Basic Memory Management

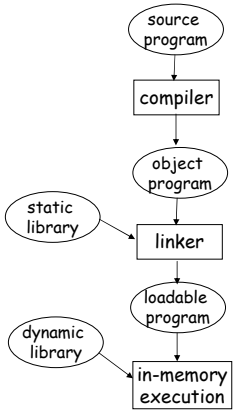
CS 256/456
Dept. of Computer Science, University of Rochester

Basic Memory Management

- Program must be brought into memory and placed within a process for it to be run.
- Mono-programming
  - running a single user program at a time
- Need for multi-programming
  - overlapping I/O with CPU
- Memory management task #1:
  - Allocate memory space among user programs (keep track of which parts of memory are currently being used and by whom).

Running a user program

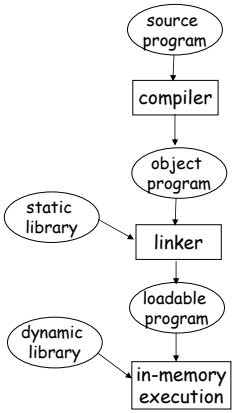
- User programs go through several steps before being run.



Address Binding

Address binding of instructions and data to memory addresses can happen at 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.
- Compare them on flexibility & protection & overhead

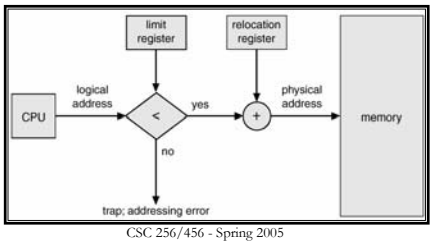


Logical vs. Physical Address Space

- Two different addresses for execution-time addressing binding:
  - *Logical address* – those in the loaded user program; seen directly by the CPU; also referred to as *virtual address*.
  - *Physical address* – address seen by the memory unit.
- Address translation from logical addresses and physical addresses must be (mostly) done in hardware
  - Why?
  - Memory-mapping unit: hardware device that maps virtual to physical address.
- Memory management task #2:
  - address translation and protection.

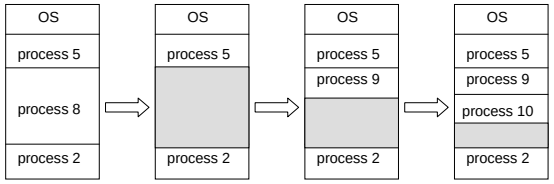
Contiguous Allocation

- Contiguous allocation
  - allocate contiguous memory space for each user program
  - Logical address always starts from 0.
- MMU: address translation and protection
  - Relocation register contains starting physical address;
  - Limit register contains range of logical addresses – each logical address must be less than the limit register.



Contiguous Allocation (Cont.)

- Memory space allocation
  - Available memory blocks of various size are scattered throughout memory.
  - When a process arrives, it is allocated memory from a free block large enough to accommodate it.
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)



Track Free Space

- Keep track of free space:
  - bitmaps
    - 2GB physical memory, 4KB basic allocation unit ⇒ size of the bitmap?
  - free block chain
- Compare them on the space overhead and the performance.

Space Allocation Strategies

How to satisfy a request of size *n* from a list of free memory blocks (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.

Compare them on space 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 minimal allocation unit, but not being used.
- Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block.
  - Issues:
    - overhead
    - problems with programs currently doing I/O

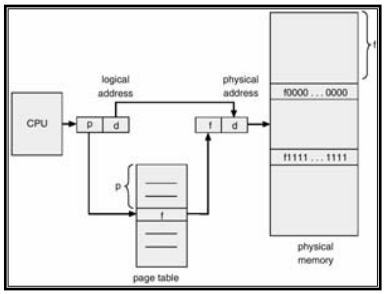
Paging (non-contiguous allocation)

- Physical 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** (typically 4KB).
- Divide logical memory into blocks of same size called **pages**.
- To run a program of size *n* pages, need to find *n* free frames and load program.
- Internal fragmentation.

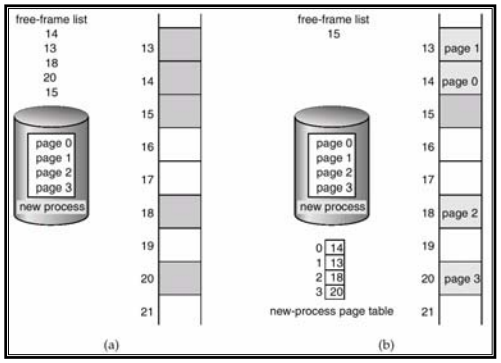
Paging: Address Translation Scheme

A logical address 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)* – the offset address with each page/frame. The same for both logical address and physical address.



Load A User Program: An Example

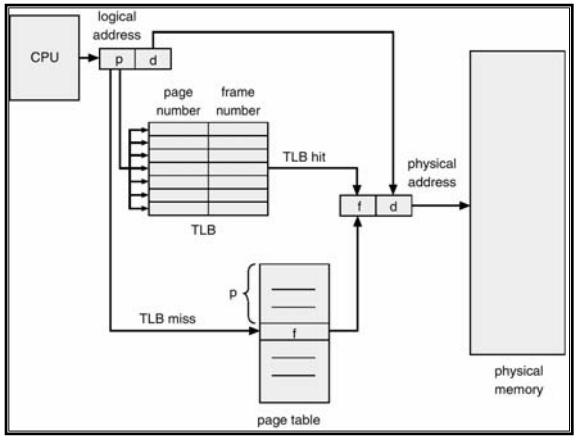


Before loading After loading

Implementation of Page Table

- Page table is (usually) kept in main memory
  - why not in registers?
  - kernel or user space?
- Hardware MMU:
  - *Page-table base register* points to the page table.
  - *Page-table length register* 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.
- Solution:
  - A special fast-lookup hardware cache called *translation look-aside buffers (TLBs)*

Paging MMU With TLB



Effective Access Time

- Assume
  - TLB Lookup = 1 ns
  - Memory cycle time is 100 ns
- Hit ratio (α)– percentage of times that a page number is found in the TLB.
- Effective memory Access Time (EAT)

$$EAT = 101 \times \alpha + 201 \times (1 - \alpha)$$

Memory Protection

- How is memory protection achieved?
  - protecting the page tables in the kernel memory space.
- Parts of the logical address space may not be mapped
  - *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.
  - Software exception if attempting to access an invalid page.

Disclaimer

- Parts of the lecture slides contain original work of Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Andrew S. Tanenbaum, and Gary Nutt. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).