TLB:
-So CPU doesn't have to walk pages every time
-When reloading cr3, hardware flushes TLB
-If something changes in page table or dir, then we need to invalidate
-TLB miss does not imply page fault, page fault implies TLB miss

Scheduling Disciplines:
-FIFO, Round-Robin, SJF (STCF), Priority Schemes, Lottery, multilevel feedback queues, real time.
-Scheduling decisions shouldn't affect results

System/Engineering lessons:
-know your goals, write them down
-compare against optimal
-There are many schedulers running simultaneously

Virtual Memory:
-Translation box gives a few things
--Protection, processes can't touch other's memory
--Relocation, two instances of a program are loaded, each think they are using the same address, when they really aren't
--sharing, processes share memory under controlled circumstances.
-In x86, there is actually a two-level translation
--Segmentation translation
---Set to identity map for rest of class
---It can not be turned off
---Not easy to grow or shrink segments
---Memory addresses treated like offsets into continuous region (linear ad = base + virt ad)
---Every instruction takes explicit or implicit register
--Page translation
---Paging is more flexible
---On 32-bit system 2^20 pages
---Page is 4096 bytes normally
---CR3 is the address to the PGDir
---logical address is broken into 10 | 10 | 12 bits for pgdir | pgtbl | offset, where each is an offset into a table, and the entry in pgdir is a pointer to the pgtbl, while the pgtbl houses the page number.
---Threads use the same CR3, processes do not. A context switch is implemented by changing CR3
---If JOS wanted to, it could map anything to anywhere.
--- Code below describes what MMU Does

```
uint
translate (uint la, bool user, bool write){
  uint pde;
  pde = read_mem (%CR3 + 4*(la >> 22));
  access (pde, user, write);
  pte = read_mem ( (pde & 0xfffff000) + 4*((la >>
12) & 0x3ff));
  access (pte, user, write);
  return (pte & 0xfffff000) + (la & 0xfff);}
// check protection. pxe is a pte or pde.
// user is true if CPL==3.
// write is true if the attempted access was a write.
// PG_P, PG_U, PG_W refer to the bits in the entry
above
void
access (uint pxe, bool user, bool write){
  if (!(pxe & PG_P)
          => page fault -- page not present
  if (!(pxe & PG_U) && user)
          => page fault -- not access for user
  if (write && !(pxe & PG_W)) {
    if (user)
           => page fault -- not writable
    if (%CR0 & CR0_WP)
           => page fault -- not writable
  }}
```

---Segments are used to switch priviledge levels into and out of kernel
---physical memory is mapped at top because kernel needs to be able to use physical memory a few times
---R/W user read and write bit, 0 no write

UVPT:
-We want a contiguous set of virtual addresses, we insert a pointer in the pgdir back to the pgdir itself, so the page tabes themselves show up in the virtual address space.

Page Faults:
-Is used for:
--Virtual memory, system thinks it has 512 MB when really it has 4MB, swap space.
--Store memory pages across network
--Copy-on-write, copy another process but don't write anything until something changes
--Accounting, figure out what is using it.
-Demand paging, when going to disk to grab page when needed. If access a page fault 1 out of 1000000 the system will slow down 10 percent.
--Thrashing even more expensive, up to 20000x

Page structures:
-Large pages sizes – lots of wasted mem
-Small page sizes – lots of pte's may not be used
-Many level page table – not much mem spent on page tables, but lots of page walking

Replacement Policy:
-Policies
--FIFO
--Optimal-throw away one that won't be used for the longest
--Least Recently used – Use queue to track recently accessed and hashmap to lookup, not good for OS
--Clock Algorithm/Nth chance – hand sweeps n times, if hand points to page with clear bit, it evicts. Each time hand passes it, it decrements the counter.
--Modify this for efficent, dirty pages cost more to evict, give them priority to not swap out.
-Conclusion
--Optimal is known as min
--LRU is good approx. for min
--Implementing LRU in hardware is a pain
--Sometimes caching doesn't save the day
-Misc replacement stuff
--x86 has us/set, modified, valid, and read only bits. can get away without modify/dirty bit by marking all pages read only. Also, get away without use/accessed bit by marking all pages not present.
-Caching doesn't work:
--If processes don't use them
--Everything doesn't fit
--all processes fit, but too much for system to handle
--Can't do anything about first 2, but can for 3rd:
---working set, union of working set fits in mem
---page fault freq, track and measure and swap.
Heap Memory Management:
-Dynamic mem allocation:
--automatic allocation, garbage collection
--Malloc, free
-Fragmentation requires 2 things:
--different sizes requests
--different lifetimes for requests
-Allocator decisions – try to reduce frag
--Placement, try to reduce frag
--Split free blocks to satisfy small requests
--Coalesce free blocks, for large requests
--Every allocation pattern has bad guy causing bad.
--Best/first fit, overall perform the same
--App request pattern, app handles memory, frees chunk at the end of use
--sbrk, a way to ask OS to expand size of heap, bad for large allocation because can't get page back
---use mmmap with MAP_ANON mode intead
I/O:
-Communicate with devices:
--Memory-mapped registers- read/write to special reg
--Device memory – write to I/O device memory
--Special I/O instruct – like load but works on I/O
--DMA – intermediate device to handle I/O
---request transfer of c bytes to address x
---tells disk controller to do so, disk controller starts
---DMA controller transfers bytes, then notifies OS
-Device Drivers – expose API, handle synchro.
-Disk
--Generally only one active head at a time, cross talk
--Seek consists of 4 phases:
---Speedup-accel arm to max speed
---Coast – at max speed (Long seeks)
---Slowdowns-stop near destination
---Adjust head to actual desired track
--Time TT = rot delay, seek, tx time
--Disk scheduling:
---FIFO, SSTF, Elevator, SEEK
Locks:
-Lock performance:
--Dance hall arch, all cpus have equal slow access
--NUMA, each cpu has close fast ram
--cache coherent-protocal to invalidate caches when local change happens
-Reader-writer problem – shared locks
-MCS lock – lock is just a pointer to qnode
typedef struct qnode {
struct qnode *next;
bool locked;} qnode;
--the lock is just a list of cpu's waiting for the lock
Dahlin's commandments:
1. Always do things the same way
2. Always use monitors
3. Always hold lock when operating on cv
4. Always grab lock at begin and release at end
5. Always use while(pred){Condition->wait()}
6. Almost never sleep()
Misc:
Transactions – when deadlock is detected, abort
Non-blocking synch – wait, lock free alg. Lots of Mem used up, but they are provably lock free.