CSC 4103 - Operating Systems
Spring 2007

LECTURE - XII

MAIN MEMORY - II

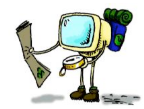
Tevfik Koşar

Louisiana State University

March 8th, 2007

Roadmap

- Dynamic Loading & Linking
- Contiguous Memory Allocation
- Fragmentation
- Paging
- Segmentation



Dynamic Loading

- Used to increase memory space utilization
- A routine is not loaded until it is called
  - All routines do not need to be in memory all time
  - Unused routines never loaded
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required to implement
- When a routine needs to call another routine:
  - Caller first checks if that routine is already in memory
  - If not, loader is called
  - New routine is loaded, and program's address tables updated

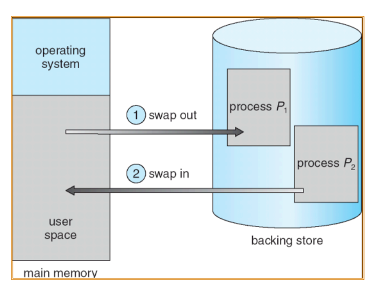
Dynamic Linking

- Linking postponed until execution time
- Otherwise each program should have a copy of its language library in its executable image
- Small piece of code, *stub*, used to locate the appropriate memory-resident library routine or how to load it
- Stub replaces itself with the address of the routine, and executes the routine
- The next time, library routine is executed directly, without need to reload
- All processes that use a language library execute only one copy of the library code
- Also useful for library updates and bug fixes
- Dynamic linking requires support from OS

Swapping

- A process must be in memory for execution
- 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

Schematic View of Swapping



Swapping *(cont.)*

- A swapped out process will be swapped back into the same memory space occupied previously.
- Ready queue: processes whose memory images are in the backing store or in memory and ready to run
- When the CPU decides to execute a process, it calls the dispatcher.
- The dispatcher checks if the process is in the memory.

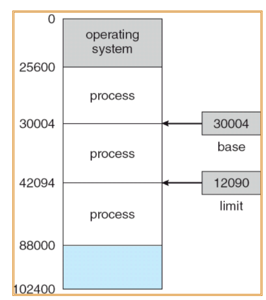
Swapping *(cont.)*

- Average swap time for a 10MB process
  - → ~ ½ seconds
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- Time quantum in multiprogramming should be substantially larger than swap time
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

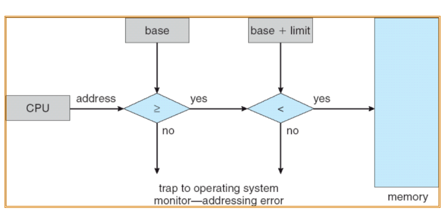
Contiguous Allocation

- Main memory usually divided 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

A base and a limit register define a logical address space

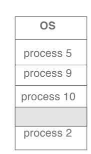


HW address protection with base and limit registers



Contiguous Allocation (Cont.)

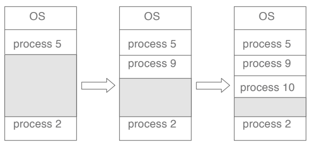
- Multiple-partition allocation
  - Divide memory into fixed-size partitions
  - Each partition contains exactly one process
  - The degree of multi programming is bound by the number of partitions
  - When a process terminates, the partition becomes available for other processes



→no longer in use

Contiguous Allocation (Cont.)

- Fixed-partition Scheme
  - When a process arrives, search for a hole large enough for this process
  - Hole – block of available memory; holes of various size are scattered throughout memory
  - Allocate only as much memory as needed
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)



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

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous (in average ~50% lost)
- **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

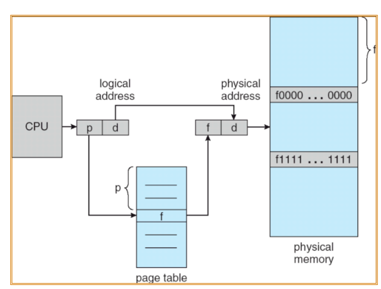
Paging

- 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
- 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

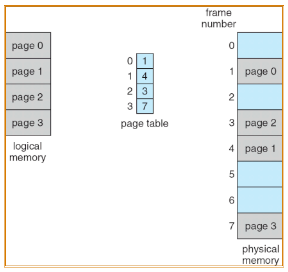
Address Translation Scheme

- 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

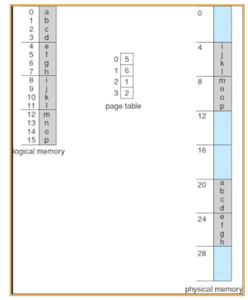
Address Translation Architecture



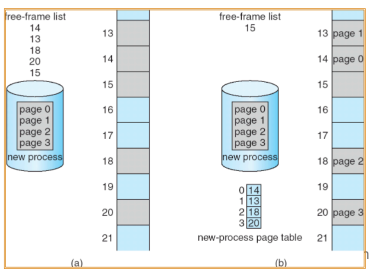
Paging Example



Paging Example



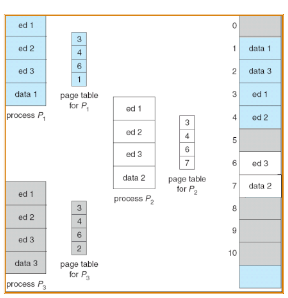
Free Frames



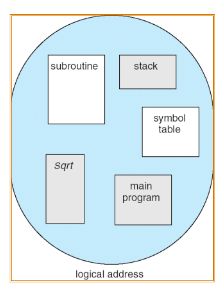
Shared Pages

- **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



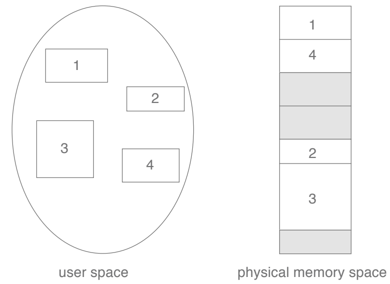
User's View of a Program



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

Logical View of Segmentation



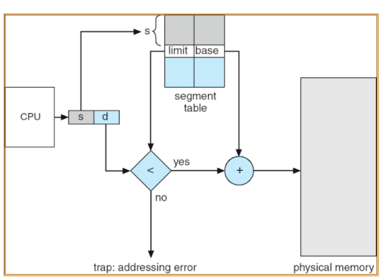
Segmentation Architecture

- 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

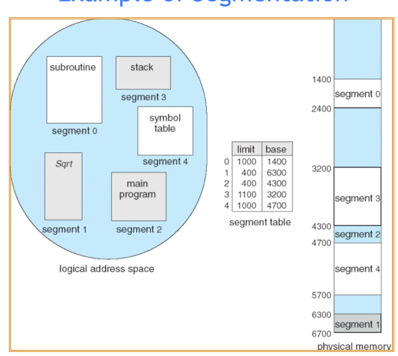
Segmentation Architecture (Cont.)

- 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
- A segmentation example is shown in the following diagram

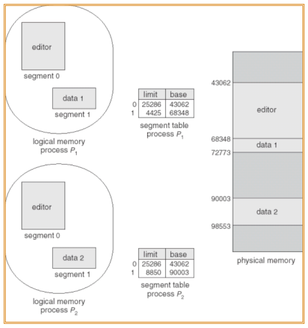
Address Translation Architecture



Example of Segmentation



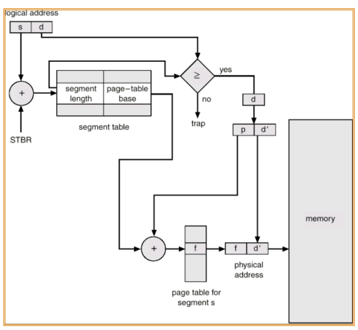
Sharing of Segments



Segmentation with Paging

- Modern architectures use segmentation with paging (or paged-segmentation) for memory management.

MULTICS Address Translation Scheme



Any Questions?

Reading Assignment

- Read chapter 8 from Silberschatz.

Acknowledgements

- “Operating Systems Concepts” book and supplementary material by Silberschatz, Galvin and Gagne.