CS-202 Week 3

Notes

Lecture 1

C recap: Lame

Stack (= static allocation)

If:

- size is known *compile-time*
- size is small (a few Mb)
- size is const

Heap (= dynamic allocation)

If not all 3 conditions are fulfilled

Good practices

- Each malloc()/calloc() should have its free()
- prefer calloc() to malloc()
- add ptr = NULL; after free(ptr);

Lecture 2

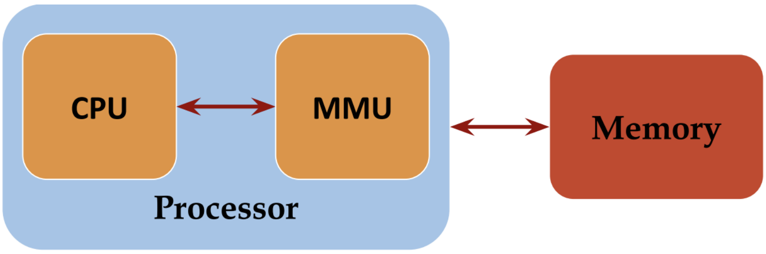
Long running background tasks -> CPU bound, batch processing
Interactive tasks -> I/O bound

Memory (intro)

Address space is a set of memory addresses that are only accessible to a program.
Stack and heap are dynamic
Text and data are static

Memory management unit (MMU)

Manages the mapping from virtual memory to physical memory address



The principle of indirection

We can solve and problem by introducing an extra level of indirection

Indirection: Practice of using an intermediate layer or mechanism to access / Manipulate data or resources instead of directly interacting with them.

The Address Space Abstraction

A memory is a ressource that is accessible at the granularity of a byte (**NOT** a bit)

Call stack (Stack)

Temporary data such as function parameters, local variables and return adresses

Heap

Dynamic memory allocation during program runtime. Grows from low to high adresses

Data

Allocates global variables and data-structures

Text

Code & constants

Importance of dynamic memory

Required memory varies dependent on running program

Dynamic data structure:

Stack

- Follows the policy of first in last out (FILO)
- Elements that enter first leave last
- Operations:
    - Push: Add's an element at the top of the stack
    - Pop: Remove element from top of stack
- Returned data in inverse order of insertion
- Each CPU provides a stack segment on a per-computation unit

Invocation order of functions on stack

- parameters (right to left),
- Return IP (RIP), function address in data
- local function variables

Heap:

API:

- **alloc:** Creates an object

- **free:** Indicates object is no longer used

Basic idea

Available heap space can represented as free list

alloc: take a free block, split, put the rest back into free list
free: add block to free list

Better implementation:

- **Alloc**: Find a fitting obj first
  - First fit: find the first object in the list and split it
  - Best fit: find the object that is closest to size
  - Worst fit: find biggest object and split it
- **Free:** If adjacent region is free, merge two blocks

Heap and OS interaction

OS gives processes a large memory region to store heap objects

The case for virtual memory

Virtualization enables isolation, isolation requires separation. A process must be prohibited from accessing other processes registers/memory.

Goals: (slide 28)

Transparency

Multiple programs coexist in memory without knowing about eachother

Protection

OS / other process should not corrupt each other

Efficiency

Sharing

Processes may share part of the address space

- Address space starts at 0x0
- Map virtual addresses to physical addresses

Virtualizing physical memory: Providing the illusion of private address space

A Simple MMU: base register

Idea: Translate every virtual address to physical address by adding an offset
Store offset in a special register (controlled by the OS, used by the MMU)
Each process has a different offset in their base register

Problem: Does not prevent processes from accessing higher addresses

Doesn't Work

A Simple MMU: base and bounds

Keep two values (in registers) for every process: **base** and **bounds**

- Base register sets the minimum address
- Bounds registers sets (virtual) limit of the address space, highest physical address that is accessible becomes base + bound

Code:

```
if (addr < bounds)
   return *(base + addr)
else
   throw new SegFaultException();
```

Pros:

- Achieves security & performance (isolated processes, addition and check are cheap)
  Cons:
- No memory sharing
- Waste of physical memory (all memory must be pre-allocated)
  - Results in memory fragmentation

Fragmentation and Segmentation

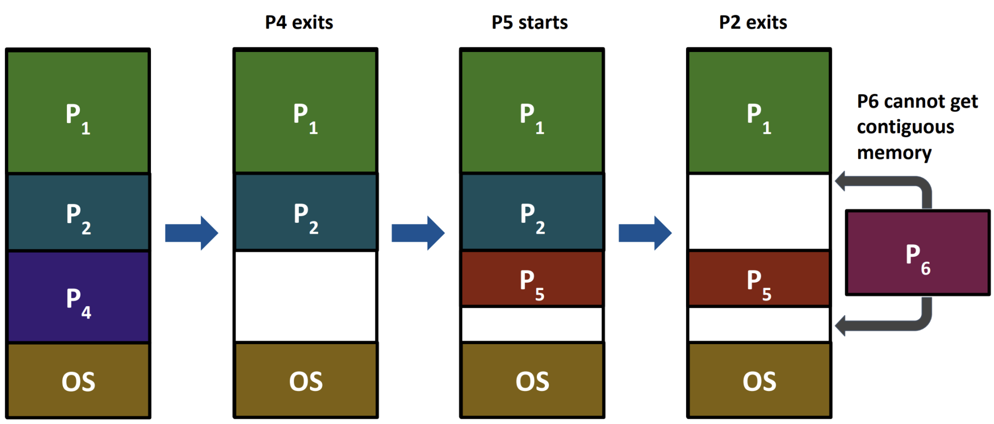
Fragmentation reduces performance

Fragmentation is the problem, Segmentation is the Solution

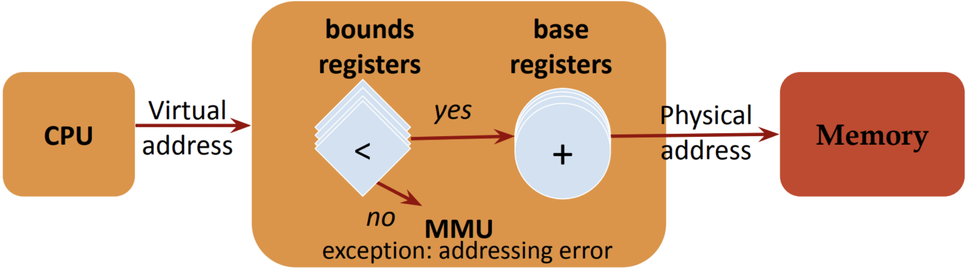
Fragmentation

Fragmentation is a phenomenon in which storage space is used inefficiently reducing capacity and often performance

External fragmentation:



A MMU: segmentation



One base and bound register per memory area

- Code segment: CS register
- Data segment: DS register
- Stack segment: SS register
- User-defined extra segments: ES/FS/GS registers

OS can place segments **independently anywhere** in physical memory

- Unlike base + bound in which process memory should be contiguous