Lecture Topics: 11/17

- Virtual vs. Physical addresses
- Whole process-swapping
- Paging

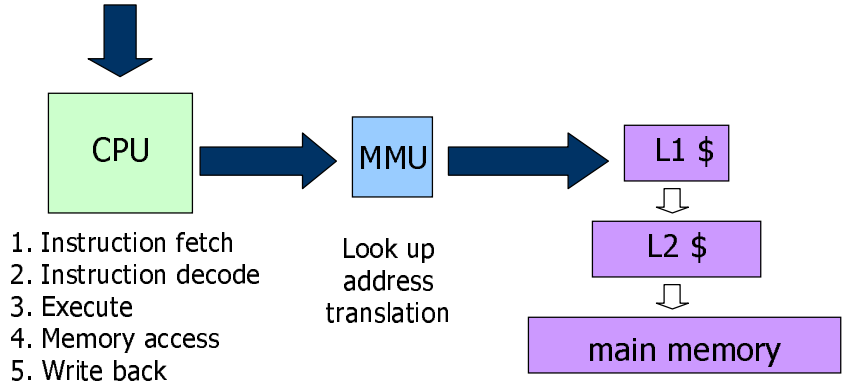
Virtual Addressing

- There are two kinds of memory addresses in modern systems
  - *Physical* addresses are the ones the memory system (the L1 and L2 cache and main memory) deals with
  - *Virtual* addresses are the ones the CPU deals with
- When the CPU executes a load or store instruction, the virtual address is *translated* to a physical address before it is passed to the memory system

Accessing Memory

Your program:

```
lw $t0, 0($t1)
```



Why Virtual Addressing?

- Allows different processes to use the same addresses without conflicts
  - All process' text segments can begin at 0x00400000
- Helps us achieve memory protection
  - Since all addresses are translated with respect to the process referencing them, process A can't access process B's memory

Swapping

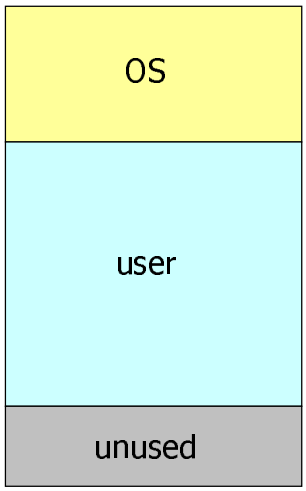
- A process currently loaded into physical memory is called *resident*
- If the OS copies the contents of the process' memory onto disk, the process is "frozen" -- this is called *swapping out* the process
- The process can be copied back in later and resumed
- What does this remind you of?

Physical Memory Layout

- Contiguous Allocation:
  - Each process gets an unbroken range of addresses
  - Single-partition allocation: one process resident at a time
  - Multiple-partition allocation: multiple processes resident at a time
- Noncontiguous allocation:
  - Paging, segmentation, or a combination

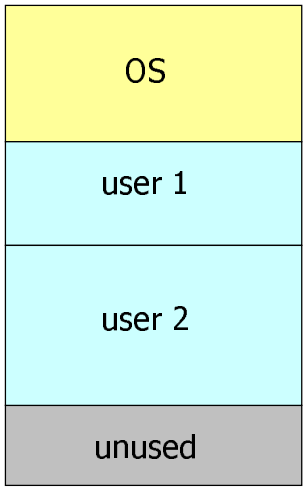
Single Partition Allocation

- Set aside some memory for the OS
- User program gets the rest
- Use MMU to translate addresses; simple addition

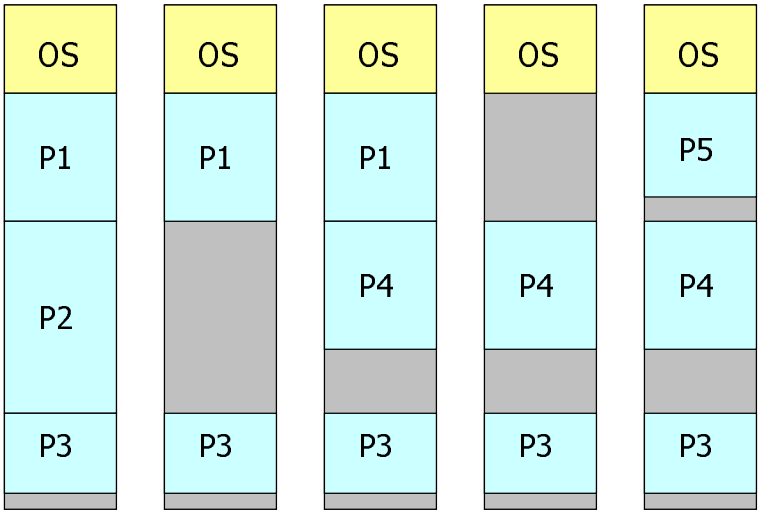


Multiple Partition Allocation

- OS still gets its own partition
- Place multiple processes in the rest of memory
- Translation is still simple, but now process-dependent



Dynamic Storage Allocation



Dynamic Storage Allocation

- Over time, unused memory will be spread out in small pieces rather than collected in one contiguous piece
- This is called *external fragmentation*
  - the holes are external to each process
- Unused memory *within* a process' allocated region is known as *internal fragmentation*
  - more common in paging schemes

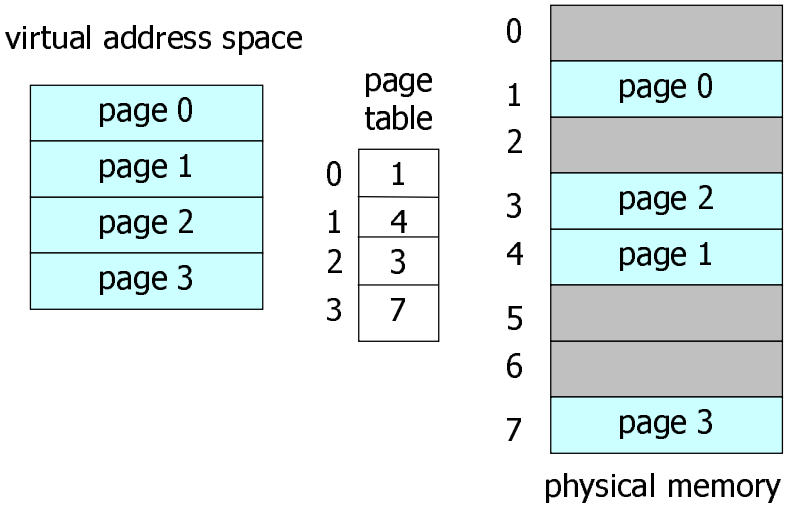
Compaction

- If fragmentation is such a problem, why not move processes around to coalesce empty spaces?
- Can be time-consuming to copy large regions of memory
- If the process is short-lived, not worth moving it
- Can be combined with swapping

Paging

- Split memory into lots of very small regions called *frames*
- Split process address spaces into lots of very small regions called *pages*
- When mapping the process into physical memory, don't insist on contiguous frames
  - any frames will do, even out of order

Pages and Frames



Address Translation for Paging

- Every virtual address can be split into two parts:
  - The VPN, or *virtual page number*
  - The page offset (how far into the page the address is)
- The VPN must be translated by the MMU into a physical frame number
- The offset stays the same

Translation Example

- Suppose you have a 32-bit address space
- Each page contains 1024 = 2^{10} bytes
- How many bits in the VPN?
- How many pages in a virtual address space?
- How many page frames in the physical address space?