Computing Parable

• The Donkey that starved

Where we are in the course

• Discussed:
  – Processes & Threads
  – CPU Scheduling
  – Synchronization & Deadlock

• Next up:
  – Memory Management

• Yet to come:
  – File Systems and I/O Storage
  – Distributed Systems
Memory Management

- 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?

Background: Computer Architecture

- Program executable starts out on disk
- The OS loads the program into memory
- CPU fetches instructions and data from memory while executing the program
Memory Management: Terminology

- **Segment**: A chunk of memory assigned to a process.
- **Physical Address**: a real address in 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.

- **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.

- **Execution time**: Compiler generates an address, and OS can place it anywhere it wants in memory.
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.

⇒ 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.
- Load a process by allocating a contiguous segment of memory in which the process fits.
- 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.
Relocation

- **Static Relocation:**
  - at load time, the OS adjusts the addresses in a process to reflect its position in memory.
  - Once a process is assigned a place in memory and starts executing it, the OS cannot move it. (Why?)

- **Dynamic Relocation:**
  - hardware adds relocation register (base) to virtual address to get a physical address;
  - hardware compares address with limit register (address must be less than limit).
  - If test fails, the processor takes an address trap and ignores the physical address.

Dynamic 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.
  - 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 memory of all active processes must fit in memory.
  - Complicates memory management.
Relocation: Properties

- **Transparency**: processes are largely unaware of sharing.

- **Safety**: each memory reference is checked.

- **Efficiency**: memory checks and virtual to physical address translation are fast as they are 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 new 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.

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

Fragmentation

- **External Fragmentation**
  - Frequent loading and unloading programs causes free space to be broken into little pieces
  - External fragmentation exists when there is enough memory to fit a process in memory, but the space is not contiguous
  - *50-percent rule*: Simulations show that for every $2N$ allocated blocks, $N$ blocks are lost due to fragmentation (i.e., $1/3$ of memory space is wasted)
  - We want an allocation policy that minimizes wasted space.

- **Internal Fragmentation**:
  - Consider a process of size 8846 bytes and a block of size 8848 bytes
    - it is more efficient to allocate the process the entire 8848 block than it is to keep track of 2 free bytes
  - Internal fragmentation exists when memory internal to a partition that is wasted
Compaction

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

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?
Paging: Motivation & Features

90/10 rule: Processes spend 90% of their time accessing 10% of their space in memory.

=> Keep only those parts of a process in memory that are actually being used

• Pages greatly simplify the hole fitting problem
• The logical memory of the process is contiguous, but pages need not be allocated contiguously in memory.
• By dividing memory into fixed size pages, we can eliminate external fragmentation.
• Paging does not eliminate internal fragmentation (1/2 page per process)

Paging: Example

Mapping pages in logical mem to frames in physical memory
Paging Hardware

• **Problem:** How do we find addresses when pages are not allocated contiguously in memory?

• **Virtual Address:**
  – Processes use a virtual (logical) address to name memory locations.
  – Process generates contiguous, virtual addresses from 0 to size of the process.
  – The OS lays the process down on pages and the paging hardware translates virtual addresses to actual physical addresses in memory.
  – In paging, the virtual address identifies the page and the page offset.
  – *page table* keeps track of the page frame in memory in which the page is located.