Memory Management

These slides are created by Dr. Huang of George Mason University. Students registered in Dr. Huang’s courses at GMU can make a single machine readable copy and print a single copy of each slide for their own reference as long as the slide contains the copyright statement, and the GMU facilities are not used to produce the paper copies. Permission for any other use, either in machine-readable or printed form, must be obtained form the author in writing.

Memory

- A a set of data entries indexed by addresses
- Typically the basic data unit is byte
- In 32 bit machines, 4 bytes grouped to words
- Have you seen those DRAM chips in your PC?
Logical vs. Physical Address Space

- The addresses used by the RAM chips are called physical addresses.
- In primitive computing devices, the address a programmer/processor use is the actual address.
  - When the process fetches byte 000A, the content of 000A is provided.

- In advanced computers, the processor operates in a separate address space, called logical address, or virtual address.
- A Memory Management Unit (MMU) is used to map logical addresses to physical addresses.
  - Various mapping technologies to be discussed
  - MMU is a hardware component
  - Modern processors have their MMU on the chip (Pentium, Athlon, …)
Continuous Mapping: **Dynamic Relocation**

- The processor wants byte 0010, the 4010th byte is fetched.

**MMU for Dynamic Relocation**

- CPU logical address 346 is added to relocation register 14000 to get physical address 14346, which is then used to access memory.
Segmented Mapping

- Obviously, more sophisticated MMU needed to implement this

Swapping

- A process can be *swapped* temporarily out of memory to a backing store (a hard drive), and then brought back into memory for continued execution.

- Major part of swap time is transfer time; total transfer time is directly proportional to the *amount* of memory swapped.

- Modified versions of swapping are found on many systems, i.e., UNIX, Linux, and Windows.
Contiguous Allocation

- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes then held in high memory.
- Single-partition allocation
  - Relocation register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register.
Hardware Support for Relocation and Limit Registers

Contiguous Allocation (Cont.)

- Multiple-partition allocation
  - **Hole** – block of available memory; holes of various size are scattered throughout memory.
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it.
Dynamic Storage Allocation Problem

How to satisfy a request of size $n$ from a list of free holes.

- **First-fit**: use the *first* hole big enough.
- **Best-fit**: use the *smallest* hole that is big enough
  - must search entire list, unless ordered by size
  - produces the smallest leftover hole

- **Worst-fit**: Allocate the *largest* hole
  - must also search entire list
  - produces the largest leftover hole

**First-fit** and **best-fit** better than **worst-fit** in terms of speed and storage utilization.
Compaction

- Must shuffle memory contents to place all free memory together in one large block.
  - This is called **compaction**
- Compaction must be transparent to processes.
  - This is achieved thru the relocation register.
- Must also coordinate with IO devices.

Paging

- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8192 bytes).
- Divide logical memory into blocks of same size called **pages**.
- To run a program of size $n$ pages, need to find $n$ free frames and load program.
- Set up a page table to translate logical to physical addresses.
Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number** \( (p) \) – used as an index into a page table which contains base address of each page in physical memory.
  - **Page offset** \( (d) \) – combined with base address to define the physical memory address that is sent to the memory unit.
Address Translation Architecture

Paging Example
Paging Example

Free Frames

(a) Before allocation

(b) After allocation
Implementation of Page Table

- Page table is kept in main memory.
- **Page-table base register** (PTBR) points to the page table.
- **Page-table length register** (PRLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called *associative memory* or *translation look-aside buffers* (TLBs)

Associative Memory

- Associative memory – hardware parallel search

<table>
<thead>
<tr>
<th>Page #</th>
<th>Frame #</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Address translation (A’, A’‘)
- If A’ is in associative register, get frame # A’‘ out.
- Otherwise get frame # A’‘ from page table in memory
Effective Access Time

- Associative Lookup = $\varepsilon$ time unit
- Assume memory cycle time is 1 microsecond
- Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers.
- Hit ratio = $\alpha$
- Effective Access Time (EAT)

$$EAT = (1 + \varepsilon) \alpha + (2 + \varepsilon)(1 - \alpha)$$

$$= 2 + \varepsilon - \alpha$$
Memory Protection with Paging

- Memory protection implemented by associating protection bit with each frame.
- **Valid-invalid** bit attached to each entry in the page table:
  - “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page.
  - “invalid” indicates that the page is not in the process’ logical address space.

Valid (v) or Invalid (i) Bit In A Page Table
Page Table Structure

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

- Break up the logical address space into multiple page tables.
- A simple technique is a two-level page table.
Two-Level Paging Example

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page number consisting of 20 bits.
  - a page offset consisting of 12 bits.
- Since the page table is paged, the page number is further divided into:
  - a 10-bit page number.
  - a 10-bit page offset.

Thus, a logical address is as follows:

![Page number and offset diagram]

where $p_1$ is an index into the outer page table, and $p_2$ is the displacement within the page of the outer page table.
Two-Level Page-Table Scheme

Address-Translation Scheme

- Address-translation scheme for a two-level 32-bit paging architecture
**Hashed Page Tables**

- Common in address spaces > 32 bits.
- The virtual page number is hashed into a page table. This page table contains a chain of elements hashing to the same location.
- Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.

![Hashed Page Table Diagram](image)
Inverted Page Table

- One entry for each real page of memory.
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.
- Use hash table to limit the search to one — or at most a few — page-table entries.

Inverted Page Table Architecture
Segmentation

- Memory-management scheme that supports user view of memory.
- A program is a collection of segments. A segment is a logical unit such as:
  - main program,
  - procedure/function,
  - object,
  - local variables,
  - global variables,
  - stack,
  - symbol table, arrays

User’s View of a Program
Logical View of Segmentation

Segmentation Architecture

- Logical address consists of a two tuple: 
  \[<\text{segment-number}, \text{offset}>\],
- This addressing scheme has to be directly supported by the processor, such as all x86 processors.
- In x86 assembly programs, every address follows the above format.
Segment table – maps two-dimensional physical addresses; each table entry has:
- base – contains the starting physical address where the segments reside in memory.
- limit – specifies the length of the segment.

Segment-table base register (STBR) points to the segment table’s location in memory.

Segment-table length register (STLR) indicates number of segments used by a program
- segment number $s$ is legal if $s < \text{STLR}$.

Segmentation Architecture (Cont.)

Protection. With each entry in segment table associate:
- 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.
  – first fit/best fit
A segmentation example is shown in the following diagram
Example of Segmentation

Sharing of Segments
Segmentation with Paging

Virtual Address Space

Seg 0
- Page 0
- Page 1
- Page 2
- Page 3

Seg 1
- Page 0
- Page 1

Seg 2
- Page 0
- Page 1
- Page 2

Physical Memory

Frame 0
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Frame 7
Frame 8
Frame 9
Frame 10
Frame 11
Frame 12
Frame 13

Address Translation

Logical address

Segment table

Page table for segment s

Physical address

Memory
Case Study: Intel i386 MMU

- segmentation with paging
- two-level page table

Intel 30386 Address Translation