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
Physical Memory

- Physical address space
- Empty
  - Process 3
  - Process 2
  - Process 1
  - OS Kernel

E.g., OS360
Loading a Process

- Relocate all addresses relative to start of partition
  - See *Linking and Loading*

- Memory protection assigned by OS
  - Block-by-block to physical memory
  - Base and limit registers

- Once process starts
  - Partition cannot be moved in memory
  - *Why?*
Physical Memory – Process 2 terminates

Physical address space

0x00000000

0x0000FFFF

OS Kernel

Process 1

Empty

Process 3

Empty

---

0x00000000
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 the process of logical address to physical address conversion with Base and Limit registers and Relocation Register.](image-url)
Physical Memory

Physical address space

Limit

Base

Ordered and available addresses:

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

Base Address: 0x00000000
Limit Address: 0x0000FFFF
Advantages

• No relocation of program addresses at load time
  • All addresses relative to zero!

• Built-in protection provided by *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

Physical address space

0x00000000

0x0000FFFF

OS Kernel

Process 1

Process 3

Process 4

Base

Limit
• **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

Physical address space

0x00000000

0x0000FFFF

Limit

Base

Empty

Process 3

Empty

Process 1

OS Kernel

0x00000000

0x0000FFFF
Physical Memory

0x0000FFFF

Physical address space

0x00000000

Process 4

Limit

Base

Process 3

Process 1

OS Kernel
New Problem:– How to Manage Memory

- Fixed partitions
  - Seems to make better use of space

- Variable partitions
  - Seems to make better use of space

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

Anything having to do with managing space — warehouse design, packaging, 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

[Diagram of memory management with process start and terminate events]
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
  - …
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 Management – List vs. Map

- Part of memory with 5 processes, 3 holes
  - tick marks show allocation units
  - shaded regions are free
- Corresponding bit map
- Same information as a list

![Memory Management Diagram](attachment:memory_management_diagram.png)
Memory Management – Bit Map

- Advantages:
  - Can see big picture
  - Easy to search using bit instructions in processor
  - Holes automatically coalesce

- Disadvantage
  - No association between blocks and processes that own them
Memory Management – List

- **Advantages:**
  - Direct association between block and process owning it

- **Disadvantages:**
  - Cannot see big picture
  - Searching is expensive
  - Coalescing adjacent blocks requires extra effort (sorted order)
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
Swapping and Scheduling

- **Swapping**
  - Move process from memory to disk (swap space)
    - Process is blocked or suspended
  - Move process from swap space to big enough partition
    - Process is ready
    - Set up Base and Limit registers
  - Memory Manager (MM) and Process scheduler work together
    - Scheduler keeps track of all processes
    - MM keeps track of memory
    - Scheduler marks processes as swap-able and notifies MM to swap in processes
    - Scheduler policy must account for swapping overhead
    - MM policy must account for need to have memory space for ready processes

- More in Chapter 3 of Tanenbaum
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

1
2
3
4

classical view - physical memory space

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

<table>
<thead>
<tr>
<th>segment #</th>
<th>offset</th>
</tr>
</thead>
</table>

virtual address

Segment registers

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

physical memory

| segment 0 |
| segment 1 |
| segment 2 |
| segment 3 |
| segment 4 |

yes

no

Physical Address

raise protection fault

"<"?

+
Segmentation

- **Protection.** With each pair of segment registers, include:
  - *validation bit = 0* ⇒ 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?