CSE 451: Operating Systems

Section 10

Final exam review

Final exam review

- Disclaimer: This is not guaranteed to be everything that you need to know for the final. This is an overview of major topics we covered in the course.
- You are responsible for all the readings and the slides only up to what we covered in class.

Exam Coverage

- Lectures: Modules 1 – 28
  - Chapters 1 – 14 in the textbook
- Sections:
  - All examples and problems gone over in section. (Not including Andriod architecture)
- Extra Readings

Major Topics

- Kernels – Micro, Monolithic, etc
- Processes – fork, vfork, execve, clone
- User and Kernel level threads
- Scheduling, overview of scheduling algs
- Paging, caching
- Memory Management
- Race conditions and synchronization variables
- Deadlock
- File systems

Kernel land vs User land separation

- Userspace processes cannot interact directly with hardware (non-privileged mode)
- Attempting to execute a system call instruction causes a trap to the kernel (privileged mode), which handles the request
- Why is it necessary to have both privileged and non-privileged mode?
- How is privileged mode enforced?
- What kind of operations require a system call?

IO from userspace

- Userspace processes interact with disks and other devices via `open()`, `read()`, `write()`, and other system calls
- Multiple levels of abstraction: kernel presents file system to userspace, and device drivers present a (mostly) unified interface to kernel code
  - What are the benefits and drawbacks of designing a system in this way?

Monolithic and microkernels

- Monolithic kernels encapsulate all aspects of functionality aside from hardware and user programs
  - Pro: Low communication cost, since everything is in the kernel's address space
  - Cons: Millions of lines of code, continually expanding, no isolation between modules, security
- Microkernels separate functionality into separate modules that each expose an API
  - Services as servers
  - Why? How?

Memory management

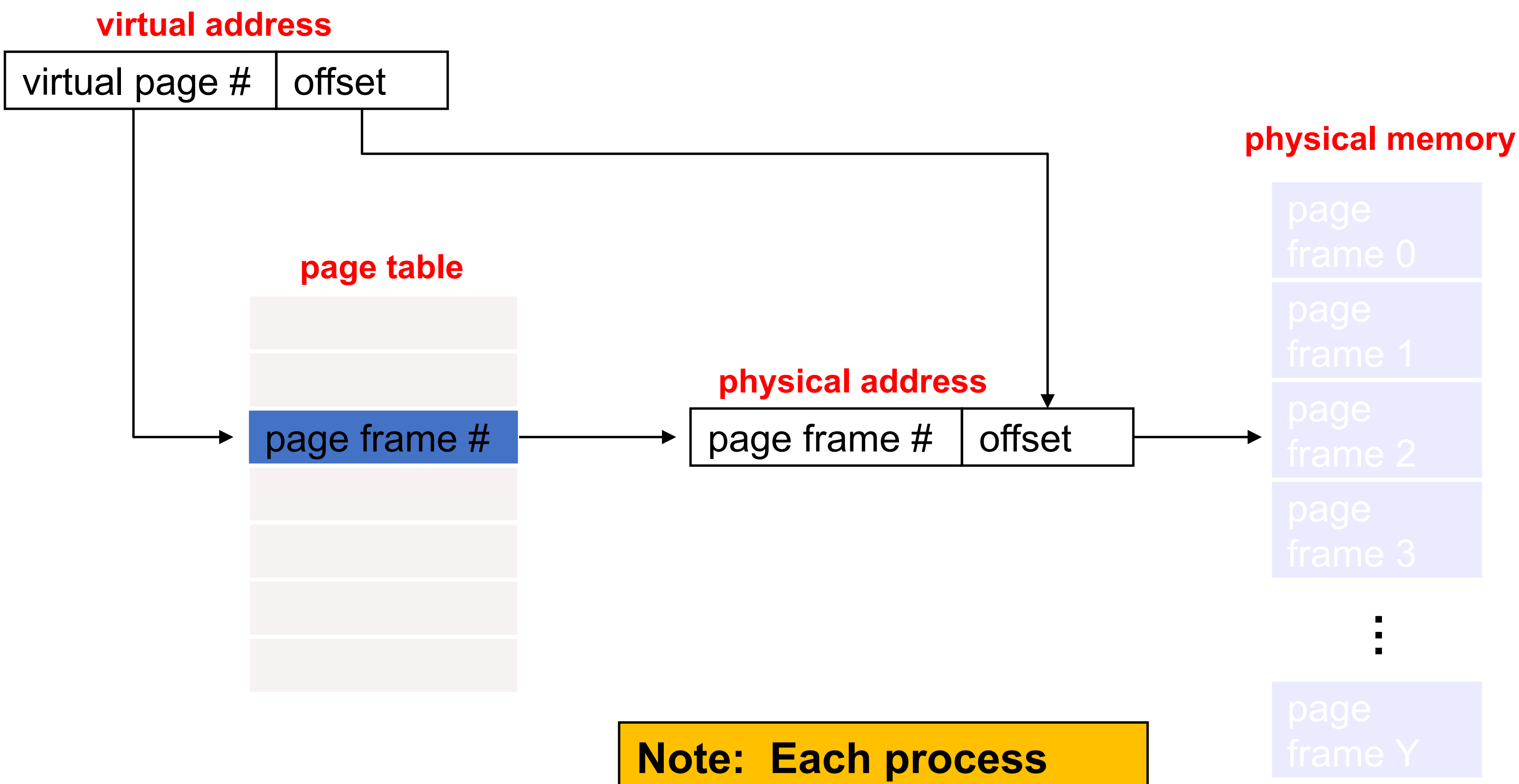
- Purposes:
  - Resource partitioning / sharing
  - ***Isolation***
  - Usability
- Paging
- Segmentation

Virtual memory

- What happens on a virtual memory access?
- How does the TLB, multilevel page tables, segmentation faults, page faults, and disk all work together to control memory access?

Virtual memory

Remember that a page table can have multiple levels.



Note: Each process has its own page table!

Page replacement

- Algorithms:
  - Belady, FIFO, LRU, LRU clock / NRU, random, working set…
  - Local vs. global
- How/why are any of these better or worse than the others?
- What happens when paging goes wrong?
  - Thrashing, 10-year old computers running XP?

Advanced virtual memory

- What problem does a TLB address?

- What problem do two-level page tables address?
  - What's the key concept?

Advanced virtual memory

- What problem does a TLB address?
  - Increases speed of virtual address translation
- What problem do two-level page tables address?
  - What’s the key concept?
    - Indirection

Processes versus threads

- Processes have multiple pieces of state associated with them
  - Program counter, registers, virtual memory, open file handles, mutexes, registered signal handlers, the text and data segment of the program, and so on
  - Total isolation, mediated by the kernel
- Threads are “lightweight” versions of processes
  - Which pieces of state listed above do threads not maintain individually?

Process creation

- `fork()` : create and initialize a new process control block
  - Copy resources of current process but assign a new address space
  - Calls to `fork()` return twice—once to parent (with pid of child process) and once to child
  - What makes this system call fast even for large processes? `vfork()` versus copy-on-write
- Difference between `fork(), vfork(), cow fork(), and clone()?`
- `exec()` : stop the current process and begin execution of a new one
  - Existing process image is overwritten
  - No new process is created
  - Is there a reason why `fork()` and `exec()` are separate system calls?

Process State

- What States can a process be in?

Process State

- What States can a process be in?
  - Running, Runnable, Waiting
- How does a process between the different states?

Threads

- How is a kernel thread different from a userspace thread?
  - Kernel thread: managed by OS, can run on a different CPU core than parent process
  - Userspace thread: managed by process/thread library, provides concurrency but no parallelism (can't have two userspace threads within a process executing instructions at the same time)
- CPU sharing
  - Threads share CPU either implicitly (via preemption) or explicitly via calls to `yield()`
  - What happens when a userspace thread blocks on IO?

Scheduling

- Operating systems share CPU time between processes by context-switching between them
  - In systems that support preemption, each process runs for a certain quantum (time slice) before the OS switches contexts to another process
  - Which process runs next depends on the scheduling policy
- Scheduling policies can attempt to maximize CPU utilization or throughput or minimize response time, for example
  - There are always tradeoffs between performance and fairness

Scheduling policies

- FIFO: first in first out
- SPT: shortest processing time first
- RR: round robin
- Any of these can be combined with a notion of Priority
  - How to avoid starvation? Lottery is one option
- What are the benefits and drawbacks of each type of scheduling policy?

Scheduling MLFBQ

- Queue of queues
  - Priority based on top level queue depth
- Defined by:
  - The number of queues
  - The scheduling algorithm for each internal queue which can be different from FIFO
  - The method used to determine when to promote a process to a higher priority queue
  - The method used to determine when to demote a process to a lower priority queue
  - The method used to determine which queue a process will enter when that process needs service

Synchronization Variables

- Locks, mutexes, semaphores, condition variables and monitors
  - Mutexes
    - Provide a waiting queue for threads that are waiting on a lock
  - Condition Variables
    - A higher level construct than mutexes. They help manage the waiting of threads by allowing them to wait until a given condition is true
    - Signal and broadcast
  - Monitors
    - Two main different types, Hoare and Mesa monitors.
    - Provides object like abstraction to synchronization. Manages condition variables and locks as well as provides methods for accessing shared memory.
    - Should be familiar with both types: http://en.wikipedia.org/wiki/Monitor_(synchroniza$on)

Thread management

- Queues
  - Why do thread libraries make use of queues?
- Synchronization
  - What are the mechanisms for protecting critical sections, how do they work, and when should one be used over another?
- Preemption
  - What is preemption and how does the process of one thread preempting another work?

Secondary storage

- Memory forms a hierarchy
- Different levels of disk abstraction:
  - Sectors
  - Blocks
  - Files
- What factor most influences the ways that we interact with disks?

Secondary storage

- Memory forms a hierarchy
- Different levels of disk abstraction:
  - Sectors
  - Blocks
  - Files
- What factor most influences the ways that we interact with disks?
  - Latency

File systems

- What does a file system give you?
  - Useful abstraction for secondary storage
  - Organization of data
    - Hierarchy of directories and files
  - Sharing of data

File system internals

- Directories
- Directory entries
- Inodes

- Files:
  - One inode per file
  - Multiple directory entries (links) per file

Inode-based file system

- Sequence of steps when I run *echo "some text" > /home/jay/file.txt* ?
  - Open file:
  - Write to file:
  - Close file:

Inode-based file system

- Sequence of steps when I run *echo “some text” > /home/jay/file.txt* ?
  - Open file:
    - Get inode for / -> get data block for /
    - Read directory entry for / -> get inode for /homes
    - Repeat… -> get data block for file.txt, check permissions
  - Write to file:
    - Modify data block(s) for file.txt in buffer cache
  - Close file:
    - Mark buffer as dirty, release to buffer cache
    - Kernel flushes dirty blocks back to disk at a later time