BUCKNELL UNIVERSITY
Computer Science

CSCI 315 Operating Systems Design

Memory Management
-- Contiguous Allocation and Paging

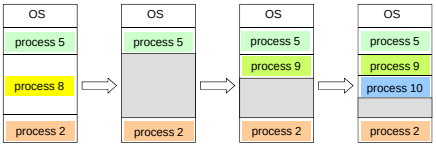
Notice: The slides for this lecture have been largely based on those accompanying a previous edition of the course text *Operating Systems Concepts, 9th ed.* by Silberschatz, Galvin, and Gagne. Many, if not all, the illustrations contained in this presentation come from this source. Revised by Xiannong Meng based on Perrone's notes.

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.

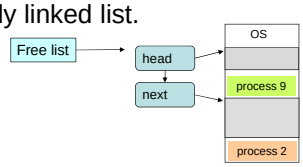
Contiguous Allocation

- 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)



Free List

- A data structure is needed for maintaining the *free list*, memory blocks that are free to use.
- Common one to use is a linked list, most likely doubly linked list.



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.
- **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.

Try on your Linux system: %getconf PAGESIZE

Address Translation Scheme

Address generated by CPU is divided into:

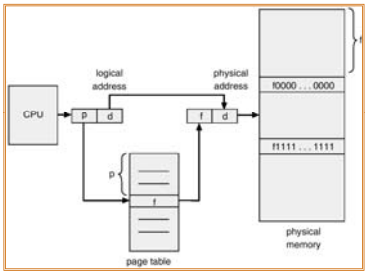
- ***Page number (p)*** – used as an index into a *page table* which contains base address of each page in physical memory.
- ***Page offset (d)*** – combined with base address to define the physical memory address that is sent to the memory unit.

Page Address Division

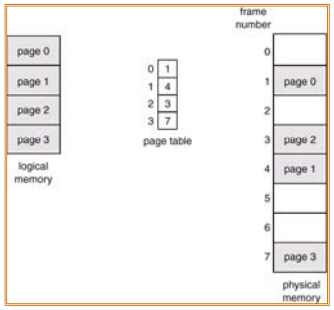
page number	page offset
p	*d*
m - n	*n*

For given logical address space 2^m and page size 2^n

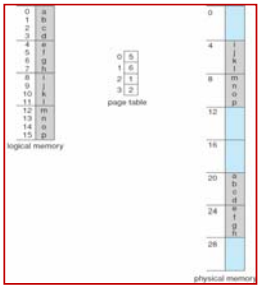
Address Translation Architecture



Paging Example



Paging Example

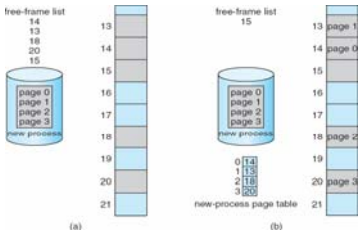


n=2 and *m*=4 32-byte memory and 4-byte pages

Paging Example

- 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
  - Worst case fragmentation = 2,048 bytes (1 frame) – 1 byte
  - On average fragmentation = 1 / 2 frame size
  - So small frame sizes desirable?
  - But each page table entry takes memory to track
  - Page sizes growing over time
    - Solaris supports two page sizes – 8 KB and 4 MB
- No external fragmentation
- Process view and physical memory now very different
- By implementation process can only access its own memory

Free Frames



Before allocation After allocation

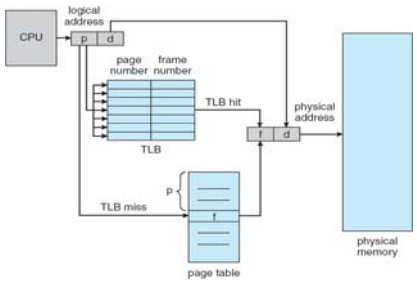
Page Table Implementation

- Page table is kept in main memory when the process is running
- **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 memory** or **translation look-aside buffers** (**TLBs**)

Translation Look-aside Buffers

- Some TLBs store **address-space identifiers** (**ASIDs**) in each TLB entry – uniquely identifies each process to provide address-space protection for that process
  - Otherwise need to flush at every context switch
- TLBs typically small (64 to 1,024 entries)
- On a TLB miss, value is loaded into the TLB for faster access next time
  - Replacement policies must be considered
  - Some entries can be **wired down** for permanent fast access

Paging Hardware With TLB



Effective Memory Access Time

- Associative Lookup = ε time unit
  - Can be < 10% of memory access time
- Memory access : 1 time unit
- Hit ratio = α
  - Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers and replacement algorithm
- **Effective Access Time** (**EAT**)

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

Effective Memory Access Time

- Associative Lookup = ε time unit
  - Can be < 10% of memory access time
- Memory access : 1 time unit
- Hit ratio = α
  - Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers and replacement algorithm
- Consider α = 80%, ε = 20ns for TLB search, 100ns for memory access
- **Effective Access Time (EAT)**

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

- Consider α = 80%, ε = 20ns for TLB search, 100ns for memory access
  - EAT = 0.80 x 120 + 0.20 x 220 = 140ns
- Consider more realistic hit ratio -> α = 99%, ε = 20ns for TLB search, 100ns for memory access
  - EAT = 0.99 x 120 + 0.01 x 220 = 121ns

Hierarchical Page Tables

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

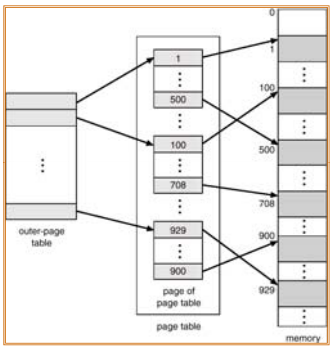
Two-Level Paging Example

- 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		page offset
p_1	p_2	d
10	10	12

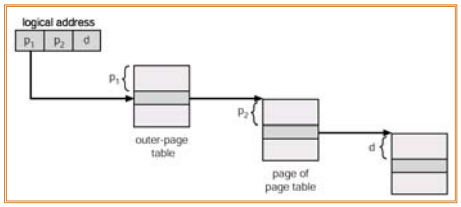
where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the outer page table.

Two-Level Page-Table Scheme



Address-Translation Scheme

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



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