mechanism, security is a policy
  - Capabilities, protected “objects” (reject layered hierarchy)
  - Rights amplification
Multics Protection and Unix

- Saltzer, “Multics Protection”
  - Uniform protection model for code, data, files, exec, etc.
  - Segmented memory, protection rings
  - Protected procedure call
    - Access control to capabilities
    - ACLs protect entry (slow), descriptors protect access (fast)
  - Principle of least privilege (e.g., root on Unix)

- Ritchie, “Unix”
  - “Full exploitation of a carefully selected set of fertile ideas”
  - Clean, uniform abstractions for files/devices/process mgmt
Pilot and Opal

- Redell, “Pilot”
  - PC operating system, all resources devoted to single user
  - Single address space
    - Memory-mapped files
  - Language-based protection (Mesa, type-safe)

- Chase, “Opal”
  - Single address space (64-bit)
    - Sharing of complex data structures with low overhead
  - Separation of addressing and protection
    - Addresses just name data, does not guarantee access
StarOS and Medusa

- Jones, “StarOS”
  - Multiprocessor OS for CM*
  - Capabilities, message passing
  - Task forces (processes for parallel applications)
  - Complex: Many different features, abstractions

- Ousterhout, “Medusa”
  - “Clean” version of “StarOS”
  - Distributed OS partitioned across nodes (recall 3 types)
    - Invoking file system likely a remote operation
  - Coscheduling, spin-waiting
Duality and Upcalls

- Lauer, “Duality”
  - Message-passing vs. Procedural OS structures
  - Argue they are equivalent by substitution
  - Devil in the details

- Clark, “Upcalls (Swift)”
  - Procedure-based OS
  - Design motivated by need for fast network stacks
  - Eliminate asynchrony where possible, make synch proc calls
    » Upward control flow via proc calls (protection implications)
Grapevine and Emerald

- Schroeder, “Grapevine”
  - Wide-area, scalable distributed system
  - Message (mail) and registration service
  - Replication for fault-tolerance, availability
  - Tradeoff between transparency and ease of design/implement
    » Eventual consistency instead of strict consistency

- Jul, “Emerald”
  - Mobile objects (code and data), fine-grained
  - Transparent invocation
  - Performance via complex implementation
    » Diff layouts, proc calls depending upon mobility of object
  - Distributed garbage collection
Accent and LOCUS

- Rashid, “Accent”
  - Network OS, communication-oriented
  - Message-based IPC
    - Network transparent
  - OS services implemented as user-level processes
    - Kernel is network-ignorant; network implemented as process
  - Copy-on-write VM for efficient page transfers

- Popek, “LOCUS”
  - Distributed OS
    - Name and location transparency, process migration
  - Single, global, uniform FS name space
  - Reliable
    - Two-phased commit for FS changes across replicas
    - Automatic recovery when partitions merge
V Kernel and Sprite

- Cheriton, “V Kernel”
  - Network OS, diskless workstations
  - Remote operations do not suffer performance
  - Generic message-based IPC as good as specialized protocol
    - Small, fixed size for control
    - VM tricks for page transfers

- Ousterhout, “Sprite”
  - Network OS for diskless workstations
  - Name and location transparency, process migration
  - Cache consistency protocol for client and server caches
Arch/OS Interaction

- Ousterhout, “Why aren’t OSes getting faster…”
  - OS primitive performance not scaling with hardware
    » Context switch, copying (insufficient mem bandwidth)
  - OS needs to decouple from disk latencies
    » Otherwise will only speed up as slow as disk latencies
    » LFS...

- Anderson, “Interaction of…”
  - Architectures make implementation of primitives difficult
    » IPC (syscall), VM (traps), Threads (context switch)
  - Modularized OS will suffer as a result
    » Mach 2.5 vs. Mach 3.0
Monitors and Monitors

- Hoare, “Monitors”
  - Monitors as synchronization mechanism, OS structure
  - Synch on monitor proc entry, condition variables
  - Integrates scheduling with synchronization

- Lampson, “Monitors in Mesa (Experience)”
  - Experiences paper
  - Change semantics slightly (notify)
    - Performance
    - Remove scheduling from monitor
  - Exceptions a challenge
    - Need to prevent deadlock, release resources
  - C.f. Java
Remote Ops and RPC

- Spector, “Remote Ops”
  - Single model for remote communication
  - Specialize implementation, protocol for specific comm requirements
  - Reliability taxonomy
- Birrell, “Implementing RPC”
  - RPC optimizations
  - Minimize packet exchanges, connection maintenance, process switches
  - Piggyback ACKs w/ data packets
LRPC and Active Messages

- Bershad, “LRPC”
  - Highly-optimized local RPC implementation
  - Use client thread in server (diff exec stacks)
  - Minimize context switch, stub overhead, copying

- Von Eicken, “Active Messages”
  - Parallel machine, message passing
  - Low-level communication mechanism
  - Need to overlap comm and comp, minimize overhead
  - Active Messages call function on delivery, run to completion
Multics VM and Mach VM

- Bensoussan, “Multics VM”
  - Segments, paging
  - Files memory-mapped
  - Complex

- Rashid, “Mach VM”
  - Machine-independent VM interface
  - Two-level implementation
  - Copy-on-write
IVY and GMS

- Li, “IVY”
  - Distributed shared memory
  - Consistency protocols
  - Distributed, dynamic necessary for perf, scalability
  - Hint-based forwarding

- Feeley, “GMS”
  - Global memory management
    - Minimize all memory accesses cluster-wide
  - Page to idle memory in remote machines
  - Efficient approximation to global LRU replacement
FFS and LFS

- McKusick, “FFS”
  - Increase disk b/w utilization for file system
  - Larger blocks (fragmentation)
  - Shorter seeks (cylinder groups)
  - Device savvy (device parameters, optimized layout)

- Rosenblum, “LFS”
  - Further increase disk b/w utilization
    - Reduce need for synchronous disk operations
  - Treat file system as append log, write in large segments
    - All data and meta data written to log
  - Extra meta-data for finding standard meta-data in log
  - Cleaning is the challenge
**SPIN and Exokernel**

- Bershad, “SPIN”
  - Extensibility, safety, and performance
  - Application extensions downloaded into kernel
  - Extensions, kernel written in type-safe language
    » Safety and performance

- Engler, “Exokernel”
  - Kernel exposes all hardware, does minimal hardware multiplexing
  - OS functionality implemented in untrusted library OS
  - Library OS can be completely customized to application
What’s Next?

- Good final question
- Dunno, but here are some guesses
- OSes configured, specialized to environment
  - Scout-like
  - Windows: desktop, server, PDA, embedded devices…
  - Active network nodes, routers
- OS support for wide-area applications
  - WebOS, grid computing