Chapter VII
Memory Management

Jehan-François Pâris
jfparis@uh.edu
Chapter Overview

- A review of classical approaches to memory management
  - Follows the evolution of operating systems from the fifties to the eighties
Solution 0

- No memory management
- The very first computers had no operating system whatsoever
- Each programmer
  - Had access to whole main memory of the computer
  - Had to enter the bootstrapping routine loading his or her program into main memory.
Solution 0

- **Advantage:**
  - Programmer is in total control of the whole machine.

- **Disadvantage:**
  - Much time is lost entering manually the bootstrapping routine.
Solution 1

- **Uniprogramming**

- Every system includes a *memory-resident monitor*
  - Invoked every time a user program would terminate
  - Would immediately fetch the next program in the queue (*batch processing*)
Solution 1

- Should prevent user program from corrupting the kernel
- Must add a Memory Management Unit (MMU)
Solution 1

- Assuming that the monitor occupies memory locations 0 to START – 1
- MMU will prevent the program from accessing memory locations 0 to START – 1
MMU for solution 1

RAM Address

≥ START

YES

NO

trap
Solution 1

- **Advantage:**
  - No time is lost re-entering manually the bootstrapping routine

- **Disadvantage:**
  - CPU remains idle every time the user program does an I/O.
Solution 2

- Multiprogramming with fixed partitions
  - Requires I/O controllers and interrupts

- OS dedicates multiple partitions for user processes
  - Partition boundaries are fixed

- Each process must be confined between its first and last address
Solution 2

- Computer often had
  - A foreground partition (FG)
  - Several background partitions (BG0, . . .)
MMU for solution 2

- RAM Address
  - ≥ FIRST
    - NO: trap
    - YES
      - ≥ LAST
        - NO: trap
        - YES
Solution 2

- **Advantage:**
  - No CPU time is lost while system does I/O

- **Disadvantages:**
  - Partitions are *fixed* while processes have different memory requirements
  - Many systems were requiring processes to occupy a *specific partition*
Solution 3

- *Multiprogramming with variable partitions*

- OS allocates contiguous extents of memory to processes
  - Initially each process gets all the memory space it needs and nothing more

- Processes that are swapped out can return to *any* main memory location
Solution 3

- Initially everything works fine
  - Three processes occupy most of memory
  - Unused part of memory is very small
Solution 3

- When P0 terminates
  - Replaced by P3
  - P3 must be smaller than P0
- Start wasting memory space
Solution 3

- When P2 terminates
  - Replaced by P4
  - P4 must be smaller than P2 plus the free space
- wasting more memory space
External fragmentation

- Happens in all systems using multiprogramming with variable partitions

- Occurs because new process must fit in the hole left by terminating process
  - Very low probability that both processes will have exactly the same size
  - Typically the new process will be a bit smaller than the terminating process
An Analogy

- Replacing an old book by a new book on a bookshelf

- New book must fit in the hole left by old book
  - Very low probability that both books have exactly the same width
  - We will end with empty shelf space between books

- Solution it to push books left and right
Memory compaction

- When external fragmentation becomes a problem, we push processes around in order to consolidate free spaces
Memory compaction

- Works very well when memory sizes were small
Dynamic address translation

- Processes do not occupy fixed locations in main memory
  - Will let them run as if they were starting at location 0
  - MMU hardware will *add the right offset*
  - Will test first that process does not try to access anything outside its boundaries
MMU for solution 3

START Address

RAM Address

\[ \leq SIZE \]

NO trap

YES

Adder
Is it virtual or real?

- MMU translates
  - *Virtual addresses* used by the process into
  - *Real addresses* in main memory
An analogy

- Living or visiting places that makes us believe we are in a different country
  - Little Italy in San Francisco, Bazaar del Mundo in San Diego, Chinatown everywhere
  - Subdivisions with “romantic” Spanish names in California
  - Streets with names of Ivy League schools or towns hosting them (Amherst, . . .)
Another way to look at it

Virtual Address

\[ \leq \text{SIZE} \]

NO

trap

YES

START Address

Adder

Physical Address
Solution 4

- **Non-contiguous allocation**
  - Partition main memory into fixed-size entities
    - **Page frames**
      - Allocate non-contiguous page frames to processes
      - Let the MMU take care of the address translation
Non-contiguous allocation

Single process address space
Virtual v. real

- Processes are provided with the illusion of a vast linear address space
  - Virtual addresses starting at address zero

- In reality, this address space is made up of disjoint page frames
  - Non-contiguous real addresses