Evolution in Memory Management

Programs used all physical memory & executed one at a time.

Programmers divided up their programs into overlays
• memory-size (or less) partitions of program and data that would not be used at the same time
• loaded into memory under user control
  ⇒ programs larger than physical memory could execute

Multiprogramming
• several programs were memory-resident at the same time
• one executed while another waited for I/O
  ⇒ better utilization of the CPU

Evolution in Memory Management

Relocation
• programs are compiled & linked wrt address 0
• relocated to some other address in physical memory
  • base register: contains the first location of the program
  • bounds register: contains the size of the program
• relocating to physical memory
  • physical address = base register + program address
  • check if physical address is within the bounds (physical address ≤ base address + bounds value)
  • if not, an exception occurs
Evolution in Memory Management

Relocation, cont’d.

• advantages of relocation
  • allows multiple programs to reside in memory
  • allows a program to reside anywhere in memory by separating program addresses & physical addresses

• problems with relocation
  • memory fragmentation
    • unallocated space between programs
    • fragmentation get worse as over time (smaller & more numerous “holes” in memory)
    • requires copying to remove the fragments
  • still requires overlays for large programs

Virtual Memory

A model (a memory abstraction) to the programmer that:

• a program starts in location 0
• a program extends contiguously in memory
• a program has available to it the entire architectural memory space ($2^{\text{wordsize}}$ bytes):
  called the virtual address space

Paging

• implementation for virtual memory
• divide the virtual address space into fixed-size chunks, called pages
• divide physical memory into chunks of the same size, called page frames
• provide a mapping between addresses in pages & address in page frames, called address translation
• if no mapping exists
  (i.e., if a virtual address is on a page that does not have a page frame in physical memory), it is on disk and has to be paged into memory
Address Translation

Address translation:
- maps addresses in the virtual address space (virtual addresses) to locations in physical memory (physical addresses)
  - CPU emits a virtual (program-generated) address
  - memory has physical addresses
- relocation mechanism is fully associative
  - a page can reside in any page frame
- mapping techniques in:
  - software data structure (page tables) &
  - hardware cache (translation lookaside buffer)
    (we’ll cover them both later)

Address Translation Using Page Tables

Operating systems data structure
- page tables are built & maintained by the OS
- one page table per process
  - process A’s virtual addresses will map to different physical locations than process B’s
- one entry in the page table per (virtual) page: called page table entry (PTE)
- PTE fields:
  - valid bit: whether the page is mapped into memory or still resides on disk
  - page frame number or disk location
  - dirty bit: indicates whether any address on the page has been written
  - reference or use bit: set if this page was used recently
  - protection bits: access privilege (read/write/execute) for user or kernel mode
Page Table Size

Calculating page table size:

\[
\text{number of page table entries} = \frac{\text{virtual address space}}{\text{page size}}
\]

\[
\text{size of page table} = \text{number of page table entries} \times \text{size of a PTE}
\]

- there are several techniques to reduce the size of the page tables

Design Trade-offs for Page Size

Choosing a page size:

- big pages
  - better throughput from disk
  - smaller page tables
  - (internal) fragmentation
- small pages
  - lower latency to fetch a page
  - larger page tables
  (but can use techniques to reduce page table size)

Current page sizes:

- 8KB
- some machines have larger ones too