ICS 332 Operating Systems review for exam 2

- exam will be Monday, October 29th, 2018
- same general format as exam 1
- questions may be from:
  - the lectures (lecture notes, material linked from the web page)
  - the assignments
  - the textbook
- answering questions may require knowing the material that was already tested in exam 1

Outline

- scheduling of processes and threads
- race conditions
- synchronization primitives, including atomic operations, spinlocks, mutexes, and java synchronized methods
- deadlocks
- counting, addressing, exponents (and 2^x), logarithms
- main memory, swapping
- segmentation, and standard Unix segments
- allocation of contiguous blocks of memory
- static vs. dynamic loading, static vs. dynamic linking
- virtual memory

Schedulers

- process scheduling is very similar to thread scheduling
  - but the context switch for threads is more lightweight
  - and can be done by user code
- no single “best” scheduling policy
- long-term scheduler decides when to submit jobs to a system, short-term scheduler decides which task to execute next
- CPU-intensive tasks vs. I/O-intensive tasks
- process state machine: running, ready, blocked/waiting
  - also terminated (zombie), and others
- preemptive scheduler (time quantum), non-preemptive
- round-robin with priorities, but also many other algorithms

Race Conditions

- a **race condition** is when multiple threads are accessing shared memory
- operations that look atomic at the programming-language level, are not atomic at the machine level
- typical concern: incrementing a variable is not atomic
  - a value in a thread pre-empted long ago can still be stored back to memory!
- any section of code that assumes shared memory is in a consistent state is a *critical section*
- only one critical section should execute at a given time
  - for a given logical unit of shared memory

Syncronization Primitives

- atomic operations
  - e.g. test-and-set, or compare-and-swap, are often provided as machine instruction by the architectures
- spinlocks
  - loop (perhaps up to a fixed number of times) until the lock is freed
  - perhaps yield/suspend if the lock is not available
- mutexes
  - lock and unlock primitives
  - only one task can hold the lock at any time
- java synchronized methods
  - only one thread at a time can execute the synchronized method

Deadlocks

- three conditions:
  - mutual exclusion
  - no preemption
  - circular requests
- prevention: avoid circular requests
  - programmer: when requesting multiple locks, request them all at the same time, or in a specific sequence
- avoidance
  - scheduling: safe sequence
  - claim edges
- detection and recovery
  - kill one or more deadlocked processes until the deadlock goes away

Counting, Addressing, Exponents, Powers of two, Logarithms

- counting: bits and bytes, KB and KiB, MiB, GiB, TiB, ...
- addressing: how many bits do you need to address n things?
  - answer: ceiling ($\log_2$ n)
- exponents
- powers of 2
  should know 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}
- logarithms and the number of digits:
  - if n is written using d digits (in base b), then
  - $d - 1 \leq \log_d n < d$

Main Memory

- main memory
- absolute addressing and PC-relative addressing
- memory virtualization: the address seen by the *program* is different from the address seen by the *hardware*
- can be done with a **base register** which is added to every address
- swapping: the virtual address space may be on disk rather than in memory
  - (relevant parts of) the swapped-out process must be brought back into memory before the process can execute or load or store data

Segmentation

- an address includes a segment number and an offset
- a Memory Management Unit translates virtual to physical addresses
- the MMU uses the segment number to:
  - compare the offset to the per-segment limit register
  - add the per-segment base register to the offset
  - the segment number may be implicit or explicit – the x86 architecture has both
- segments may be used for unix-like segments (text, data, heap, and stack) or may be used for individual data structures
- either way, overflow and underflow are detected by the MMU

Standard Unix Segments

- text segment: the code to execute
  - with dynamic linking or loading, may have more than one text segment, but the official text segment is the one with the main function/method
- data segment: global variables
  - the values for initialized variables are in the executable
  - all other global variables initialized to zero
- heap segment: dynamic memory allocation
- stack segment: function/method call parameters and return addresses, local variables

Memory Allocation

- assume you want to allocate contiguous areas of memory
- or contiguous areas of the address space (the problem is the same)
- once you have non-contiguous allocated areas of memory, in-between them are areas of free memory
- this free memory may be sufficient for the next allocation, but no single block is large enough:
  - give up (`malloc` fails, or asks the OS for more memory)
  - relocate (some garbage collectors do this)
- first fit, best fit, worst fit

Static vs. Dynamic Loading and Linking

- dynamic loading:
  - user code loads a library and calls functions/methods from that library
  - explicit code in the user program
- dynamic linking:
  - compiler records that user code calls a library, but does not link to it until runtime
  - at runtime, each call to a library stub must be replaced by a call to the actual library function or method

Virtual Memory

- break up allocations into smaller units, use base and offset to address each unit
  - a kind of segmentation
- allocating variable-sized blocks of memory is NP-hard
- so instead, allocate fixed-sized pages
  - fixed-size means no need to keep per-page **limit**
- with page size a power of two (2^n), the low order n bits of the address are the page offset
- the remaining high order bits are the Virtual Page Number or the Frame Number
- page table has one entry per virtual page
- there will be more virtual memory in the next few lectures

Suggestions for Doing Well

- sleep well the night before the exam
- review all the material well in advance
- review again on the day of the exam
- practice problems
- review, understand the homeworks!
- read, understand the textbook!
  - and practice problems, at least to the point of sketching a solution
- read, understand the lecture notes!