Introduction to Memory Management

CS-3013 Operating Systems
Hugh C. Lauer

(Slides include materials from Modern Operating Systems, 3rd ed., by Andrew Tanenbaum and from Operating System Concepts, 7th ed., by Silbershatz, Galvin, & Gagne)
In the Beginning (prehistoric)...

- Single usage (or batch processing) systems
  - One program loaded in physical memory at a time
  - Runs to completion
- If job larger than physical memory, use *overlays*
  - Identify sections of program that
    - Can run to a result
    - Can fit into the available memory
  - Add commands after result to load a new section
  - Example: passes of a compiler
  - Example: SAGE – *North American Air Defense System*
Still near the Beginning (multi-tasking) ...

- Multiple processes in physical memory at the same time
  - allows fast switching to a ready process
  - *Partition* physical memory into multiple pieces
    - One partition for each program
- Some modern operating systems
  - *Real-time* systems
  - Small, dedicated systems (mobile phone, automotive processors, etc.)

- Partition requirements
  - *Protection* – keep processes from smashing each other
  - *Fast execution* – memory accesses can’t be slowed by protection mechanisms
  - *Fast context switch* – can’t take forever to setup mapping of addresses

Still true in many real-time, and embedded systems
Physical Memory

Physical address space

- Empty
- Process 3
- Process 2
- Process 1
- OS Kernel

E.g., OS360

0x00000000

0x0000FFFF
Physical Memory – Process 2 terminates

Physical address space

- Process 1
- Process 3
- Empty
- Empty
- OS Kernel
Problem

What happens when Process 4 comes along and requires space larger than the largest empty partition?

- Wait
- Complex resource allocation problem for OS
- Potential starvation
Physical Memory

- Empty
- Process 3
- Empty
- Process 1
- OS Kernel

Process 4
Solution

- **Virtual Address**: an address used by the program that is translated by computer into a *physical address* each time it is used
  - Also called *Logical Address*

- When the program utters 0x00105c, …
- … the machine accesses 0x01605c
First Implementation

- **Base** and **Limit** registers
  - Base is automatically added to all addresses
  - Limit is checked on all memory references
  - Introduced in minicomputers of early 1970s

- Loaded by OS at each context switch

![Diagram showing memory management process]

- CPU sends a logical address to Limit Reg.
- If Limit Reg < Reloc Reg (yes), physical address is sent to Physical Memory.
- If Limit Reg ≥ Reloc Reg (no), an error occurs.
- Reloc Reg is added to the logical address if needed to get the physical address.
Physical Memory

Physical address space

0x00000000

0x0000FFFF

Limit

Base

Empty

Process 3

Empty

Process 1

OS Kernel

0x00000000

0x0000FFFF
Advantages

• All Program addresses relative to zero

• Built-in protection provided by \textit{Limit}
  • No physical protection per page or block

• Fast execution
  • Addition and limit check at hardware speeds within each instruction

• Fast context switch
  • Need only change base and limit registers

• Partition can be suspended and moved at any time
  • Process is unaware of change
  • Potentially expensive for large processes due to copy costs!
Physical Memory

Process 4

Process 3

Process 1

OS Kernel

Physical address space

Base

Limit

0x0000FFFF

0x00000000
Definition

• **Virtual Address Space:**
  – The address space in which a process or thread “thinks”
  – Address space with respect to which pointers, code & data addresses, etc., are interpreted
  – Separate and independent of physical address space where things are actually stored

*A fundamental decoupling of addresses uttered by program from hardware addresses*
Physical Memory

0x00000000

0x0000FFFF

Physical address space

OS Kernel

Process 1

Process 3

Empty

Empty

Limit

Base
Physical Memory

Physical address space

0x00000000

0x0000FFFF

Process 1

Process 3

Process 4

OS Kernel

Base

Limit
New Problem:— How to Manage Memory

- Fixed partitions
  - Anything having to do with managing space — warehouse design, packaging, etc.

- Variable partitions
  - Seems to make better use of space

This is a general problem with broad applicability — e.g., to files systems, databases, etc.
Partitioning Strategies – Fixed

- Fixed Partitions – divide memory into equal sized pieces (except for OS)
  - Degree of multiprogramming = number of partitions
  - Simple policy to implement
    - All processes must fit into partition space
    - Find any free partition and load the process

- Problem – what is the “right” partition size?
  - Process size is limited
  - *Internal Fragmentation* – unused memory within a partition that is not available to other processes
Partitioning Strategies – Variable

• Idea: remove “wasted” memory that is not needed in each partition
  • Eliminating *internal fragmentation*

• Memory is dynamically divided into partitions based on process needs

• Definition:
  – *Hole*: a block of free or available memory
  – Holes are scattered throughout physical memory

• New process is allocated memory from hole large enough to fit it
Variable Partitions

- More complex management problem
  - Must track free and used memory
  - Need data structures to do tracking
  - What holes are used for a process?

- **External fragmentation**
  - memory that is outside any partition and is too small to be usable by any process
Definitions – Fragmentation

- Unused space that cannot be allocated to fill a need
- **Internal** fragmentation
  - Unused or unneeded space *within* an allocated part of memory.
  - Cannot be allocated to another task/job/process
- **External** fragmentation
  - Unused space *between* allocations.
  - Too small to be used by other requests
- Applies to all forms of *spatial* resource allocation
  - RAM
  - Disk
  - Virtual memory within process
  - File systems
  - …

The Heap
Memory Allocation – Mechanism

• MM system maintains data about free and allocated memory alternatives
  – *Bit maps* – 1 bit per “allocation unit”
  – *Linked Lists* – free list updated and coalesced when not allocated to a process

• At swap-in or process create
  – Find free memory that is large enough to hold the process
  – Allocate part (or all) of memory to process and mark remainder as free

• *Compaction*
  – Moving things around so that *holes* can be consolidated
  – Expensive in OS time

See Tanenbaum, §3.2.3
Memory Allocation – Policies

- Policy examples
  - First Fit: scan free list and allocate first hole that is large enough – fast
  - Next Fit: start search from end of last allocation
  - Best Fit: find smallest hole that is adequate – slower and lots of fragmentation
  - Worst fit: find largest hole

- Simulation results show that First Fit usually works out to be the best
Can we do better?
User’s View of a Program

Logical address space

subroutine

stack

symbol table

main program

Sqrt
Memory Management — Beyond Partitions

- Can we improve memory utilization & performance
  - Processes have distinct parts
    - *Code* — program and maybe shared libraries
    - *Data* — pre-allocated and heap
    - *Stack*
  - Solution — slightly more Memory Management hardware
    - Multiple sets of “base and limit” registers
    - Divide process into logical pieces called *segments*

- Advantages of *segments*
  - Code segments don’t need to be swapped out and may be shared
  - Stack and heap can be grown — may require segment swap
  - With separate I and D spaces can have larger virtual address spaces
    - “I” = *Instruction* (i.e., code, always read-only)
    - “D” = *Data* (usually read-write)
Logical View of Segmentation

user space

physical memory space

1
2
3
4

1
4
2
3
Segmentation

• Logical address consists of a pair:
  \(<\text{segment-number, offset}>\)

• Segment table – maps two-dimensional physical addresses; each table entry has:
  – \textit{Base}: contains the starting physical address where the segments reside in memory.
  – \textit{Limit}: specifies the length of the segment.
Segment Lookup

Index to segment register table

Segment registers

<table>
<thead>
<tr>
<th>limit</th>
<th>base</th>
</tr>
</thead>
</table>

virtual address

segment # | offset

physical memory

segment 0
segment 1
segment 2
segment 3
segment 4

Physical Address

<?

raise protection fault

<?

yes

+
**Segmentation**

- **Protection.** With each pair of segment registers, include:
  - *validation bit* = 0 $\Rightarrow$ illegal segment
  - *read/write/execute* privileges

- **Protection bits** associated with segments; code sharing occurs at segment level.

- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
  - With all the problems of fragmentation!
Segmentation

- Common in early minicomputers
  - Small amount of additional hardware – 4 or 8 segments
  - Used effectively in classical Unix
- Good idea that has persisted and supported in current hardware and OSs
  - Pentium, x86 supports segments
  - Linux supports segments (sort of)
- Still have *external fragmentation* of memory
- What is the next level of Memory Management improvement?
  - Next topic
Questions?