CS 3723 Operating Systems: Memory Management (SGG-08)

Instructor: Dr. Tongping Liu

Thank Dr. Dakai Zhu and Dr. Palden Lama for providing their slides.

Objectives

- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques, including paging and segmentation
- To provide a detailed description of the Intel Pentium, which supports both pure segmentation and segmentation with paging

Background

- Think memory as an array of words containing program instructions and data
- How do we execute a program?
  - Fetch an instruction → decode → may fetch operands → execute → may store results
- Memory unit sees a stream of ADDRESSES
- How to manage and protect main memory while sharing it among multiple processes?
  - Keeping multiple process in memory is essential to improving the CPU utilization

Memory Management

- Background
- Swapping
- Contiguous Memory Allocation and Fragmentation
- Paging
- Structure of the Page Table
- TLB

Simple one: Base and Limit Registers

- Memory protection is required to ensure correct operation
- A pair of base and limit registers define the logical address space of a process. Every memory access is checked by hardware. Any problem? Too slow
Why we need address binding?

- A process can reside in any part of physical memory.
- How to handle the addresses?
  - Addresses in the source code is symbolic, such as "count"
  - After the compilation, addresses may be relocatable, such as offset 14 from the start
  - Linker or Loader turns this into an absolute address, such as 0x100014.
- Binding: is a mapping between one address space to another.

Binding 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 is known beforehand, **absolute code** can be generated; must recompile code if starting location changes (e.g. DOS .com programs)
  - **Load time**: Compiler generates **relocatable code**, final binding occurred at load time
  - **Execution time**: Binding delayed until execution time if the process can be moved during its execution from one memory segment to another

Logical vs. Physical Address Space

- **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
  - The mapping form logical address to physical address is done by a hardware called a **memory management unit** (MMU).
  - We will mainly study how this mapping is done and what hardware support is needed

Memory-Management Unit (MMU)

- **logical addresses** (in the range 0~max) and physical addresses (in the range R+0 to R+max for a base value R).
- The user program generates only logical addresses and thinks that the process runs in locations 0 to max.
- **logical addresses must be mapped to physical addresses** before they are used

Dynamic Loading

- Why dynamic loading?
  - Without this, the size of a process is limited to that of physical memory
- Dynamic loading:
  - Dynamically load routines when they are called
  - All other routines are kept on disk in a loadable format
  - Better memory-space utilization since unused routines are never loaded
  - Useful when large amounts of code are needed to handle infrequently occurring cases like error handling
**Dynamic Linking and Shared Libraries**

- Static linking
  - System language and library routines are included in the binary code.
- Dynamic linking: similar to dynamic loading
  - Linking postponed until execution time.
  - Without that, every library will have a copy in the executable file, wasting both disk space and memory.
  - A stub is used to locate and load the appropriate memory-resident library routine, when the routine is not existing.
  - If it is existing, no need for loading again (PLT).
- Dynamic linking is particularly useful for libraries (one copy, transparent updates).

**Memory Management**

- Background
- Swapping
- Contiguous Memory Allocation and Fragmentation
- Paging
- Structure of the Page Table
- TLB

**Swapping**

Consider a multi-programming environment:
- Each program must be in the memory to be executed.
- Processes come into memory and leave memory when execution is completed.

![Swapping Diagram](image)

What if no free region is big enough?

- Backing store – 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.
- Swapping can free up memory for additional processes.

**Swapping (cont’ d)**

- Major part of swap time is transfer time:
  - Total transfer time is directly proportional to the amount of memory swapped (e.g., 10MB process / 40MB per sec = 0.25 sec).
  - May take too much time to be used often.
- Standard swapping requires too much swapping time.
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows), but it is often disabled.

**Memory Management**

- Background
- Swapping
- Contiguous Memory Allocation and Fragmentation
- Paging
- Structure of the Page Table
- TLB
Contiguous Allocation

- Main memory is usually divided into two partitions:
  - Resident operating system, usually held in low memory
  - User processes, usually held in high memory
- Relocation registers are used to protect user processes from each other, and from changing operating-system code and data
  - MMU maps logical address to physical addresses dynamically
  - But the physical addresses should be contiguous

Dynamic Storage-Allocation Problem

How to satisfy a request of size \( n \) 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. But all suffer from fragmentation

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 internal fragmentation

How can we reduce external fragmentation

- **Compaction**: Move memory to place all free memory together in one large block, possible only if relocation is dynamic and is done at execution time

I/O problem \( \rightarrow \) large overhead

Paging: Basic Ideas

- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16MB or more
- Divide logical memory into blocks of same size called **pages**
- To run a program with size \( n \) pages, we need \( n \) free frames
- Set up a **page table** to translate logical to physical addresses
  - User sees memory a contiguous space (0 to MAX) but OS does not need to allocate it this way
Paging: Internal Fragmentation

- Calculating internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
  - On average fragmentation = 1 frame – 1 byte

- So small frame sizes desirable? → more entries
  - Each page table takes memory to track

- Page sizes growing over time
  - Solaris supports two page sizes – 8 KB and 4 MB

Address Translation

- Suppose the logical address space is $2^m$ and page size is $2^n$ so the number of pages is $2^m / 2^n$, which is $2^{m-n}$

- Logical Address ($m$ bits) 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

Demonstration of page numbers and page offsets:

(page number | page offset)

$m - n$ $n$

Paging Hardware

Suppose the page size is 4-byte pages.

Paging Example

What would you say about the size of logical address space, the size of physical address space, and the size of the page table?

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
**Another Example**

- Example:
  - 64 KB virtual memory
  - 32 KB physical memory
  - 4 KB page/frame size → 12 bits as offset (d)

**Address Translation**

- Virtual address: 16 bits
- How many virtual pages?

- Frame #: 3 bits
- Offset: 12 bits
- How many physical frames?
- Physical address: 15 bits

**Page Table Size for 32bit System**

- Modern Systems/Applications
  - 32 bits virtual address
  - System with 1GB physical memory → 30 bits physical address
  - Suppose the size of one page/frame is 4KB (12 bits)

- Page table size
  - # of virtual pages: 32 – 12 = 20 bits → 210 PTEs
  - Page table size = PTE size * 2^10 = 4 MB per process → 2^10 frames

- If there are 128 processes
  - Page tables occupy 128 * 4MB = 512 MB
  - 50% of memory will be used by page tables?

**More on Page Table**

- Different processes have different page tables
  - CR3 points to the page table
  - Change CR3 registers when context switches

- Page table resides in main (physical) memory
  - Continuous memory segment

**Why??**

**Address Translation Architecture**

- How big is the page table?

- CPU
- Virtual address
- Page table
- Frame number
- Page offset
- Page number
- Physical memory
- How can we get smaller page table?!!