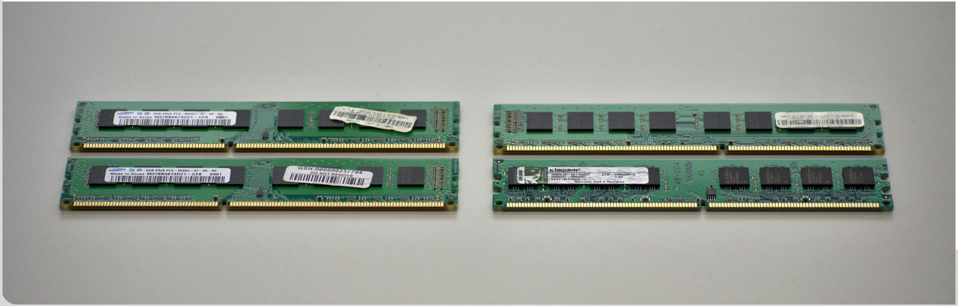
Operating Systems

05. Segmentation

Prof. Dr. Frank Bellosa | WT 2020/2021

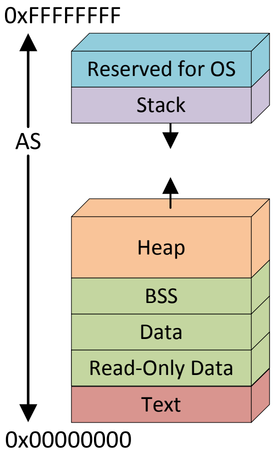
KARLSRUHE INSTITUTE OF TECHNOLOGY (KIT) – ITEC – OPERATING SYSTEMS



Process Address Space Layout

OS	Addresses of the kernel
Stack	Local variables, function call parameters, return addresses
Heap	Dynamically allocated data
BSS	Uninitialized data
Data	Initialized data
RO-Data	Read-only data
Text	Executable code

How does the OS organize and provision memory for multiple processes?

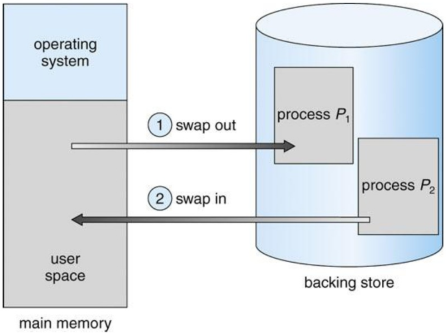


Main Memory

- Main memory and registers are the only storage that the CPU can access directly
- Program must be brought into memory from background storage and placed within a process' address space for it to be run
- Early computers had no memory abstraction
  - Programs accessed physical memory directly
- Multiple processes can be run concurrently even without memory abstraction
  - Swapping
  - Static Relocation

Swapping

- Swapping denotes
  - saving a program's state on background storage (roll-out)
  - replacing it with another program's state (roll-in)[SGG12]



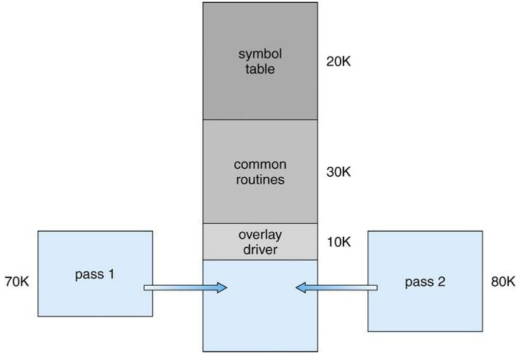
\+ Only needs hardware support to protect the kernel, but not to protect processes from one another

– Very slow
  - Major part of swap time is transfer time
  - Total transfer time is directly proportional to the amount of memory swapped

– At every point in time only one process runs: no parallelism
  - This process owns the entire physical user space

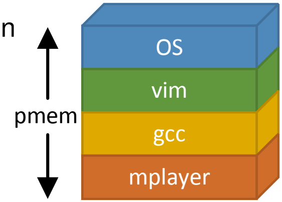
Overlays

- What if the process you want to run needs more memory than available?
- – Need to partition program manually



Static Relocation

- Another possibility to solve address conflicts when loading multiple processes is static relocation
  - The OS adds a fixed offset to every address in a program when loading it and creating a process from it



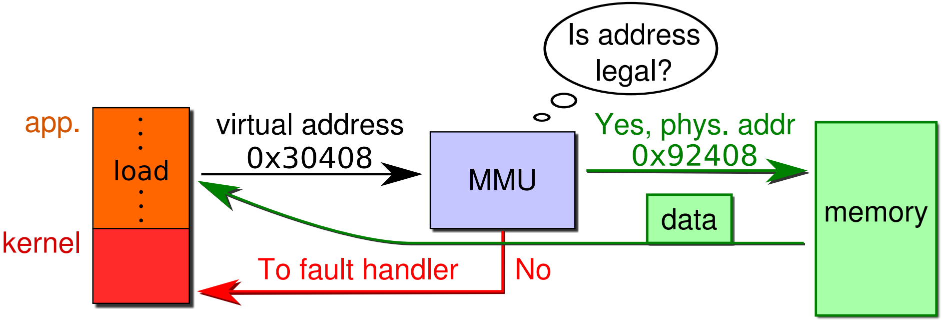
- Same address space (physical addresses) for every process
  - No protection: Every program sees and can access every address.
  - What if gcc needs more memory for its abstract syntax tree?
  - What if mplayer is pausing playback and currently doesn't need memory? Can it be reused by other processes?
  - What if no contiguous free region fits program?

Desired properties when sharing physical memory

- Protection
  - A bug in one process must not corrupt memory in another
  - Don’t allow processes to peek into other processes’ memory
- Transparency
  - A process shouldn’t require particular physical memory addresses
  - Processes should be able to use large amounts of contiguous memory
- Resource exhaustion
  - Allow that the sum of sizes of all processes is greater than physical memory

Ideal Memory-Management Unit (MMU)

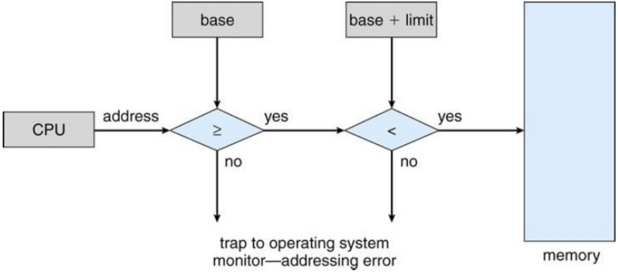
- Architectural support to achieve safe and secure protection
- Hardware device maps virtual to physical address
- The user program deals with virtual addresses
  - It never sees the real physical addresses



[Maz20]

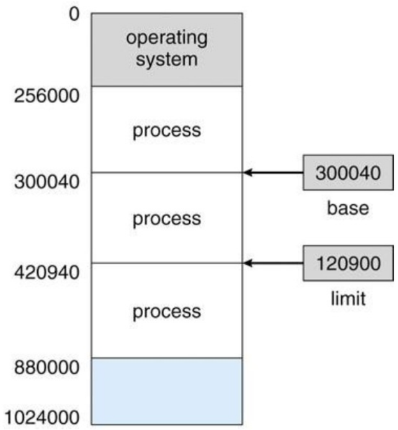
Base and Limit Registers

- Provides protection with static relocation
  - Introduce special base and limit registers
- On every load/store the MMU
  - Checks if the address is larger or equal to **`base`**
  - Checks if the address is smaller than **`base`** + **`limit`**
  - Use the CPU address as the physical address in memory [SGG12]



Protecting the Kernel with Base and Limit Registers

- Need to protect OS from processes
- Main memory split into two partitions
  - Resident operating system, usually held in low memory
  - User processes held in high memory
- OS can access all process partitions e.g., to copy syscall parameters
- MMU denies processes access to OS memory [SGG12]



Base and Limit Registers

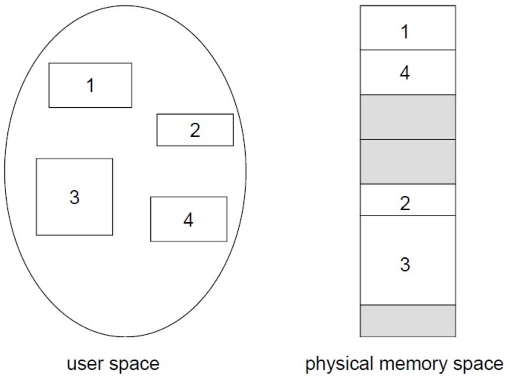
+ Straight forward to implement MMU
  - Only need to load new base and limit registers to switch address space
+ Very quick at run-time
  - Only two comparisons (can do both in parallel)
  - Compute **`base`** + **`limit`** in advance

– How do you grow a process' address space?

– How do you share code or data?

Segmentation

- Solution for shortcomings of Base + Limit approach:
  - Use **multiple** Base+Limit register pairs **per process**

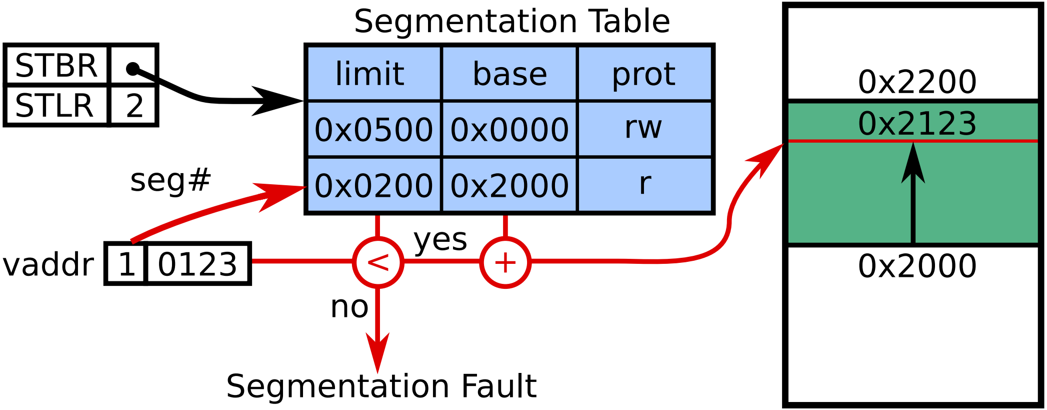


→ Can keep some segments private, share others

Segmentation Architecture

- Virtual address consists of a tuple: $<$segment #, offset$>$
  - Can be encoded in the address (PDP-10 – seg #: high bits, offset: low bits)
  - Can be selected by an instruction or an operand
- Each process (address space) has a segment table that maps virtual address to physical addresses in memory
  - Base Starting physical address where the segment resides in memory
  - Limit Length of the segment
  - Protection Access restriction (read/write) to make safe sharing possible
- The MMU has two registers that identify the current address space
  - Segment-table base register (STBR) points to the segment table location of the current process
  - Segment-table length register (STLR) indicates number of segments used by the process (segment # is legal if it is $<$ STLR)

Segmentation Mechanics



[Maz20]

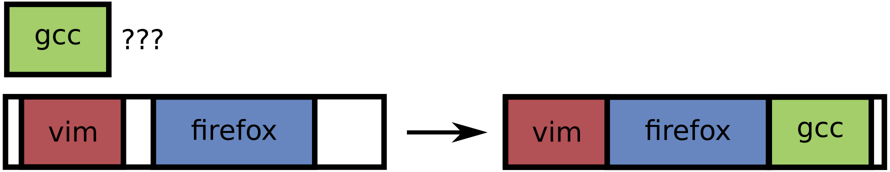
Segmentation Trade-offs

+ Dynamic relocation is possible
  (adaption of base in segment-table)
+ Makes data/code sharing between processes possible without compromising confidentiality and safety/security
+ Process doesn't need one large contiguous physical memory area
  → easier placement
+ Don't need entire process in memory
  → Can overcommit memory with segement swap after a segmentation fault

– Segments need to be kept contiguous in physical memory
– Fragmentation of physical memory

External Fragmentation

- Fragmentation $\equiv$ The inability to use free memory
- External Fragmentation Sum of free memory satisfies requested amount of memory. Request cannot be fulfilled with contiguous memory.
- Compaction reduces external fragmentation
  - Close gaps by moving allocated memory in one direction
  - Results in a large free block on the other side
  - Compaction is possible only if relocation is dynamic, and can be done at execution time.



– Expensive: Need to halt process while moving data and updating tables.

References I

[Maz20] David Mazières. Cs140 – operating systems. *Stanford University*, 2020.

[SGG12] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. Wiley Publishing, 9th edition, 2012.