Chapter 8: Memory Management

- Background
- Swapping
- Contiguous Allocation
- Paging
- Segmentation
- Segmentation with Paging

Main Memory

- Provide a description of organizing memory hardware
- Introduce memory management techniques:
  - paging and segmentation

Background

- Program must be brought into memory and placed within a process for it to be executed.
- Input queue – collection of processes on the disk that are waiting to be brought into memory for execution.
- User programs go through several steps before being executed.

Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at three different stages.

- Compile time: If memory location known a priori, absolute code can be generated; must recompile code if starting location changes.
- Load time: Must generate relocatable code if memory location is not known at compile time.
- Execution time: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers).

Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management
  - Logical address – generated by the CPU; also referred to as virtual address
  - Physical address – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes
- Logical (virtual) and physical addresses differ in execution-time address-binding scheme

Multistep Processing of a User Program

Logical vs. Physical Address Space
Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with logical addresses; it never sees the real physical addresses

Dynamic Relocation Using a Relocation Register

Dynamic Loading

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required (implemented through program design)

Dynamic Linking

- Linking postponed until execution time
- Small piece of code, stub, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system needed to check if routine is in processes’ memory address
- Dynamic linking is particularly useful for libraries

Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
- Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- Roll out, roll in – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- 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)

Swapping (Cont.)

- Normally, a swapped out process will be swapped back into the same memory space it occupied previously.
  - Reason: dictated by address binding method.
    - Binding at load or assembly time, then processes cannot be moved!
    - Execution time binding: processes can be moved.
- Swapping is costly
  - E.g., user size of a process:
    - Transfer time: 10 Mbyte/40 Mbyte/sec = 250 ms
    - Average latency: 8 ms
    - Seek time: often zero ms
    - Total: 2*(250+8) = 512 ms
- Quantum when using Round-Robin should be >> 512 milliseconds
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 scheme used to protect user processes from each other, and from changing operating-system code and data
  - 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
  - Mapping from logical to physical addresses is done by MMU

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
  - Operating system maintains information about:
    - allocated partitions
    - free partitions (hole)
- First-fit: Allocate the first hole that is big enough.
- Best-fit: Allocate 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.

Dynamic Storage-Allocation Problem

How to satisfy a request of size \( S \) from a list of free holes.

- First-fit: Allocate the first hole that is big enough.
- Best-fit: Allocate 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 are better than worst-fit in terms of speed and storage utilization (shown by simulation studies).
### Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous.

- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.

- Reduce external fragmentation by compaction:
  - Shuffle memory contents to place all free memory together in one large block.
  - Compaction is possible only if relocation is dynamic, and is done at execution time.
  - I/O problem:
    - Latch job in memory while it is involved in I/O.
    - Do I/O only into OS buffers.

### Example of Compacting

- Solution 1:

- Solution 2:

### Paging

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available.

- 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.

- Keep track of all free frames: frame table - maintained by OS.

- 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.

- Internal fragmentation.

### 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 Examples

Free Frames

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:
1) one for the page table and 2) 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/register or
- translation look-aside buffers (TLBs)

Associative Register

- Associative memory – parallel search

Address translation (A’ , A”)
+ If A’ is in associative register, get frame number out
+ Otherwise get frame number from page table in memory
( TLB miss )
  - If so, include frame in associative register (updating)

Paging Hardware with TLB
Effective Access Time

- **Associative Lookup** = $\varepsilon$ time unit (e.g., $\varepsilon=20$ nanoseconds)
- **Assume memory cycle time** is $\beta$ (e.g., $\beta=100$ nanoseconds)
- **Hit ratio** $\alpha$ 
  - percentage of times that a page number is found in the associative registers 
  (e.g., $\alpha=80\%$; normally between 80-98)
- **Effective Access Time (EAT)**
  \[
  EAT = (\beta + \varepsilon) \times \alpha + (2 \times \beta + \varepsilon)(1 - \alpha)
  \]
  \[= 120\times0.8 + 220\times0.2 = 140 \text{ nanoseconds}\]

Memory Protection

- **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.

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:

\[
\begin{array}{c|c|c}
\text{page number} & \text{page offset} & \alpha \\
\hline
p_1 & p_2 & d \\
\end{array}
\]

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.
Address-Translation Scheme

- Address-translation scheme for a two-level 32-bit paging architecture

Hashed Page Table

- 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

- 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: [ <process id>, <page number>, <offset> ]
- 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

- 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: [ <process id>, <page number>, <offset> ]
- 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

- Shared Pages
  - Shared code
    - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
    - Shared code must appear in same location in the logical address space of all processes
  - Private code and data
    - Each process keeps a separate copy of the code and data
    - The pages for the private code and data can appear anywhere in the logical address space
Shared Pages Example

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, method,
  - object,
  - local variables, global variables, common block, stack,
  - symbol table, arrays,

Segmentation Architecture

- Logical address consists of a two tuple: <segment-number, offset>
- 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 < STLR

Segmentation Architecture (Cont.)

- Relocation
  - dynamic
  - by segment table
- Sharing
  - shared segments
  - same segment number
- Allocation
  - first fit/best fit
  - external fragmentation

Segmentation Hardware

Segmentation Architecture (Cont.)

- Protection. With each entry in segment table associate:
  - 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
Examples of Segmentation

Sharing of segments

Segmentation and its segment table

Recommended Reading and Exercises

- Reading
  - [SGG7] Chapter 8
    - Section 8.7 (Intel Pentium Memory Management)
  - Chapter 9 (sixth edition)

- Exercises
  - All

- No project