Virtual Memory

- MMU: determines physical memory addresses from virtual
- virtual addresses are outside the actual physical addresses, pretend we have more memory than we actual do
- overlay: manual unload and load programs
- running two programs simultaneously
    - multiple virtual address spaces
- three main ideas:
    - pretend we have more memory than we do
    - map “virtual” addresses to “physical” ones
    - hide complexity from programs
        - programs should think they’re “isolated”
- CPU — [virtual addresses]—> MMU — [physical addresses] —> Memory
- page table entries:
    - v = 0x21ff
    - break into two parts
        - number of bit for virtual address space: 2 (page number), #2 elements in array
        - offset: 1ff
            - number of bit that are the size of the page (offset into the page)
    - using page number in page tables and then construct a new address
    - 5 - 1ff, p = 0x51ff
    - control bits in a page table entry

What does the OS do?

- each processes is going to have its own set of page tables
    - processes can grow as much as they want
    - don’t interfere with each other

Context switch:

1. push registers
2. save stack pointer
3. pick next process
4. restore stack pointer
5. pop registers
6. return

Context switch now

1. push registers
2. save stack pointer
3. pick next process
4. switch page tables
5. restore stack pointer
6. pop registers
7. return

How else does the OS use virtual memory?

- which pages in physical memory correspond to which files
- use page cache
    - what pieces of data that had be loaded into physical memory from disk

loading without page cache

1. cope .text segment into free memory
2. copy.data segment into free memory
3. Link
4. execute program

Loading with page cache

1. map virtual address 0-0x1000 to P's .text
2. Map virtual address 0x100-0x2000 to P's .data
3. execute program

Modern OS Memory Management:

- kernel keeps track of files and their pages
- processes have a set of maps they've made
    - but the page tables don't reflect those maps yet
- When a process tries to access a piece of memory that its mapped… but the mapping isn't in the page table
    - a page-fault occurs

Multi-level page tables:

- top bits: page directory to locate page table
- go to page table given and then use second top most bits to index into page table to get physical address

What does a memory access cost? Page walk

- add a cache! TLB
- Translation Lookaside Buffer
    - saving translation of virtual to physical in a cache, TLB

TLB invalidation

- when mapping update page tables and invalidate TLB

When we update our cache table entry we need to update our TLB
interprocessor interrupt: invalidate other core's TLB: have to wait for a response and is very time consuming

circular buffer ….?