Memory Management

CS 502
Spring 99
WPI MetroWest/Southboro Campus

Memory Management Outline

- Background
- Logical versus Physical Address Space
- Swapping
- Contiguous Allocation
  - Static Partitions
  - Dynamic Partitions
- Paging
- Segmentation
- Segmentation with Paging

Background

- Program must be brought into memory and placed within a process for it to be executed.
- Process 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.

2

Binding of Instructions and Data to Memory

- **Compile Time** — If final physical memory location is known a priori, *absolute* code can be generated. This implies that code must be recompiled if starting location changes.
- **Load Time** — Compiler or assembler must generate relocatable code if memory location is not known at compile time. Loader transforms relocatable code based on starting load address.
- **Execution Time** — Binding is delayed until run time. This enables the process to be moved during its execution from one portion of memory to another. This requires hardware support.

3

Dynamic Loading

- Routine is not loaded into memory 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.

4

Dynamic Linking

- Linking of module is deferred until execution time.
- Small piece of code, called a stub, is 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

5

Overlays

- Keep in memory only those instructions and data that are needed at any given time.
- Needed when process is larger than amount of memory allocated to it.
- Implemented by user, no special support needed from operating system; programming design of overlay structure is complex.

6

Logical versus 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 a 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.

7

Memory Management Unit (MMU)

- Hardware device that maps virtual to physical addresses.
- 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.

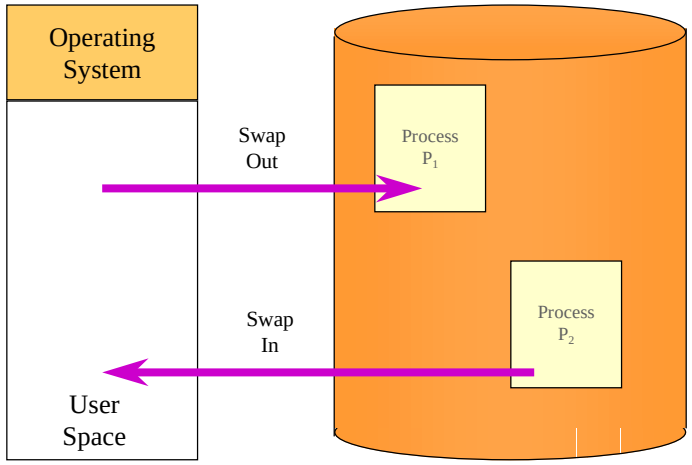
8

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 and Microsoft Windows.

9

Schematic View of Swapping



10

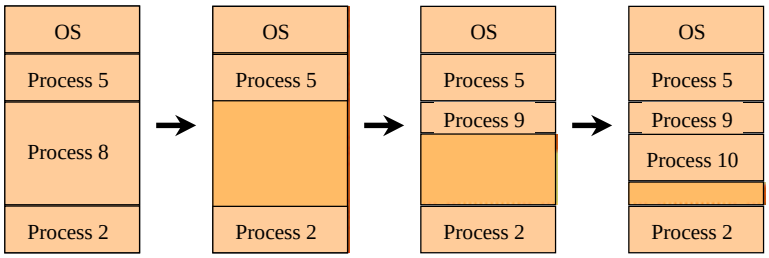
Contiguous Allocation

- 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.

11

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:
    a) allocated partitions b) free partitions (hole)



12

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 better than worst-fit in terms of speed and storage utilization.

13

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.

14

Paging

- Logical address space of a process can be noncontiguous; process is allocated physical memory wherever 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.
- 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.

15

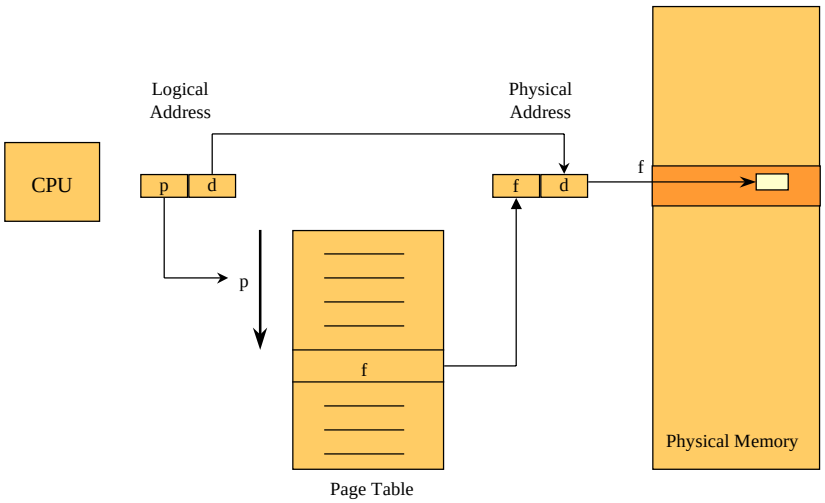
Address Translation Scheme

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.

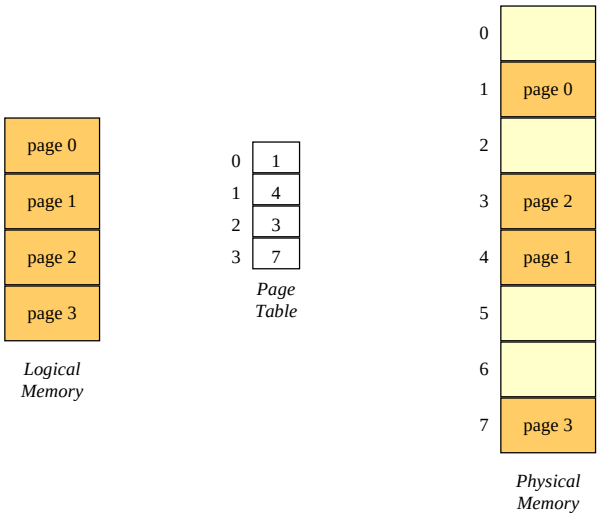
16

Address Translation Architecture



17

Paging Example



18

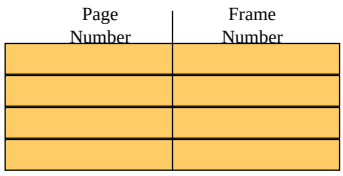
Implementation of Page Table

- Page table is kept in main memory.
- *Page-table base register* (PTBR) points to the page table.
- *Page-table length register* (PTLR) 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 registers or translation look-aside buffers (TLBs).

19

Associative Registers

- Associative Registers — parallel search



- Address Translation (p, d)
  - If p is in an associative register, get frame number out
  - Otherwise, translate through page table in memory

20

Effective Access Time

- Associative lookup = ε 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 and locality of process
- Hit ratio = α
- Effective Access Time (EAT)

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

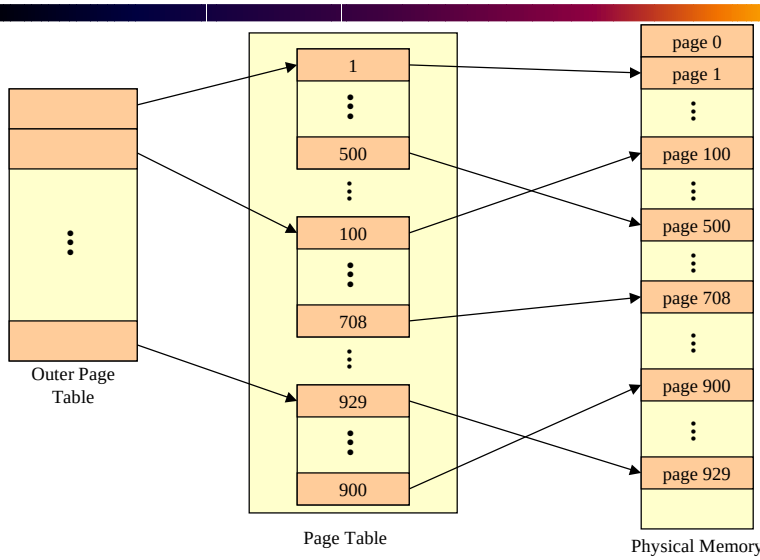
21

Memory Protection

- Memory protection implemented by associating protection bits 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.
- Extend mechanism for access type (read, write, execute)

22

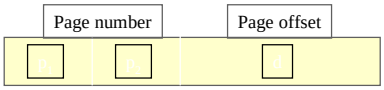
Two Level Paging Scheme



23

Two Level Paging Example

- A logical address (on 32 bit machine with 4K page size) is divided into:
  - a logical 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 offset
- Thus, a logical address is as follows:

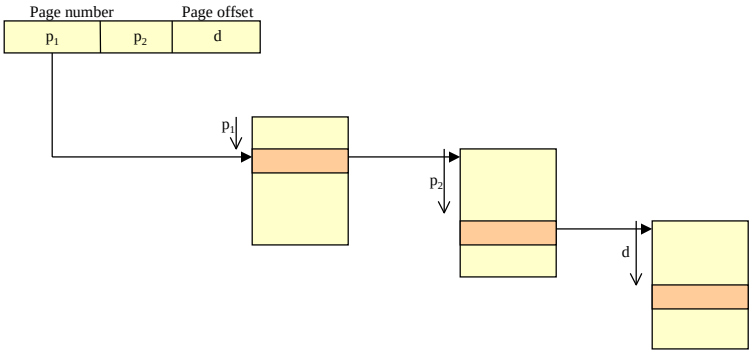


  - where p1 is an index into the outer page table, and p2 is the displacement within the page of the inner page table.

24

Address–Translation Scheme

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



25

Multilevel Paging and Performance

- On a two-level paging scheme, two memory accesses are required to convert from logical to physical, plus the memory access for the original reference.
- To make this or higher levels of paging performance feasible, caching of translation entries is required
- Example:
  - 4-level paging scheme; 100 nsec access time; 20 nsec TLB lookup time; 98% TLB hit rate:

    EAT = 0.98 x 120 + 0.02 x 520

    = 128 nsec.
  - Which is only a 28 percent slowdown in memory access time.

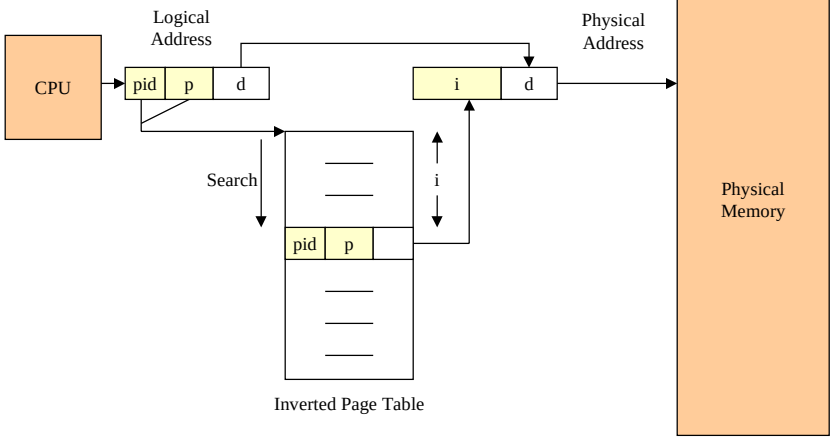
26

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.

27

Inverted Page Table Architecture



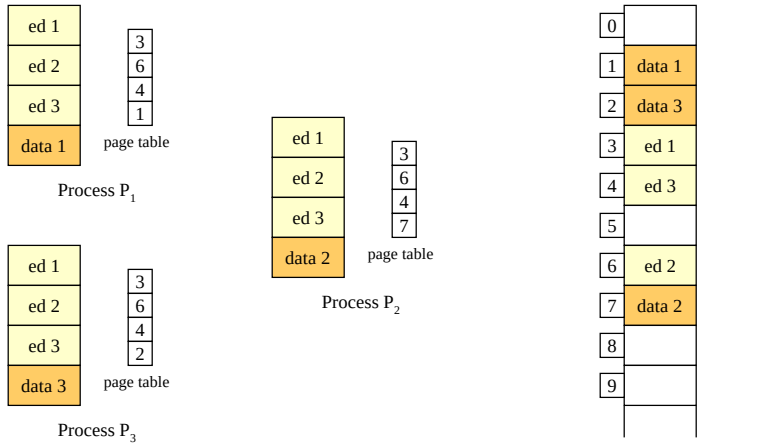
28

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 the 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.

29

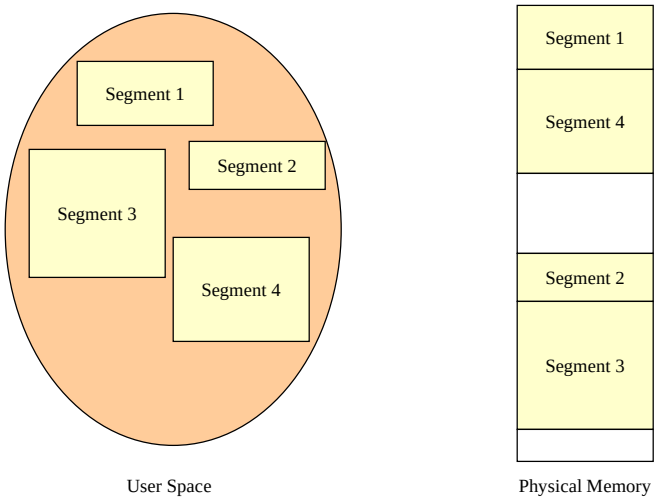
Shared Pages Example



Segmentation

- Memory–management scheme that supports a logical user view of memory.
- A program is a collection of segments. A segment is a logical unit such as:
  - Main program
  - Procedure
  - Function
  - Local variables
  - Global variables
  - Common block
  - Stack
  - Heap
  - Symbol Table
  - Arrays

Logical View of Segmentation



32

Segmentation Architecture

- Logical address consists of a two–tuple:
  - <segment-number, offset>
- Segment table – maps two-dimensional user-defined addresses into one-dimensional physical address; each entry has:
  - base – contains the starting physical address where the segment resides 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 the number of segments used by a program

33

Segmentation Architecture (Cont.)

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

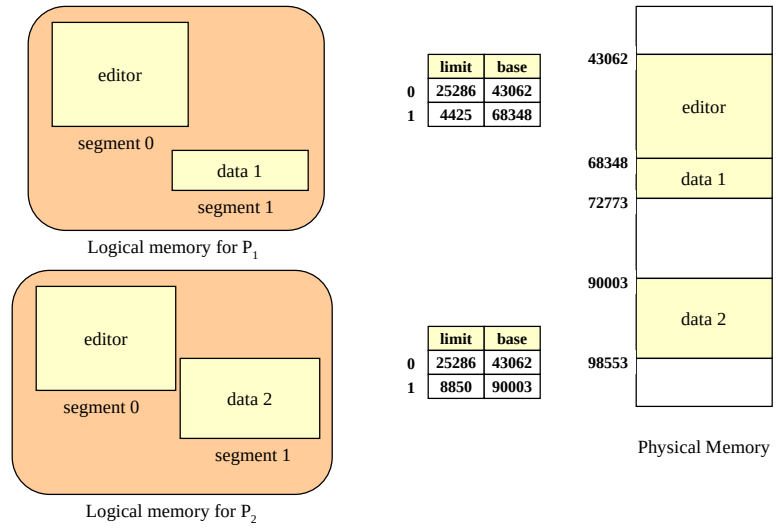
34

Segmentation Architecture (Cont.)

- Protection. With each entry in segment table associate:
  - valid/invalid bit
  - read/write/execute access modes
- Protection bits are associated with segments; code sharing occurs at the segment level.
- Since segments vary in length, memory allocation is a dynamic storage–allocation problem.

35

Segmentation Example



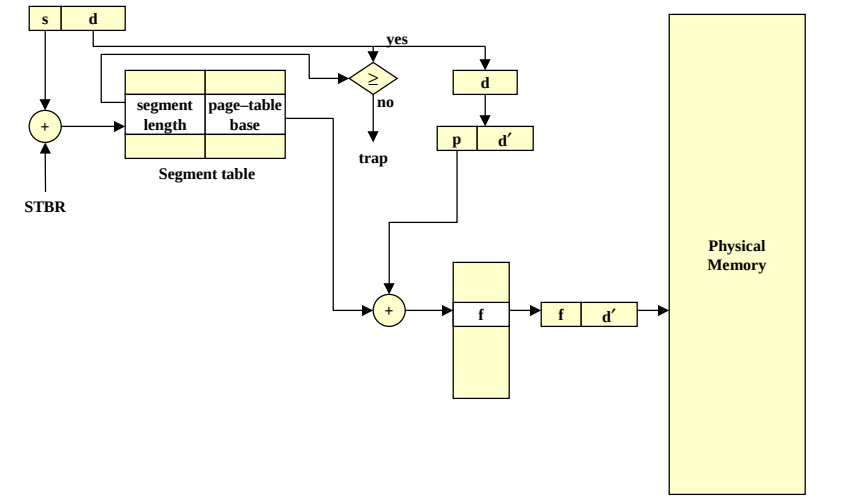
36

Segmentation with Paging – MULTICS

- The MULTICS system solved the problems of external fragmentation and lengthy search times by paging the segments.
- Solution differs from pure segmentation in that the segment–table entry contains not the base address of the segment, but rather the base address of a *page table* for the given segment.

37

MULTICS Address Translation Scheme



38

Comparing Memory–Management Strategies

- Hardware support
- Performance
- Fragmentation
  - Internal and External
- Relocation
- Swapping
- Sharing
- Protection

39