

Realtime Issues

12/13/2011

CSC 2/456

1

Implementing Real-Time Systems

- In general, real-time operating systems must provide:
 - (1) Preemptive, priority-based scheduling
 - (2) Preemptive kernels
 - (3) Latency must be minimized

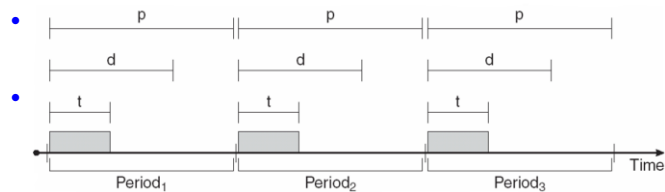
12/13/2011

CSC 2/456

2

Real-Time CPU Scheduling

- Periodic processes require the CPU at specified intervals (periods)
- p is the duration of the period

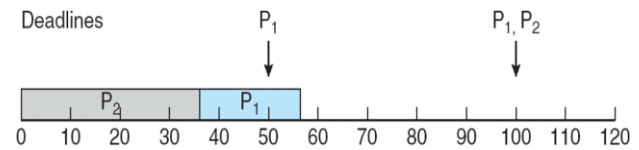


12/13/2011

CSC 2/456

3

Scheduling of tasks when P_2 has a higher priority than P_1



P1: period=50, processing time = 20, deadline = period
 P2: period=100, processing time = 35, deadline = period

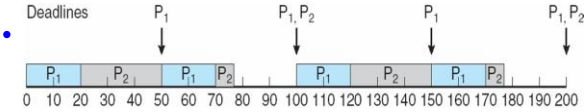
12/13/2011

CSC 2/456

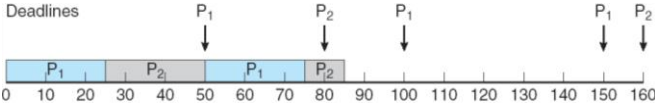
4

Rate Monotonic Scheduling

- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority



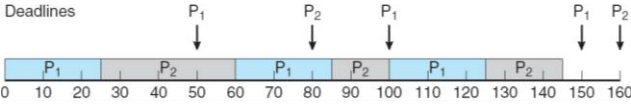
Missed Deadlines with Rate Monotonic Scheduling



Earliest Deadline First Scheduling

- Priorities are assigned according to deadlines:

the earlier the deadline, the higher the priority;



Multiprocessor Issues

Multiprocessor Scheduling

- Disabling signals not sufficient
- Acquire scheduler lock when accessing any scheduler data structure, e.g.,

yield:

```

disable_signals
acquire(scheduler_lock) // spin lock
enqueue(ready_list, current)
reschedule
release(scheduler_lock)
re-enable_signals
    
```

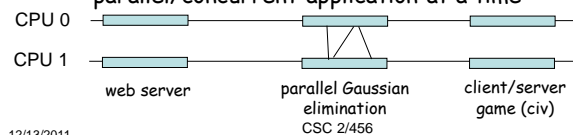
12/13/2011

CSC 2/456

9

Multiprocessor Scheduling

- Timesharing
 - similar to uni-processor scheduling - one queue of ready tasks (protected by synchronization), a task is dequeued and executed when a processor is available
- Space sharing
- cache affinity
 - affinity-based scheduling - try to run each process on the processor that it last ran on
- caching sharing and synchronization of parallel/concurrent applications
 - gang/cohort scheduling - utilize all CPUs for one parallel/concurrent application at a time



12/13/2011

CSC 2/456

10

Anderson et al. 1989 (IEEE TOCS)

- Raises issues of
 - Locality (per-processor data structures)
 - Granularity of scheduling tasks
 - Lock overhead
 - Tradeoff between throughput and latency
 - Large critical sections are good for best-case latency (low locking overhead) but bad for throughput (low parallelism)

12/13/2011

CSC 2/456

11

Performance Measures

- Latency
 - Cost of thread management under the best case assumption of no contention for locks
- Throughput
 - Rate at which threads can be created, started, and finished when there is contention

12/13/2011

CSC 2/456

12

Optimizations

- Allocate stacks lazily
- Store deallocated control blocks and stacks in free lists
- Create per-processor ready lists
- Create local free lists for locality
- Queue of idle processors (in addition to queue of waiting threads)

12/13/2011

CSC 2/456

13

Ready List Management

- Single lock for all data structures
- Multiple locks, one per data structure
- Local freelists for control blocks and stacks, single shared locked ready list
- Queue of idle processors with preallocated control block and stack waiting for work
- Local ready list per processor, each with its own lock

12/13/2011

CSC 2/456

14

Multiprocessor Scheduling in Linux 2.6

- One ready task queue per processor
 - scheduling within a processor and its ready task queue is similar to single-processor scheduling
- One task tends to stay in one queue
 - for cache affinity
- Tasks move around when load is unbalanced
 - e.g., when the length of one queue is less than one quarter of the other
 - which one to pick?
- No native support for gang/cohort scheduling or resource-contention-aware scheduling

12/13/2011

CSC 256/456

15

SMP-CMP-SMT Multiprocessor

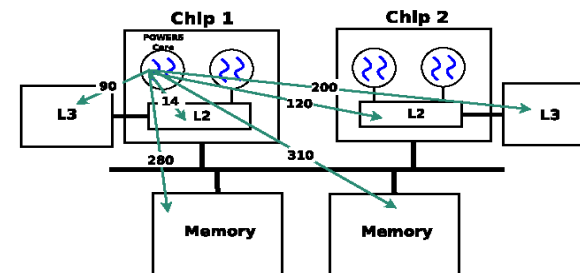


Image from <http://www.eecg.toronto.edu/~tamda/papers/threadclustering.pdf>

12/13/2011

CSC 256/456

16

CPU Scheduling on Multi-Processors

- Cache affinity
 - keep a task on a particular processor as much as possible
- Resource contention
 - prevent resource-conflicting tasks from running simultaneously on sibling processors

12/13/2011

CSC 2/456

17

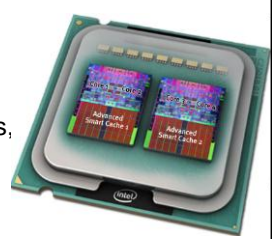
Resource Management To Date

- Capitalistic - generation of more requests results in more resource usage
 - Performance: resource contention can result in significantly reduced overall performance
 - Fairness: equal time slice does not necessarily guarantee equal progress

18

The Multi-Core Challenge

- Multi-core chip
 - Dominant on market
 - Last level on-chip cache is commonly shared by sibling cores, however sharing is not well controlled
- Challenge: Performance Isolation
 - Poor & unpredictable performance
 - Denial of service attacks



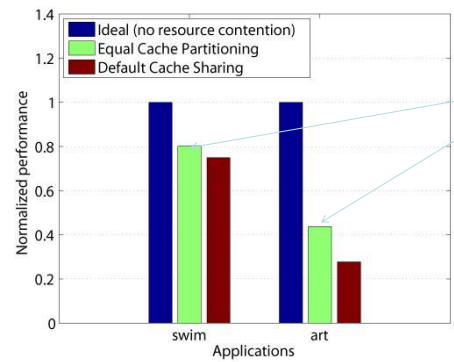
source: <http://www.intel.com>

12/13/2011

CSC 2/456

19

Poor Performance Due to Uncontrolled Resource Contention

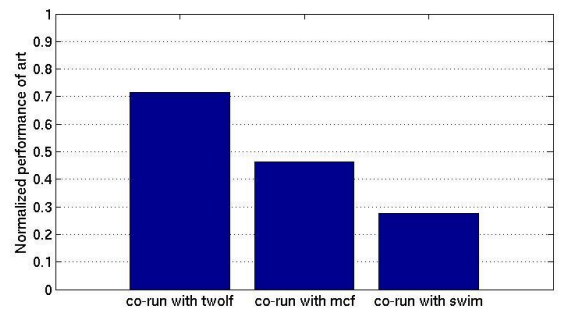


Win-win situation

Experiments were conducted on a 3Ghz Intel Core 2 Duo processor with a shared 4MB L2 cache

Fluctuating Performance Due to Uncontrolled Resource Contention

Performance of art when co-running with different applications on an Intel dual-core processor with a 4MB shared L2 cache



Fairness and Security Concerns

- Priority inversion
- Poor fairness among competing applications
- Information leakage at chip level
- Denial of service attack at chip level



Resource Contention-Aware Scheduling I

- Hardware resource sharing/contention in multi-processors
 - SMP processors share memory bus bandwidths
 - Multi-core processors share L2 cache
 - SMT processors share a lot more stuff
- An example: on an SMP machine
 - a web server benchmark delivers around 6300 reqs/sec on one processor, but only around 9500 reqs/sec on an SMP with 4 processors
- Contention-reduction scheduling
 - co-scheduling tasks with complementary resource needs (a computation-heavy task and a memory access-heavy task)
 - In [Fedorova et al. USENIX2005], IPC is used to distinguish computation-heavy tasks from memory access-heavy tasks

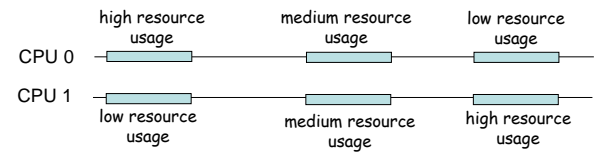
12/13/2011

CSC 2/456

23

Resource Contention-Aware Scheduling II

- What if contention on a resource is unavoidable?
- Two evils of contention
 - high contention \Rightarrow performance slowdown
 - fluctuating contention \Rightarrow uneven application progress over the same amount of time \Rightarrow poor fairness
- [Zhang et al. HotOS2007] Scheduling so that:
 - very high contention is avoided
 - the resource contention is kept stable



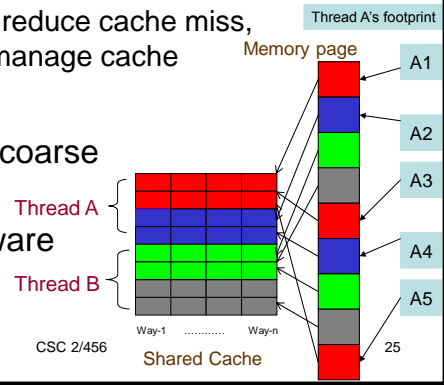
12/13/2011

CSC 2/456

24

Existing Mechanism(I): Software based Page Coloring

- Classic technique to reduce cache miss, now used by OS to manage cache partitioning
- Partition cache at coarse granularity
- No need for hardware support

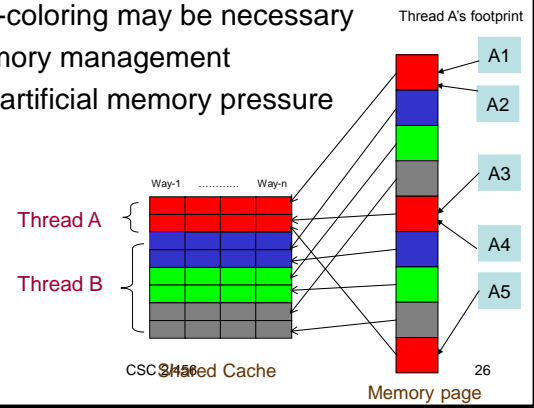


12/13/2011

CSC 2/456

Drawbacks of Page Coloring

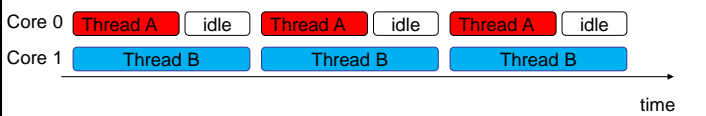
- Expensive re-coloring cost
 - Prohibitive in a dynamic environment where frequent re-coloring may be necessary
- Complex memory management
 - Introduces artificial memory pressure



12/13/2011

Existing Mechanism(II): Scheduling Quantum Adjustment

- Shorten the time slice of app that overuses cache
- May let core idle if there is no other active thread available



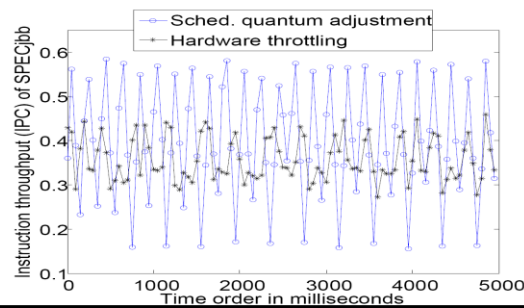
12/13/2011

CSC 2/456

27

Drawback of Scheduling Quantum Adjustment

- Coarse-grained control at scheduling quantum granularity may result in fluctuating service delays for individual transactions



12/13/2011

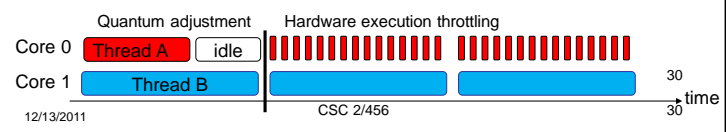
28

New Mechanism: Hardware Execution Throttling

- Throttle the execution speed of app that overuses cache
 - Duty cycle modulation
 - CPU works only in duty cycles and stalls in non-duty cycles
 - Different from Dynamic Voltage Frequency Scaling
 - Per-core vs. per-processor control
 - Thermal vs. power management
 - Enable/disable cache prefetchers
 - L1 prefetchers
 - IP: keeps track of instruction pointer for load history
 - DCU: when detecting multiple loads from the same line within a time limit, prefetches the next line
 - L2 prefetchers
 - Adjacent line: Prefetches the adjacent line of required data
 - Stream: looks at streams of data for regular patterns

Comparison of Hardware Execution Throttling to other two mechanisms

- Comparison to page coloring
 - Little complexity to kernel
 - Code length: 40 lines in a single file, as a reference our page coloring implementation takes 700+ lines of code crossing 10+ files
 - Lightweight to configure
 - Read plus write register: duty-cycle 265 + 350 cycles, prefetcher 298 + 2065 cycles
 - Less than 1 microseconds, as a reference re-coloring a page takes 3 microseconds
- Comparison to scheduling quantum adjustment
 - More fine-grained controlling



Review

CS 256/456
 Dept. of Computer Science, University
 of Rochester

What is an Operating System?

- An extended machine
 - Hides the messy details about hardware resources
 - Presents users with a resource abstraction that is easy to use
- A resource manager
 - Allows multiple users/programs to share resources fairly, efficiently, ...

Overall Picture

- OS components
 - Process Management and Scheduling
 - Synchronization
 - Memory Management
 - I/O System Management
 - File and Secondary-Storage Management
- OS structure/organization
 - Monolithic kernel
 - Micro-kernel (and exokernel)
 - Virtual machines

12/13/2011

CSC 2/456

33

Topics Covered

- Processes and threads
- Signals
- Synchronization
 - Classical problems and synchronization primitives
 - Synchronization within the kernel
- CPU scheduling
 - Uniprocessor and multiprocessor scheduling
 - Real-time and user-level scheduling
- Memory management
 - Virtual memory
 - Replacement policies
 - Page coloring and address translation
- I/O systems and storage devices
- File systems
 - Traditional and distributed file systems
- Security and protection
- Multiprocessor Oses
- Software-isolated processes
- Reinforcement learning in Oses
- Virtual machines
- Embedded (sensor, smart phone) Oses
- Instructional Oses

12/13/2011

CSC 2/456

34

Processes & Threads

- Process
 - Process concept
 - OS data structure for a process
 - Operations on processes
- Thread
 - Thread concept
 - Compared with process
 - less context switch overhead
 - more efficient synchronization between threads
 - User/kernel threads

12/13/2011

CSC 2/456

35

CPU Scheduling

- Selects from among the processes/threads that are ready to execute, and allocates the CPU to it.
- CPU scheduling may take place at:
 1. Hardware interrupt/software exception.
 2. System calls.
- Scheduling schemes:
 - FCFS
 - Shortest job first
 - Priority scheduling
 - Round-robin
- CPU scheduling in practice

12/13/2011

CSC 2/456

36

Synchronization

- Concurrent access to shared data may result in race condition
- The Critical-Section problem
 - Pure software solution
 - With help from the hardware
- Synchronization without busy waiting
 - Semaphore
 - Mutex lock

12/13/2011

CSC 2/456

37

High-level Synchronization

- Classic synchronization problems
 - Bounded buffer (producer/consumer)
 - Dining philosopher
- High-level synchronization primitives
 - Monitor
 - Condition variables

12/13/2011

CSC 2/456

38

Deadlocks

- Deadlocks
 - Four criteria: Mutual exclusion, Hold and wait, No preemption, Circular wait
- Handling deadlocks:
 - Ignore the problem and pretend that deadlocks will never occur
 - Ensure that the system will *never* enter a deadlock state
 - deadlock prevention
 - deadlock avoidance
 - Allow the system to enter a deadlock state and then detect/recover.

12/13/2011

CSC 2/456

39

Virtual Memory

- **Virtual memory** - separation of user logical memory from physical memory
 - Only part of the program address space needs to be in physical memory for execution
 - Copy-on-write: allows for more efficient process creation
 - Memory-mapped I/O
- Page replacement algorithm: the algorithm that picks the victim page
 - FIFO, Optimal, LRU, LRU approximation

12/13/2011

CSC 2/456

40

Memory Management

- Address binding
 - compile-time, load-time, execution-time
 - Logical vs. physical address
- Memory management
 - space allocation & address translation (memory mapping unit)
- Paging (non-contiguous allocation)
 - address translation: page tables and TLB
 - hierarchical page tables, inverted/hashed page tables
- Segmentation
 - compile time: segmented logical addresses
 - execution time: translated into physical addresses

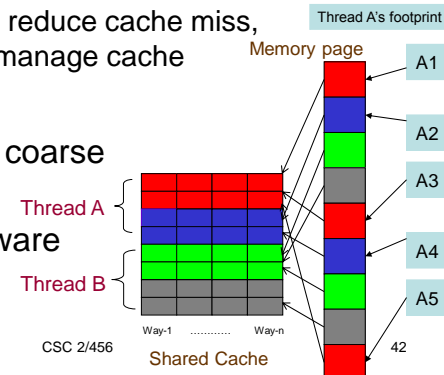
12/13/2011

CSC 2/456

41

Page Coloring

- Classic technique to reduce cache miss, now used by OS to manage cache partitioning
- Partition cache at coarse granularity,
- No need for hardware support



12/13/2011

CSC 2/456

42

I/O & Storage Systems

- I/O:
 - interrupt-driven
- Disk Structure
- Disk Scheduling
 - FCFS, SSTF, elevator, anticipatory scheduling

12/13/2011

CSC 2/456

43

File Systems

- File system interface
 - files/directories
 - access models and operations
- Space allocation for disk files
 - contiguous allocation, linked allocation, indexed allocation
 - space efficiency and access efficiency (random/sequential)
- Free space management
 - bit map, linked list,
- I/O buffer management
 - caching and prefetching

12/13/2011

CSC 2/456

44

Log-Structured File Systems

- With CPUs faster, memory larger
 - buffer caches can also be larger
 - most of read requests can come from the memory cache
 - thus, most disk accesses will be writes
 - poor disk performance when most writes are small
- LFS Strategy [Rosenblum&Ousterhout SOSP1991]
 - structures entire disk as a log
 - always write to the end of the disk log
 - when updates are needed, simply add new copies with updated content; old copies of the blocks are still in the earlier portion of the log
 - periodically purge out useless blocks

12/13/2011

CSC 2/456

45

Log-structure File Systems and Solid State Drives

- Log-structure file systems
 - Improve individual disk write throughput when using large caches for reads
 - Improve reliability and recovery overhead via journaling
- Solid State Drives
 - Fast reads, random access
 - Slow write/erase (in blocks) cycle
 - Finite number of writes requiring wear leveling

12/13/2011

CSC 2/456

46

Security

- User authentication
 - UNIX user authentication and attacks
 - Login spoofing
- Buffer overflow attack
- Viruses and anti-virus techniques
- Disk encryption and data security

12/13/2011

CSC 2/456

47

Protection

- Operating system consists of a collection of objects, hardware or software (e.g., files, printers)
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so
- Access control lists & capabilities

12/13/2011

CSC 2/456

48

Overall Picture

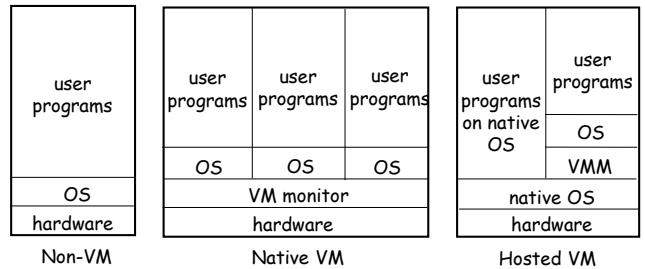
- OS components
 - Process Management and Scheduling
 - Synchronization
 - Memory Management
 - I/O System Management
 - File and Secondary-Storage Management
- OS structure/organization
 - Monolithic kernel
 - Micro-kernel (and exokernel)
 - Virtual machines

Microkernel

- Microkernel structure:
 - Moves functionalities from the kernel into "user" space.
- Benefits:
 - Modular design
 - More reliable (less code is running in kernel mode)
- Disadvantage:
 - Tend to have more frequent domain crossings (performance)
- Two types of micro-kernels:
 - Running user-level OS in a trusted server - Mach
 - Running user-level OS within untrusted user processes - Exokernel
 - more secure (less trusted code)
 - more flexibility (user-level customization is easy)

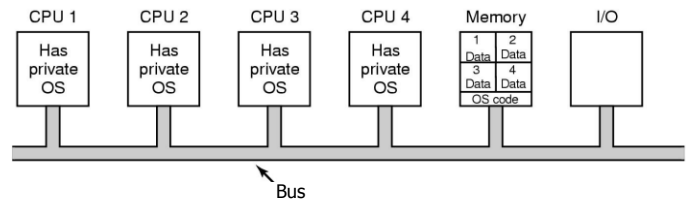
Virtual Machines

- Virtualization: provides isolated duplicates of the real machine



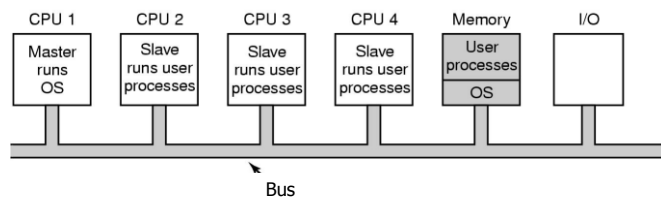
Need architecture where "sensitive" instructions are a subset of "privileged" instructions

Multiprocessor OS



- Each CPU has its own operating system
 - quick to port from a single-processor OS
- Disadvantages
 - difficult to share things (processing cycles, memory, buffer cache)

Multiprocessor OS – Master/Slave



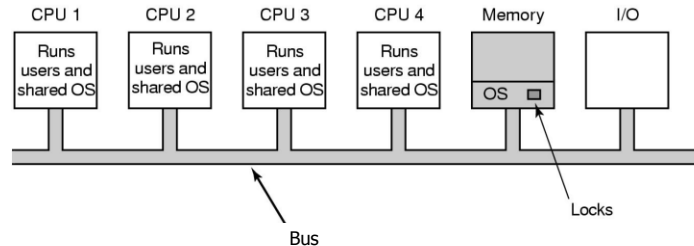
- All operating system functionality goes to one CPU
 - no multiprocessor concurrency in the kernel
- Disadvantage
 - OS CPU consumption may be large so the OS CPU becomes the bottleneck (especially in a machine with many CPUs)

12/13/2011

CSC 2/456

53

Multiprocessor OS – Shared OS



- A single OS instance may run on all CPUs
- The OS itself must handle multiprocessor synchronization
 - multiple OS instances from multiple CPUs may access shared data structure

12/13/2011

CSC 2/456

54

Multiprocessor OSes

- Issues
 - Synchronization
 - E.g., Linux kernel synchronization issues
 - Scheduling
 - Time sharing
 - Space sharing
 - Cache affinity
 - Cache sharing and application coordination: gang/cohort scheduling

12/13/2011

CSC 2/456

55

Distributed File Systems: Issues

- Naming and transparency (location transparency versus location independence)
 - Host:local-name
 - Attach remote directories (mount)
 - Single global name structure
- Remote file access
 - Remote-service mechanism
 - Stateful vs. stateless
 - Caching and coherence
 - Cache update policy (write through vs. delayed write)
 - Client-initiated vs. server-initiated
- Reliability and file replication
 - Naming transparency
 - Availability vs. consistency

12/13/2011

CSC 2/456

56

The Google File System

- Large, distributed, data-intensive workload
- Large streaming read/write with mainly file appends
- Centralized management of file metadata
- Data chunked (64MB) across servers
- Fault tolerance via replication

12/13/2011

CSC 2/456

57

Singularity

- Software/language-based protection and isolation
 - Removes need for hardware protection mechanisms
 - Moves error detection closer to design time
 - Single address space
 - Requires all processes to use type-safe language
 - All communication via messages

12/13/2011

CSC 2/456

58

Topics Covered

- Processes and threads
- Signals
- Synchronization
 - Classical problems and synchronization primitives
 - Synchronization within the kernel
- CPU scheduling
 - Uniprocessor and multiprocessor scheduling
 - Real-time and user-level scheduling
- Memory management
 - Virtual memory
 - Replacement policies
 - Page coloring and address translation
- I/O systems and storage devices
- File systems
 - Traditional and distributed file systems
- Security and protection
- Multiprocessor Oses
- Software-isolated processes
- Reinforcemet learning in OSes
- Virtual machines
- Embedded (sensor, smart phone) Oses
- Functional Oses

12/13/2011

CSC 2/456

59