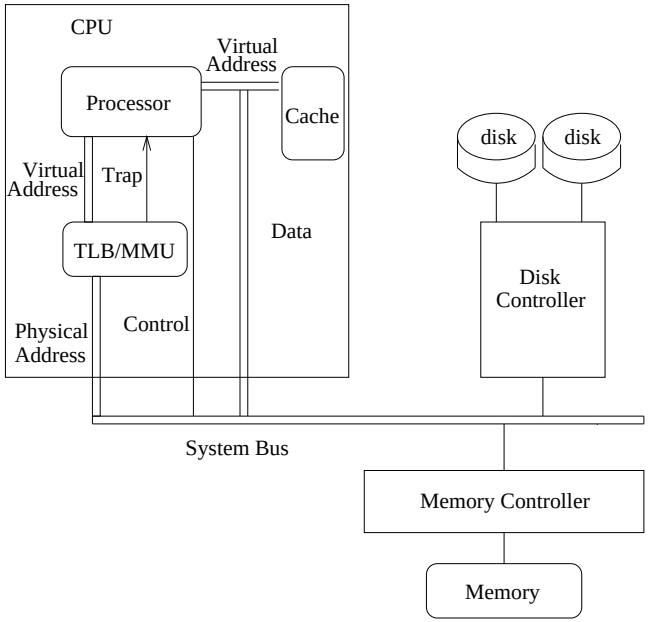
Where we are in the course

- Discussed:
  - Processes & Threads
  - CPU Scheduling
  - Synchronization & Deadlock
- Next:
  - Memory Management
- Remaining:
  - File Systems and Permanent Storage
  - Distributed Systems

Memory Today

- Where is the executing process?
- How do we allow multiple processes to use main memory simultaneously?
- What is an address and how is one interpreted?

Back to Architecture

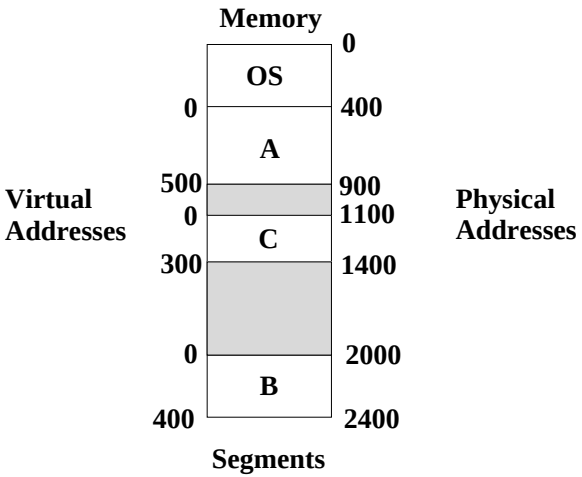


TLB: Translation Look aside Buffer

MMU: Memory Management Unit

- Program executable starts out on disk.
- To execute, the OS loads the process into memory.
- Data and instructions migrate into the cache.

$\Longrightarrow$ OS focus: memory



Segment: A chunk of memory assigned to a process.

Physical Address: a real address in memory (from 0 to the size of the memory).

Virtual Address: an address relative to the start of a process's address space.

Where do addresses come from?

How do programs generate instruction and data addresses?

Compile time: The compiler generates the exact physical location in memory starting from some fixed starting position k. The OS does nothing.

Advantages:

Disadvantages:

Load time: Compiler generates an address, but at load time the OS determines the process' starting position. Once the process loads, it does not move in memory.

Advantages:

Disadvantages:

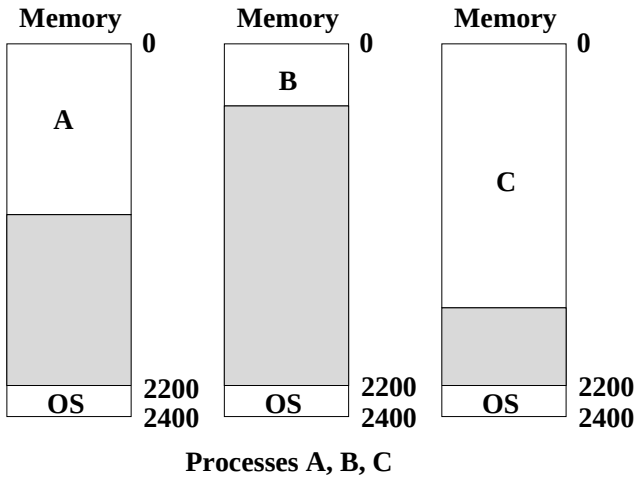
Execution time: Compiler generates an address, and OS can place it any where it wants in memory.

Advantages:

Disadvantages:

Uniprogramming

- OS gets a fixed part of memory (highest memory in DOS).
- One process executes at a time.
- Process is always loaded starting at address 0.
- Process executes in a contiguous section of memory.
- Compiler can generate physical addresses.
- Maximum address = Memory Size - OS Size
- OS is protected from process by checking addresses used by process.



$\Longrightarrow$ Simple, but does not allow for overlap of I/O and computation.

Multiple Programs Share Memory

Transparency:

- We want multiple processes to coexist in memory.
- No process should be aware that memory is shared.
- Processes should not care what physical portion of memory they are assigned to.

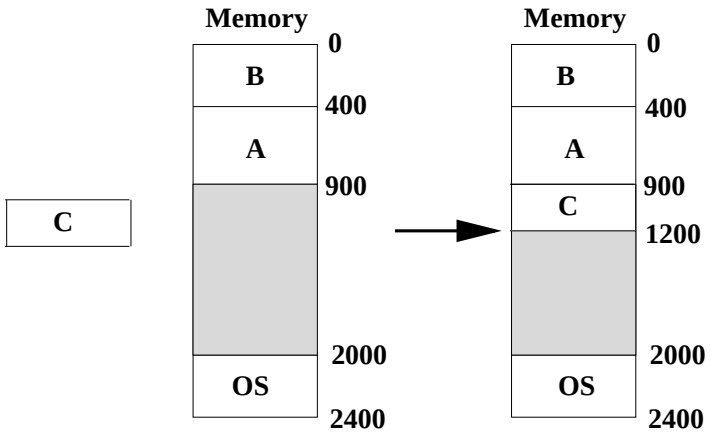
Safety:

- Processes must not be able to corrupt each other.
- Processes must not be able to corrupt the OS.

Efficiency:

- Performance of CPU and memory should not be degraded badly due to sharing.

Relocation



- Put the OS in the highest memory.
- Assume at compile/link time that the process starts at 0 with a Maximum address = Memory Size - OS Size.
- When the OS loads the process, it allocates a contiguous segment of memory in which the process fits. If it does not fit, the OS waits until another process terminates.
- The first (smallest) physical address of the process is the *base* address and the largest physical address the process can access is the *limit* address.
- The base address is also known as the relocation address. The OS stores the base address into the relocation register, and then when the process generates a logical address, the hardware adds the relocation address to it.

Relocation

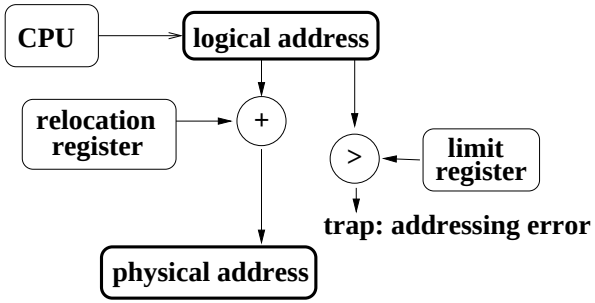
Static Relocation:

- at load time, the OS adjusts the addresses in a process to reflect its position in memory. (Similar to the linker determining where procedure calls go with separate compilation.)
- Once a process is assigned a place in memory and starts executing it, the OS cannot move it. Why?

Dynamic Relocation: In parallel,

1. hardware adds relocation register (base) to virtual address to get a physical address;
2. hardware compares address with limit register (address must be less than base).

If test fails, the processor takes an address trap and ignores the physical address.



Relocation

Advantages:

- OS can easily move a process during execution.
- OS can allow a process to grow over time.
- Simple, fast hardware: two special registers, an add, and a compare.

Disadvantages:

- Slows down hardware due to the add on every memory reference. (In the 60's this was a show stopper!)
- Can't share memory (such as program text) between processes.
- Process is still limited to physical memory size.
- Degree of multiprogramming is very limited since all the memory of all active processes must fit in memory.
- Complicates *memory management*.

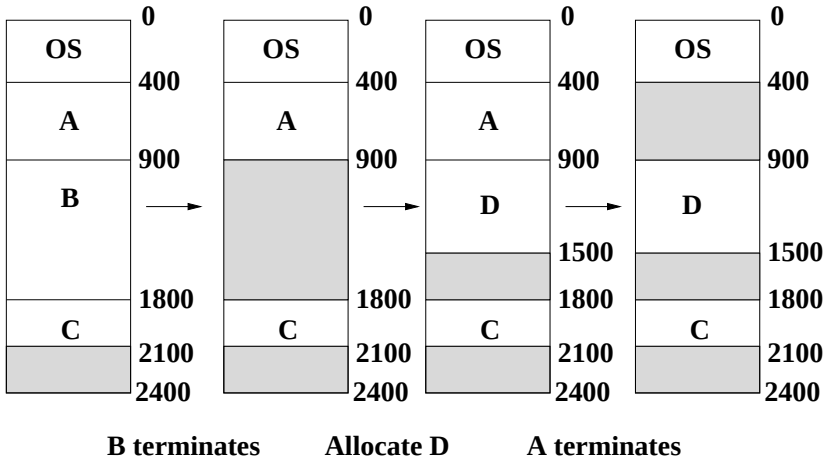
Transparency: processes are largely unaware of sharing the memory space.

Safety: each memory reference is checked.

Efficiency: memory checks are done in hardware so they are fast, translation from virtual addresses to physical addresses is also done in hardware, BUT if a process grows, it may have to be moved which is very slow.

Memory Management: Memory Allocation

As processes enter the system, grow, and terminate, the OS must keep track of which memory is available and utilized.



Holes: pieces of free memory (shaded above in figure)

Given a memory request from a starting process: the OS must decide which hole to use for the process.

Memory Allocation Policies

First-Fit: allocate the first one in the list in which the process fits. The search can start with the first hole, or where the previous first-fit search ended.

Best-Fit: Allocate the smallest hole that is big enough to hold the process. The OS must search the entire list or store the list sorted by size hole list.

Worst-Fit: Allocate the largest hole to the process. Again the OS must search the entire list or keep the list sorted.

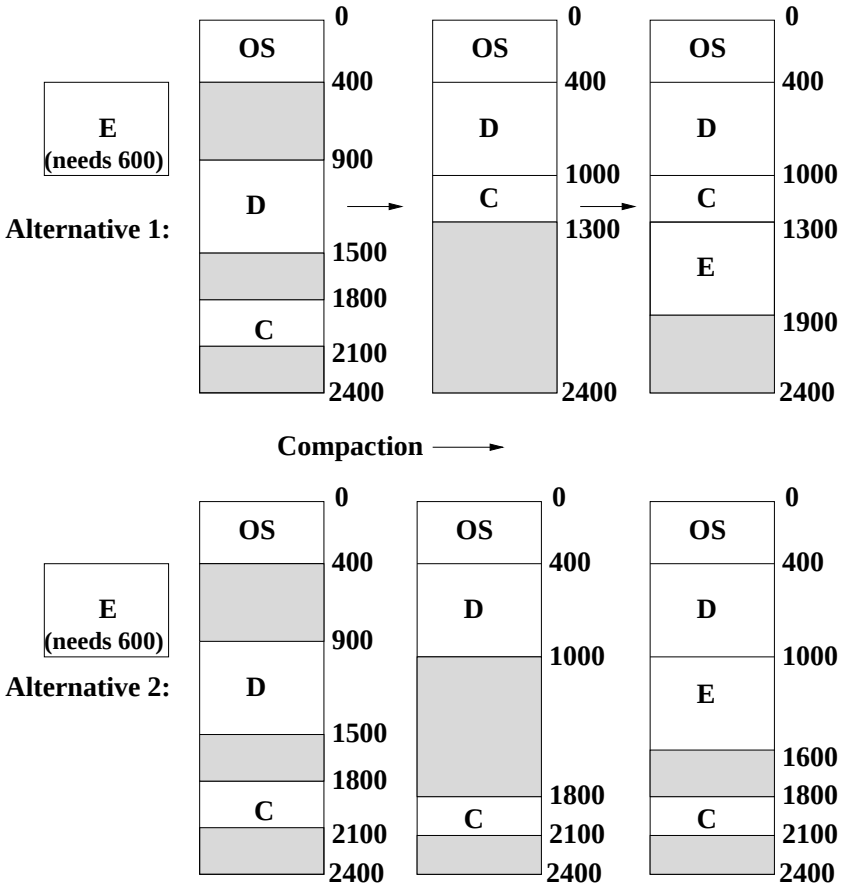
$\Longrightarrow$ What might you expect the distribution of the size of the holes to be over time?

$\Longrightarrow$ Simulations show first-fit and best-fit usually yield better storage utilization than worst-fit; first-fit is generally faster than best-fit.

External Fragmentation - holes between processes.

- There may be enough memory to fit a process in memory, but because it is fragmented, the process must wait.
- Simulations with first-fit show up to 1/3 of memory may be lost to fragmentation, i.e., the pieces of free memory are too small to fit a process.
- We want an allocation policy that minimizes wasted space.

Compaction



- How much memory is moved?
- How big a block is created?
- Any other choices?

Internal Fragmentation

Given a process of size 8846 bytes and a block of size 8848 bytes,

$\Longrightarrow$ it is more efficient to allocate the process the entire 8848 block than it is to keep track of 2 free bytes.

Swapping

- Roll out a process to disk, releasing all the memory it holds.
- When process becomes active again, the OS must reload it in memory.
  - With static relocation, the process must be put in the same position.
  - With dynamic relocation, the OS finds a new position in memory for the process and updates the relocation and limit registers.
- If swapping is part of the system, compaction is easy to add.
- How could or should swapping interact with CPU scheduling?

Summary

- Processes must reside in memory in order to execute.
- Processes generally use virtual addresses which are translated into physical addresses just before accessing memory.
- Segmentation allows multiple processes to share main memory, but makes it expensive for processes to grow over time.
- Swapping allows the total memory being used by all processes to exceed the amount of physical memory available, but increases context switch time.
- Your C and C++ program memory managers use the same principals as segmentation and paging to attain fast allocation and freeing time, with a minimum of wasted space (fragmentation).
- Modern memory systems use paging (next time)