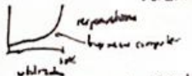
Matthew Tran
CS162
MT1

- OS provides abstractions from hardware, makes illusionist & referee & glue
- Process
  - address space
  - 1+ threads in
  - other (files, sockets, etc.)
- 4 OS Concepts
  - Thread of Control
    - lots of registers, executing when present, suspended when not loaded
    - illusion of multicore
    - TCB - stores info, in kernel for now
  - Address Space
    - address state, not only RAM, maybe IO, etc
    - need some area of memory allowed to use,
      - ex. Base and Bound, translation, paged memory
  - Process
    - execution environment w/ 1+ thread
    - protected from each other and from OS
    - threads = concurrency
  - Privilege (Dual Mode Operation)
    - ≥2 modes: kernel, user (0, 3)
    - Kernel mode transfer – syscall, interrupt, trap/exception
      - kernel specifies how to handle transfer, ex vector table
    - PCB - status, register state, etc maintained by kernel
    - scheduler picks which to run
- Thread Abstraction
  - concurrency ≠ parallelism, OS must handle multiple things at once
  - multiprocessing = multicore
  - multiprogramming = multithreading, concurrency by OS
  - states: RUNNING, READY, BLOCKED
  - IO - slow, so block thread until done
  - unique TCB + stack, but shared globals in process
  - non determinism - any order, any time
  - race condition - depends on the interleaving
  - synchronization
  - mutual exclusion - only one thread does thing
  - critical section - only one thread can execute at once
  - lock - only one thread holds, atomic ops
- Process
  - even shell is one, process manages other process
  - bootstrapping - kernel starts init, which starts other processes
  - syscalls - exit, fork, exec, wait, kill, sigaction
- File / IO
  - Semaphore - generalized lock
    - unsigned int, atomic op
    - P() == down()
    - V() == up()
  - everything is a file
  - file & directory
  - each process has current working directory (CWD)
  - streams FILE*, internal user buffer, high level
  - file descriptor fd, low level ← don't mix
    - buffered in kernel ← immediate response RAW
  - ioctl, dup, pipe
  - FILE internal buffer, so flush when flushes or fflush
    - basically faster bc minimizes syscalls
  - open file description maintained in kernel
    - file descriptor returned
    - forking copies the descriptor but both refer to same kernel file description
      - how we can do IPC, pipes
  - never fork() in multithreaded, only 1 thread copied
    - fork() copies everything
    - exec() safe bc replaces entire address space
    - don't fork() w/ FILE* bc don't know if flush
- IPC, Pipes, Sockets
  - comms looks like File I/O
  - using actual file is slow, so pipes! (1 way)
    - EOF when last process holding fd is closed
  - also need protocol, usually part of RPC facility
  - TCP Network Connect, Socket
    - 2 way queue
    - server socket: listen(), accept() (new socket)
    - thread pool to serve clients bc if too many
    - 5 tuple
      - source IP
      - dest IP
      - source Port
      - dest Port
      - Protocol
    - server port - usually well known; client port usually random
- Synchronization
  - Process life states - new, ready, running, waiting, terminated
  - multiple scheduler queues like for ready, disk, etc
  - scheduler/dispatcher gets control w/ preemption or voluntary
  - context switch - save stuff, move over
    - 10-100ms frequency
    - process switch takes 3-4 μsec
    - thread switch takes 100 nsec
    - yield() in userspace even cheaper
  - Thread Pool - top of stack, housekeeping
  - shared data corruption - locks
    - also done w/ lockstep, via bounded buffer ex. rdm
      - circular buffer - need
  - Lock Implementation
    - interrupt enable/disable - naive, doesn't scale
    - atomic load/store
    - sleep during lock, need to unlock
    - Atomic Read-Modify-Write Instruction
      - test&set() - set to 1 and return old value
      - swap - swap, x86
      - compare&swap - if true, set register
      - load-linked/store-conditional - stuff
    - Busy waiting - does nothing
      - bad for cache, CPU usage
      - priority inversion
      - better way to sleep()
    - Futex - fast userspace mutex
      - no syscalls, used in pthreads
  - Monitor - lock + condition variable
    - cv - list of threads waiting
    - wait() - atomic release lock, sleep, acquire
    - signal() - wake one
    - broadcast() - wake all
    - Hoare semantics - immediate transfer
    - Mesa - popular, signaler keeps lock, waiter on ready queue
    - popular but lock not acquired immediately, need while loop to check again
  - kernel thread and each user thread
  - Page Table for process
- Device Driver
  - top half: called by syscall
  - bottom half: interrupt handler
- x86 Assembly (32-bit)
  - 8 registers: eax, esp, ebp, ecx
  - AT&T Syntax
    - register %eax
    - immediate: $x
    - dereference: ( )
    - offset: $( )
    - inst source, destination
  - Caller (esp % 16 == 0)
    - args on stack, reverse order, 16 byte aligned
    - push RA
    - return, stack
  - Callee
    - bak ebp, esp onto ebp
    - return value save
    - restore esp
    - return
  - addl add, sub, mov
  - pushl, popl, call, leave, ret, pushal, popal
- Extra
  - security ops in userspace

Matthew Tran
CS162
MT2

- Scheduling
  - Assumptions
    - 1 program/user
    - 1 thread/program
    - independent progs
    - execution model
      - CPU bursts then IO
  - Goals
    - minimize response time
    - maximize throughput
    - fairness - sorts response time
  - FCFS
    - FIFO - first come first served, nonpreemptive
    - convoy effect - small task behind big
  - Round robin
    - preemption - each process gets "time quanta"
    - optimal q - big => FCFS, small => too many context switch
    - response time = time of finish - time queued
    - 10-100ms today; 0.1 ms for context switch
  - Strict Priority Scheduling
    - starvation - some don't run ± deadlock (← can cause)
    - priority inversion - high priority wait for lower priority
  - SJF/STCF (Shortest Job First)
  - SRTF (Shortest Remaining Time First)
    - preemptive version of SJF
    - SRTF/SJF best at minimizing average response time
    - can lead to starvation
    - can't predict how long process takes ↑ real world
      - adaptive - predict future based on past
  - lottery scheduling
    - each process gets tickets, randomly pick one
  - MLFS (Multi Level Feedback Scheduling)
    - multiple queue w/ diff priority w/ diff scheduler (RR, FCFS)
    - if timeout expires, move down, if not move up
    - IO stuff near top, CPU at bottom, like SRTF
    - countermeasure - do pointless IO to keep priority high
  - Multicore scheduling
    - very similar to normal, but maybe per-core scheduler data
    - cache - want to reassign task to same core
    - cache coherence big issue
    - spinlock - busy wait on something } more efficient when
    - gang scheduling/parallel apps - want to schedule multithread together
      - want to work in linear part of curve



- Device Driver → user → kernel IO → top half → bot half → HW
- Linux O(1) scheduler -
  - 140 priorities, 40 "nice", 100 kernel/RT
  - lots adhoc heuristics, too complex
  - Real Time scheduling
    - Predictability !!! NOT PERFORMANCE !!!
    - Hard RT - meet all deadlines (if possible)
      - Earliest Deadline First (EDF)
      - Least Laxity First (LLF)
      - Rate-Monotonic Scheduling (RMS)
      - Deadline Monotonic Scheduling (DM)
    - Soft RT - attempt to meet deadlines w/ high prob
- Earliest Deadline First (EDF)
  - tasks periodic w/ period P and computation C in each period
  - feasible if $\Sigma \frac{C}{D} \leq 1$
- work-conserving scheduler - if work to do, CPU never idle
- Starvation - task never gets run (but could under right circumstances)
  - FCFS - nonpreemptive, so possible
  - RR - no, but throughput sucks
  - Priority Scheduling - if strict, yeah, but also priority inversion!
    - priority donation!
  - SRTF/MLFQ - can starve b/c short runs first
  - Proportional Share Scheduling
    - idea: share CPU, but not strict, "proportionally"
    - no starvation! but random
  - Stride Scheduling
    - remove "luck for small number"
    - "stride" = W/N_i, more tickets N_i = smaller stride
    - each job "pass" counter; pick lowest, run, add stride to pass
    - can be messy
  - Linux Completely Fair Scheduler (CFS)
    - currently used, goal to perfectly divide
    - O(log n) heap, run thread w/ least time
    - good for interactive!
    - target latency - Quanta = tar/n ← can't be too big
    - throughput - avoid overhead, min time slice
- Deadlock
  - circular dependency
  - one fix - fixed resource acquire order
  - Requirements
    1. mutual exclusion
    2. Hold and wait
    3. no preemption (voluntary release)
    4. circular wait

- Virtual Memory
  - memory multiplexing
    - protection
    - translation
    - controlled overlap
  - Base & Bound
    - add base, check bound
  - MMU - translation box ↑
    - fragmentation, sparse table, sharing issues
  - Segmentation
    - diff segments have separate region, anywhere physically
    - segment map in processor ← base + limit, valid/invalid
      - entry per chunk
    - x86 has registers for this, but sh/s has built in queue to pass
    - many issues - variable chunks, mapping, fragmentation
  - Paging - fixes ↑
    - PTE - PPN + bits (read, write, etc.)
    - sharing? - simple! PTE point to same page, read only?
    - kernel PT size ∝ PTE, consumes user space
    - multi level PT
      - single table too long ← can cache PT
      - ex. 10-10-12 (4KB PTs) (1 page!)
      - higher levels can be null or in disk too
  - PTE - pointer to next PT or page + permission bits
    - x86 PTE - PPN (20 bit)
      - P: present/valid
      - W: writeable
      - U: user accessible
      - PWT: page write transparent
      - PCD: page cache disabled
      - A: accessed
      - D: dirty (PTE only) ← top level PT
      - PS: page size (directory only)
  - Multilevel Translation: Segments + Pages ← also like
    - segment - memory w/ variable size (base + limit)
    - page - equal size blocks ← more memory used!
    - seg → PT → PPN
    - context switch - save top level segment/PT
  - x86 segment descriptors
    - 6 segment registers - SS, DS, CS, ES, FS, GS
      - points to segment descriptor
      - G/L bit selects GDT/LDT (global/local descriptor table)
      - RPL - requestor privilege level
    - GDTR/LDTR hold pointer to these tables
      - G: granularity of segment (1 byte, 4KB)
      - DB: default operand size (16 bit, 32 bit)
      - A: anything
      - P: segment present
      - DPL: descriptor privilege level
      - S: system segment (syscall, etc. switch)
      - Type: code, data, segment
    - 1 GDT for all, diff set of LDT for each process
      - usually protection for code segments - legacy
      - modern - "flattened", 1 segment 4GB
    - Intel - mistake on GS or FS for Thread Local Storage (TLS)
      - modern for 64 bit - too long a line ↑ issue
  - Inverted Page Table
    - hash table, fixed size, good for 64 bit
    - poor cache locality, complex hash chains
  - MMU - does all the translation
  - TLB - cache for translations VPN → PPN
    - PT might be in normal cache
    - can multilevel
    - fully associative, w/ direct mapped in front
    - multiple TLBs?
    - context switch? invalidate TLB

- Deadlock detection - Resource allocation graph
  - n Threads, m Resource types ← w instances of each
  - request(), use(), release()
  - request edge - T → R
  - assignment edge - R → T
  - alg (cycle ≠ deadlock) but necessary
    - add all nodes to unfinished
    - available resource array
    - loop thru unfinished
      - if node request can satisfiable
        - remove
        - add request resource to available
        - not done yet
    - basically check if all nodes can eventually run
- Deadlock deal with
  1. prevention
  2. recovery
  3. avoidance
  4. denial
  - OS pretends own deadlock, that's it
  - Bankers Algorithm
    - only good if no deadlock (run potential deadlocks)
    - max-alloc instead of request

- Cache
  - 4 C's
    - compulsory - must
    - capacity - too small
    - conflict - 2 map to same
    - coherence - other updates
  - Address
    - block - offset
    - index - which to index
    - tag - ID
  - Types
    - Direct Mapped - 1 address possible line
    - Set Associative - index picks "set"
    - Fully Associative - 1 index, all tag
  - Replacement
    - Random
    - LRU
  - Write
    - write through - do both
    - write back - cache only, but dirty
  - Physically indexed vs. Virtual index cache
    - x86 - VA → TLB → Cache
    - VA → Cache (but flush on switch?)
  - overlap TLB and cache
    - index + byte = VA offset
    - virtual cache faster?

- Demand Paging
  - page fault - no translation fault/trap
  - main memory is "cache" for disk
    - block size - 1 page
    - fully associative
    - replacement policy?
    - write back b/c disk slow af
  - can share read only pages
    - also mmap?
  - OS maintains how to find in disk
  - Working Set Model
    - program use certain sections at a time
  - Replacement Policy
    - compulsory, capacity, conflict, policy miss
    - FIFO - fair, sucks
    - RANDOM
    - MIN - provably optimal, replace page won't be used in longest time
    - LRU - least recently used, approx of MIN
    - Belady's Anomaly - ↑ mem ↑ misses
      - affects FCFO
    - Clock Algorithm
      - approx of LRU - replace old, not oldest
      - all pages in circle, have use bit
      - HW sets use bit ← young or old
      - on fault - check bit, if 1 - clear, move on; 0 - replace
    - N^th chance algorithm
      - clock but N chances
      - if counter = N, replace; N sweeps
      - give dirty page extra chance?
    - Clock Alg Variation
      - no "modified" bit ← dirty bit
        - use "read only" bit + OS for page on write
        - if write, otherwise, no sw mod bit
      - no "use" bit too
        - also emulate in SW
        - mark all page invalid first
    - Second Chance Alg (VAX/VMS)
      - active list, SC list
        - FIFO, LRU
      - SC list access is page fault
        - move to front of active
        - move active to SC
        - not in SC, LRU and bring in
    - Free List
      - similar to SC
      - filled by diff alg like clock
  - Reverse Page Mapping "Coremap"
    - which PTE to invalidate?
    - Linux - object based reverse mapping
  - Page Frame Allocation
    - each process need minimum
    - global or local replacement?
    - equal/proportional/priority allocation
  - Compulsory miss? bring in multiple pages? track working set?
- Meltdown - exploit speculative to read kernel mem
  - patch - diff page tables for user and kernel, flush TLB twice on syscall, slow

- IO
  - Bus - multiple devices, one at once,
  - PCI - parallel
  - PCIe - more serial
  - PMIO - in/out instructions
  - MMIO - just memory
  - Programmed IO - CPU does each byte
  - DMA - controller handles transfer
  - Interrupt vs. Polling
    - ↑ hardware / wake cycle
- Device Driver
  - user → kernel IO → top half → bot half → HW
  - standard calls
  - Block device - block
  - Character Device - byte
  - Network Device - socket
- blocking, nonblocking, async
- HDD
  - cylinder, multiplatter
  - seek time - head
  - rotational latency - wait disk (← rotation time)
  - transfer time
- SSD
  - limited write, also block only
  - FTL - flash translation layer
    - virtual mem but for flash
    - just write to new page instead
    - wear leveling!
    - garbage collection

in each dirty, ed., undefiled, valid

Matthew Tran
CS162
MT3

- IO
  - Block, character, network devices
    - get/put, socket
  - blocking, nonblocking, asynchronous interface
  - HDD
    - sectors → ring (track) separated by guard regions
    - outside tracks have more sectors
    - SMR – overlapping tracks = ↑ density/capacity
      - but write harder, DSP for reading
    - cylinder – all tracks under head at point (multiplatter)
    - latency = queue time + controller + seek + rotation + transfer
      - seek – move head over track ~4–6ms
      - rotational – wait for sector ~4–8ms (half rot)
      - transfer – transfer bits ~sweep, 25–50 MB/s
    - lots of ECC
  - SSD
    - latency = queue + controller + transfer
    - write only 256KB at a time, slow, limited!
    - FTL – flash translation layer, like VM
      - don't overwrite page, write to new and redirect
      - wear leveling! garbage collection!
  - NANTERO? carbon nanotube, DRAM speed, no wear
  - Performance measure metrics
    - latency – time complete
    - response time – time initiate to response
    - throughput/bandwidth – rate
    - startup/overhead – time to initiate
- Filesystems
  - Little's Law – stable system in & out
    - N jobs = λ (rate) · L (latency)
  - utilization $\rho = \frac{\lambda}{\mu_{max}}$ ← maximum rate/bottleneck
    - $\lambda > \mu$ = queue grows w/o bound
    - burstiness = bad mode
  - queuing theory
    - $C = \frac{\sigma^2}{m^2}$ – C=1 ⇒ memoryless, – C=0 ⇒ deterministic
    - M/M/1 memoryless $T_q = \frac{\rho}{1-\rho} \cdot T_s$
    - M/G/1 general $T_q = \frac{1+C}{2} \cdot \frac{\rho}{1-\rho} \cdot T_s$
    - Little's law $L_q = \lambda T_q = \frac{\rho}{T_s} \cdot T_q = \frac{\rho^2}{1-\rho}$
  - Disk Scheduling
    - FIFO – long seek times
    - SSTF – possible starvation
    - SCAN – elevator, closest request in direction of travel
    - C-SCAN – one direction, skips on way back, no middle bias
  - block → files/directories
    - also track free blocks, data blocks, directory
    - expand file too
    - open before read/write (protection checks)
  - 4 – directory, index structure, storage blocks, free space map
  - no read/write raw bytes of directory
  - cwd – allows relative path
  - inode – file info, on disk, one copy in memory
  - FAT (File Allocation Table)
    - linked list of blocks
    - file number = index in table
    - root / at block 2
    - directory has file metadata ?
  - Berkeley FFS (Unix)
    - file number = index into inode array
    - inode/file holds metadata ?
    - direct, indirect pointers
    - con – index all 1 place, how big file? ← of FAT
    - locality – block groups, keep things close (HDD)
    - con – rotational delay
      - fix: interleaving or read ahead
    - con – lots of space for large files
  - Ext2/3 – Linux
    - block groups – locality, also has free maps
    - Ext3 has journaling
  - Hard link – name to file no.
    - inode has reference count to know when delete
  - Symlink – link to file name
  - B-trees for large directories
  - Windows NTFS
    - variable length extents
    - master file table – like FAT and inodes
      - metadata + contents if small
      - 1KB entries
  - mmap file
    - file + sharing
  - buffer cache – for files, in software
    - replacement policy? LRU?
    - read ahead?
    - write back/write – periodic flush
- File system reliability/recovery
  - availability, durability, reliability
  - Reed-Solomon ECC
  - multiple copies
  - RAID 1 – full copy
  - RAID 5+ – striping, XOR
  - RAID 6 – 2 disk failures
  - Random crashing?
    - careful ordering – FAT, FFS
      - operations in order
      - clean up in recovery
    - copy on write (COW)
      - ZFS, WAFL
      - make new file, move over
    - atomic transactions
      - commit changes!
    - Journaling
      - update in log, apply later
      - NTFS, AHFS, XFS, JFS, ext4
      - after commit, complete and apply
      - recovery – discard partial
      - common – usually only metadata update
  - Log FS (LFS)
    - log is the storage
    - all sequential, wrap around
    - buffer cache makes fast
    - segment summary to track if needed
    - free block = check which needed, move to end (replay)
  - F2FS – flash file system
    - for SSDs: random == sequential read; random write bad (sequential is most)
    - node address table (NAT) instead of FTL
- Distributed Systems
  - IP – one network layer protocol
- Distributed Decision Making
  - General's Paradox
    - can't guarantee simultaneity
  - Two-phase Commit (2PC)
    - persistent log (atomic)
    - prepare phase, commit phase
    - all or nothing
    - no time guarantee
    - can blocking
    - alts – 3PC, PAXOS, RAFT
  - Byzantine General Problem
    - malicious nodes
  - Network Protocol
    - Broadcast, P2P
    - 5 layers
      - application
      - Transport – TCP/UDP
      - Network – IP
      - Datalink
      - Physical
    - TCP
      - 4 ack, 2 ptr
      - window size = remaining space
      - never send more than can receive
      - seq no. + length ↑ adaptive alg.
      - 3 way handshake
      - 4 way teardown
  - RPC
  - Virtual FS (VFS)
    - superblock, inode, dentry, file
    - stateless – all info needed in request
  - NFS
    - uses VFS, NFS layer
    - write through
    - stateless, idempotent
  - AFS
    - write through on close
  - key value store
  - consistent hashing
    - deal pos. among machines
    - ring, closest node w/ smaller ID, lower than key

$\lambda = \frac{L_q}{T_q}$

$T_q = \frac{1+C}{2} \cdot \frac{\rho}{1-\rho} \cdot T_s$

$\mu_{max} = 50ps$

$L_q = \lambda T_q = \lambda\left(1.25 \cdot \frac{\lambda/\mu}{1-\lambda/\mu} \cdot T_s\right)$

$L_q = \lambda\left(1.25T_s \cdot \frac{\lambda}{\mu-\lambda}\right)$

$L_q\mu - L_q\lambda = 1.25T_s\lambda^2$

$1.25T_s\lambda^2 + L_q\lambda - L_q\mu = 0$