CISC 7310X
C06: Memory Management
Hui Chen
Department of Computer & Information Science
CUNY Brooklyn College
Outline

• Recap & issues
  • Project 1 feedback

• Memory management: main memory
  • No memory abstraction
  • Memory abstraction: address space
  • Memory segmentation

• Assignment
Project 1 Feedback

• Via commenting on git commits at Github
Questions?

• Project 1?
• Project 2?
Memory Hierarchy

- Registers: <KB
- Cache: ~MB
- Main Memory: ~GB
- Secondary Memory (e.g., Magnetic Disk): ~TB
Memory Management

• How does an OS provide an abstraction of the memory hierarchy to make it “useful”?

• Main memory

• Virtual memory

• Caching
Main Memory Management

• Without abstraction
• With abstraction
• Segmentation
No Abstraction

• Directly address physical memory
  • three variations

- Simple memory organization [Figure 3-1 in Tanenbaum & Bos, 2014]
What would happen if ... 

• Three variations 

```
int *p = malloc(...)
while (1) {
    *p = value;
    p ++;
}
```

• Simple memory organization [Figure 3-1 in Tanenbaum & Bos, 2014]
Protection and Parallelism

• When without abstraction,
  • How to provide protection: prevent a program overwrite another program or the OS?
  • How to run multiple programs?
• Need additional hardware for protection
Example: IBM 360

- Allow access only when PSW key matches RAM block key

<table>
<thead>
<tr>
<th>PSW</th>
<th>KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1101</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KEY</th>
<th>2KB Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td></td>
</tr>
</tbody>
</table>

- Stop ill-behaved program
Example: Running Multiple Programs

- Naively loading two programs to run

- Loading two programs [Figure 3-2 in Tanenbaum & Bos, 2014]
Example: Static Relocation

• OS adds an offset to every address when loading
  • If the program is to load to 16384, then add 16380 to every address
    • JMP 28 → JMP 28 + 16384 → JMP 16412
  • Needs to know which is an address, which is not
    • MOV REGISTER1, 28
    • Is “28” an address or a constant?
Directly Addressing of Physical Memory

• Simple embedded devices
  • Examples: radios, washing machines, microwave ovens, coffee machines
  • A library on ROM loads a program to run and the program addresses physical memory without abstraction
  • Example: the eCos system (http://ecos.sourceforge.org/)
    • “eCos is a single process, multiple thread operating environment. As such, memory management is not required. ... There is no notion of separate address "spaces" in eCos like there would be in a system like Linux. All threads share the same address space and access capabilities.”
    • https://sourceware.org/viewvc/ecos/
Questions?

• No abstraction: directly addressing physical memory
  • How to run multiple programs?
  • How to provide protection?
  • How to provide relocation (static relocation)?
  • Where is it commonly used?
Memory Abstraction

• Notion of address space
• Base and limit registers
• Swapping
• Managing free memory
Address Space

• Two problems
  • Protection and relocation

• A logical concept
  • A process has its own independent “address space”, a set of addresses the process can use to address memory
    • How do we establish the correspondence between the “logical address” and the address of the physical memory?
      • Dynamic relocation
Dynamic Relocation

- Example: using base and limit registers
  - Map each process’s address space onto a different part of physical memory
  - Processor has two registers
  - Base register loaded with beginning physical address
  - Limit register loaded with length of the program
Base and Limit Registers

- Hardware support, when reference to a memory, CPU does the following,

```
Base register
Address in address space in process

Yes

Within length of the limit register

No

Fault
```
Running Multiple Programs

• Base and limit registers realize a simple dynamic relocation and protection
  • Base register: dynamic relocation done by CPU
  • Limit register: protection done by CPU

• Dynamic relocation via base and limit registers [Figure 3-3 in Tanenbaum & Bos, 2014]
Base and Limit Registers: Examples

• Implementation vary
  • Whether base and limit registers are protected? Who can modify these registers?
    • CDC 6600 provides protection; Intel 8088 does not
  • Whether a processor has different base and limit registers to protect for programs (instructions) and data, respectively
Swapping

• A corollary to Parkinson’s law: “Programs expands to fill the memory available to hold them”

• Running many programs may require more memory than is available

• Two solutions
  • Swapping
  • Virtual memory
Swapping & Virtual Memory

• Swapping
  • Bringing in a process in entirety from the “disk”, run the process, and putting it back on the “disk”
  • Idle processes are mostly stored on disk

• Virtual memory
  • Allow a process to run even if it is only partially in main memory
  • To be discussed next
Swapping: Example

- Swapping of a few processes

- Swapping of multiple processes [Figure 3-4 in Tanenbaum & Bos, 2014]
Memory Compaction

• Swapping creates multiple holes in memory
• Sometimes it is necessary to combine multiple holes into big one
Growing Program Data Segment

• In many programming languages, program data can grow in size
  • Example:
    • int *a = malloc(...)
    • Object o = new Object()
  • What if memory hole is not big enough to accommodate the growing program data?
    • Move process
    • Compact memory
    • Swap out one or more processes
    • Suspend the process until more memory is available
    • Killed
Proactive Approach

- Process growing in size expected, allocate extra memory when a process is swapped in or moved
- Growing data segment
- Growing stack segment
Proactive Approach: Examples

• For data segment; and for data & stack segments

  ![Diagram](image)

  - Growing program data [Figure 3-5 in Tanenbaum & Bos, 2014]
Managing Free Memory

• Keep track of memory usage
• Data structures & algorithms
  • Bitmaps
  • Linked lists
Bitmaps

• A data structure keep tracks memory allocation
• Specify an allocation unit, have many allocation units
  • Example: 4 KB
• A bit in a bitmap indicate an allocation unit is free or allocated
  • Example
    • Free: 1
    • Allocated: 0
Bitmap: Example

- An example allocation and its bitmap

- Linked list of memory allocation [Figure 3-6 in Tanenbaum & Bos, 2014]
Design of Bitmap

• What should be the size of the allocation unit?

• Two competing factors
  • When allocation unit become smaller, bitmap larger
    • How big is a bitmap?
  • When allocation unit becomes larger, memory waste larger
    • How much waste is there?
  • How about search the bitmap to locate free memory?
Linked List

• Maintain a linked list of allocated memory and free memory segments

• Linked list of memory allocation [Figure 3-6 in Tanenbaum & Bos, 2014]
Linked List: Example

- Allocation and deallocation scenarios

Linked list of memory allocation [Figure 3-7 in Tanenbaum & Bos, 2014]
Design of Linked List

- Single-linked list or double-linked list?
- Separate list of processes or holes?
- A few algorithms
  - First fit: starting from beginning
  - Next fit: starting from last time
  - Best fit: takes the smallest hole
  - Worst fit: takes the largest hole
  - Quick fit: based on allocation pattern
- Additional data structure?
  - Example: Holes can be sorted in size (any data structure for ordered list)
Questions?

• Address space
  • Concept?
  • Simple mechanism for dynamic allocation and protection
  • Swapping
Address Space and Process

• One process one address space?
Compiled Tables

• A compiler typically builds many tables when processing a program
  1. The source text being saved for the printed listing
  2. The symbol table, names and attributes of variables.
  3. The table containing integer and floating-point constants used.
  4. The parse tree, syntactic analysis of the program.
  5. The stack used for procedure calls within compiler.

• Each of these grows continuously as compilation proceeds
Growing Tables: Example

- One address space for all?

-One address space for all [Figure 3-30 in Tanenbaum & Bos, 2014]
Segmentation

• Provide many completely independent address spaces, called segments
  • Each segment consists of a linear sequence of address
  • Different segments can have different lengths
  • Segments can grow or shrink independently without affection others
Segmentation: Example

• Segments can grow or shrink independently without affection others

• Multiple segments [Figure 3-31 in Tanenbaum & Bos, 2014]
Questions

• Concept of segmentation
Assignment

• Practice assignment
• Review Guide #1 and Take-Home Test #1