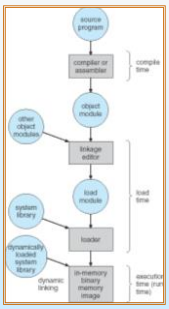
Memory Management 1

- Background
- Swapping
- Contiguous Allocation
- Variable size Multiple-partition allocation
- Multiprogramming and memory management

Background

- Program must be resident in the memory to be executed within context of a process.
- **Input queue (Long Term Scheduling)** – collection of programs on the disk that are waiting to be brought into memory to run within respective processes.
- User programs go through tree main steps: compile, load, execute

Steps of Processing a User Program



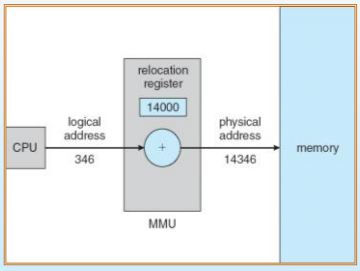
Logical vs. Physical Address Space

- The concept of a logical *address space* to be bound to a *physical address space* is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as *virtual address*
  - **Physical address** – address seen by the memory unit
- Logical addresses are valid in compile-time, load-time and execution time.
- physical addresses are valid in execution time only

Memory-Management Unit (MMU)

- Virtual address need to be mapped to physical address just before execution.
- For example, in MMU schemes, the relocation register is updated so that the actual physical address is created by adding the content of the relocation register to the logical address generated by a user process just before the execution.
- The user programs deal with *logical* addresses; it never sees the *real* physical addresses

Example: Dynamic relocation using a relocation register



Dynamic Loading Schemes

- A program routine is not loaded until it is called
- Advantages:
  - 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

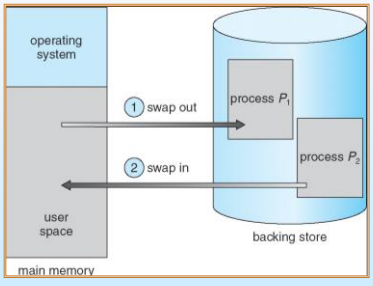
Dynamic Linking

- Linking to library routines is postponed until execution time
- Small piece of code, *stub*, used to locate the appropriate memory-resident library routine
- A so called **stub function** replaces itself with the address of the routine, and facilitates the execution of the routines.
- Dynamic linking is particularly useful for dynamically loadable library routines.

Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution (Mid term scheduling)
- **Backing store** – disk space allocated should be large enough to accommodate copies of all memory images for all users; must provide direct access to these images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; a lower-priority process is swapped out to create main memory space so that a higher-priority process can be loaded and executed
  - Total transfer time is directly proportional to the amount of memory swapped
  - Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

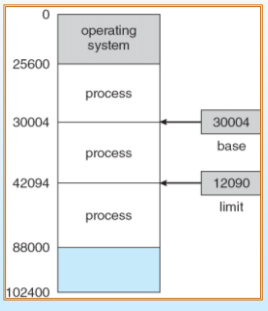
Schematic View of Swapping



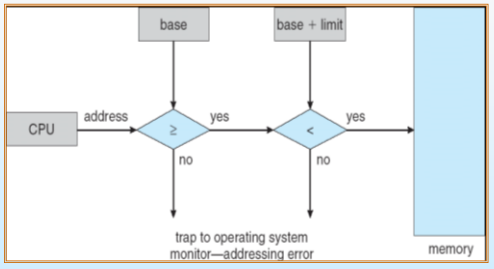
Contiguous Memory Allocation Scheme

- Main memory is usually partitioned into two:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes held in high memory
- Single-partition allocation
  - Relocation-register scheme used to protect user processes from each other, and operating-system code and its 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

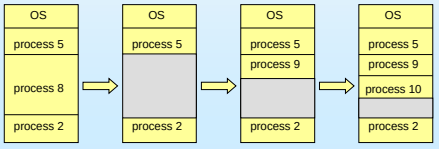


HW address protection with base and limit registers



Contiguous Allocation (Cont.)

- Multiple-partition allocation
  - *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)



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 the worst-fit in terms of speed and storage utilization

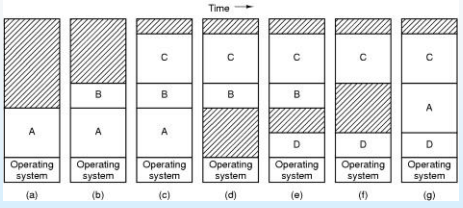
Fragmentation

- **External Fragmentation** – total memory space exists is big enough, but it is not contiguous to be allocated to any program.
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition
- 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
    - Do not relocate a program while it is involved in I/O
    - Do I/O only into OS buffers

Variable size 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)

Swapping with Variable size partitioning-1

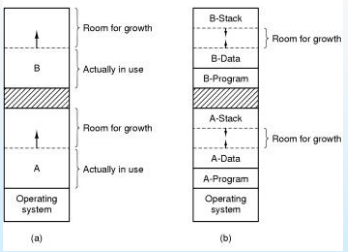


Memory allocation changes as processes

- created or moved into the memory
- terminated or moved out of the memory

(In the figures the shaded regions are unused memory)

Space allocation



- Allocating space for growing data segment
- Allocating space for growing stack & data segment

Modeling multi-Programming-1

- If p is the fraction of the time a process spends in I/O wait state, then
- CPU utilization with n processes in the system = $1 - p^n$

  ==> higher the multiprogramming ==> better is the CPU utilization.
  - Balancing of n based on p: balancing is maintained by a mix of CPU- versus I/O-intensive jobs

Memory management rules

- **Fifty percent rule**: On the average (over time), if the mean number of processes in memory is n, the mean number of holes is n/2 (adjacent holes merge, but adjacent programs do not).
- **Unused memory rule**: If k is the ratio of the average size of a hole to that of a process, then the fraction of memory occupied by holes, f = k/(k+2).

Memory management rules(cont.)

- **Computation of** f = k/(k+2), where number of processes is n and the memory size is m.
  - If s: avg. process size,
    h: avg. hole size
    ==> k=h/s, h=ks
  - Total hole memory H=h(n/2)= ks(n/2),
    also H=m-ns
    ==>ks(n/2)=m-ns,
    ==> m=ns(k/2 +1),
    from f=H/m, f= (ksn/2)/m= (ksn/2)/(ns(k/2+1)
    thus, f=k/(k+2).

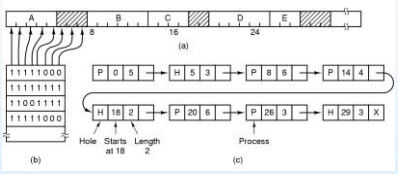
How to keep track of memory usage in variable size partition

1. Bit Maps
2. Linked Lists
3. Buddy System

Bit Maps

- Memory Management with Bit Maps
  - Divide up the memory into allocation units.
  - Corresponding to each allocation unit is a bit in the bit map that is 1/0 (allocated/free).

Bit Maps vs. linked list



a. Part of memory with 5 processes, 3 holes
   - tick marks show allocation units
   - shaded regions are free

b. Corresponding bit map

c. Same information as a list

Linked Lists-1

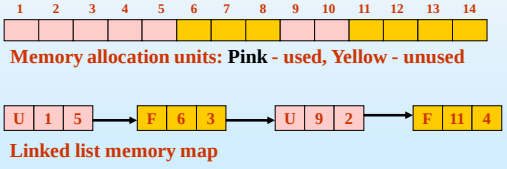
Four neighbor combinations for the terminating process X

Linked List-2

Memory Management with Linked Lists

- Just keep track of the end points of allocated and free memory segments.

Linked lists-3

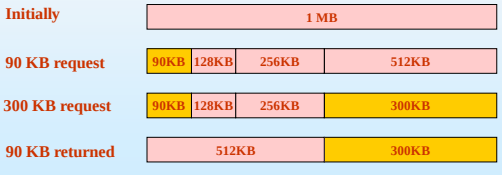


Buddy system-1

- Memory Management with Buddy System
- Deal with the memory as segments of size 2^k for some positive k.
- Buddy partition splitting or coalescing.
  - It is fast, but suffers from both internal and external fragmentation.

Buddy System-2

- Memory is allocated in powers of 2 sized units during load time



Buddy System-3

- This method makes de-allocation fast. If a block of size 2^k is released then the memory manager checks only the list of 2^k sized holes to merge them into a 2^{k+1} sized partition
- Internal fragmentation is caused since memory requests are fitted in 2^k sized partitions