Preview

- Memory Management
  - With Mono-Process
  - With Multi-Processes
  - Multi-process with Fixed partition
  - Multi-process with variable partition
  - Modeling Multiprogramming
  - Swapping
  - Free Memory Space Management
    - With Bitmap
    - With Free-List
      - Memory allocation algorithm – first, best, worst, next fit

Memory Management

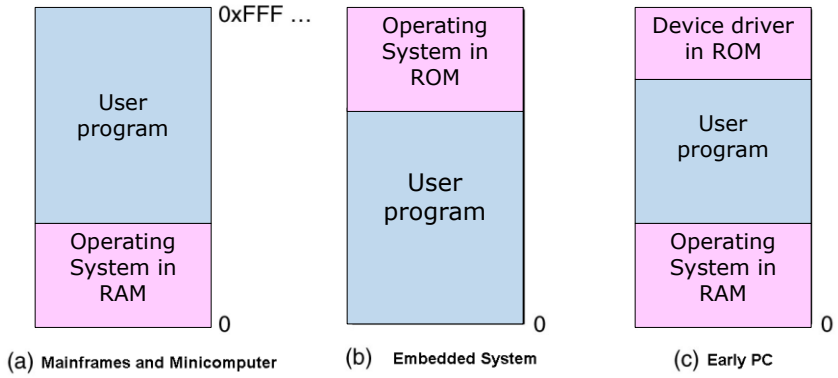
- Ideal Memory – Infinitely Large, Faster, non-volatile and inexpensive
- Since there is no such a memory, most computers has a memory hierarchy
- Memory hierarchy –
  - Small, fast, very expensive registers (volatile)
  - Expensive cache memory(volatile)
  - Megabyte or Gigabyte medium-speed, medium-speed RAM(volatile)
  - Huge size of slow cheap, disk storage, Hard Disks (non-volatile)
- Memory management is a part of operating system which manage the memory hierarchy

Memory Management (mono-process)

Mono-programming without Swapping or Paging

- There is only memory sharing between a user program and the operating system in this system.
- Only one program could be loaded in the memory.
- When the program finish its job, a scheduler (**long term scheduler**) choose one job from the pool of jobs and load into the memory and start to running.

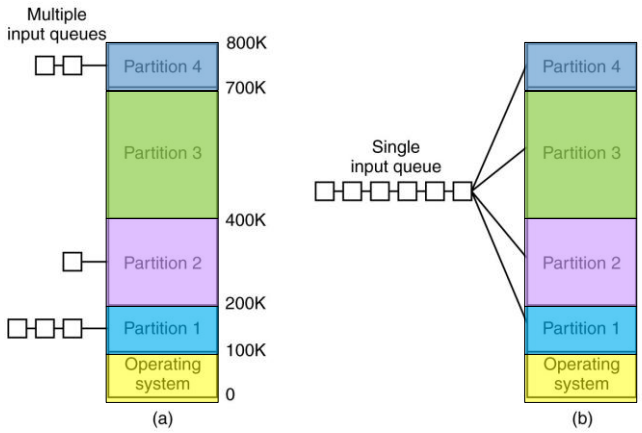
Memory Management (mono-process)



Memory Management (multi-process)

- **Multiprogramming with Fixed Partition** - Memory is divided into n partitions (possibly unequal size) – partition can be done manually when the system started.
  1. Fixed memory partition **with separate input queue** for each partition
  2. Fixed memory partition with a **single input queue.**

Memory Management
(multi-process with fixed partition)



Memory Management
(multi-process with fixed partition)

- Fixed memory partition with **separate input queue** for each partition
  - When a job arrives, it can be put into queue for the **smallest partition large enough** to hold it.
  - Disadvantage – Large partition queue is empty but small jobs are waiting on the small partition queue (see previous figure).

Memory Management
(multi-process with fixed partition)

- Fixed memory partition with a **single input queue**.
  - Whenever a partition become free, the closest to the queue that fits in it is chosen and loaded into the empty partition and run – wasting memory. Or
  - Search whole queue to find out a job which is the **best fit** to the free partition. – discriminate against small jobs.
  - Solution for discrimination.
    - One partition for small jobs
    - Count the time skipped by scheduler. If a job was skipped more than k time, it automatically granted next time.

Modeling Multiprogramming

Simple Unrealistic Model

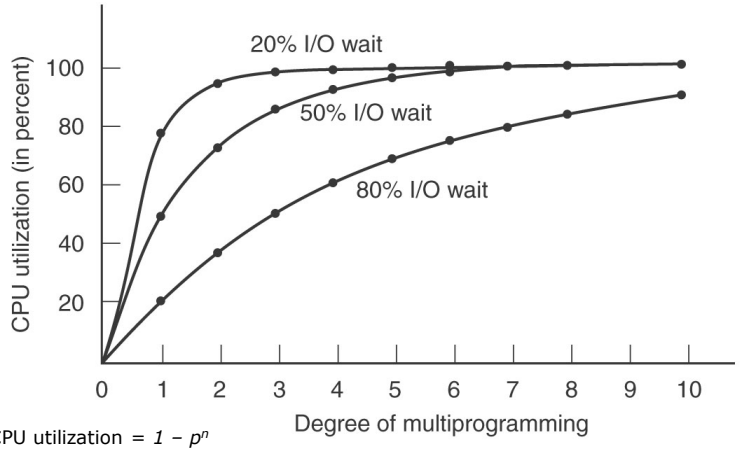
- Assumption: all five processes will never be waiting for I/O at the same time. At least one of process is in ready queue (CPU never be idle)
- If the average process use 20% (CPU) computing time and 80 % use for waiting I/O, with five processes in the memory at once (OS support multiprogramming degree = 5), the CPU should be busy 100 %

Modeling Multiprogramming

Probabilistic model

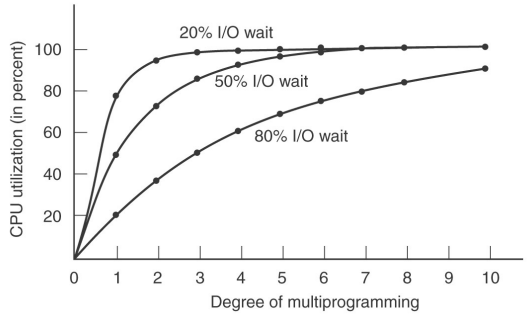
- Lets p is a fraction of time a process is waiting for I/O.
- If there are n processes (degree of multiprogramming) in the memory at once, the probability that all n processes are waiting for I/O is p^n.
- Then, the CPU utilization is given by
- CPU utilization = $1 - p^n$ (the probability that at least one of processes is using CPU)

Modeling Multiprogramming



CPU utilization = $1 - p^n$

Modeling Multiprogramming



Ex) A computer has 32MB of memory, with operating system taking up 16 MB and each program taking up 4MB.

- 4 programs to be in memory at once (4 X 4 = 16MB).
- With an p=80 % average I/O wait, we can have about 60 % (= 1- 0.8 ^{4}) of CPU utilization.
- Adding 16MB memory, (4 more programs: total 8 programs to be in the memory) we can get about 80% (= 1- 0.8 ^{8}) of CPU utilization.
- Adding yet another16MB memory, (total 12 programs to be in the memory) we can get about 92 % (= 1- 0.8 ^{12}) of CPU utilization

CPU utilization = $1 - p^n$

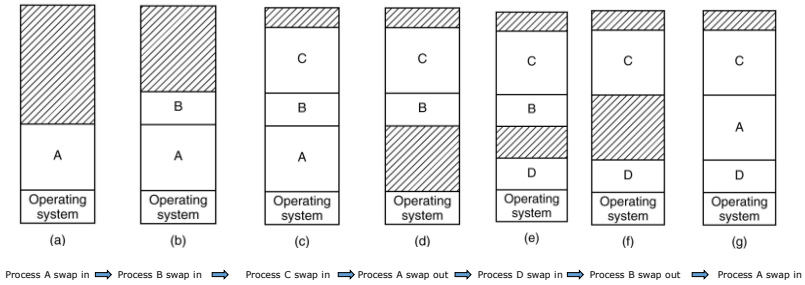
Memory Management

- Usually, there is not enough main memory to hold all the currently active processes, so excess processes must be kept on disk (or secondary memory) and brought in to run dynamically.
- Two general approached to memory management (depending on size of processes and the available hardware)
  1. **Swapping** – bringing in each process in it's entirely, running it for a while, then putting it back on the disk.
  2. **Virtual Memory** – allows programs to run with partially loaded in the memory.

Swapping

- Main difference between fixed partition and variable partition
  - Fixed partition
    - The number, location, and size of the partitions are fixed.
    - OS knows the address of each processes.
    - Simple to manage the memory.
    - Internal fragmentation-wasting space.
  - Variable partition
    - The number, location, and size of the partitions vary dynamically based on size of a process.
    - OS need to keep track of partition information dynamically for memory allocation and deallocation
    - Might create multiple holes – Combine them all into one big hole (external fragmentation: need memory compaction: expensive cycle)

Swapping with Variable Partition



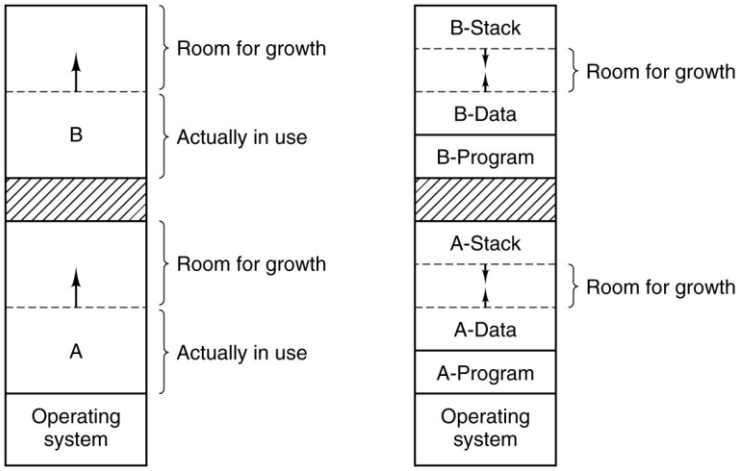
Variable partition example

Swapping

How much memory should be allocated for a process when it is created or swapped?

- If a processes are created with a fixed size – simple
- But a processes are usually change its size by dynamically allocating memory – Dynamic memory allocation (heap), recursion
  - Solution for glowing space:
    - Allocate extra memory for each process
    - Use adjacent hole for managing the growing size
- If there is not enough memory space for a process based on the dynamically changing size, the process might be swap out or killed.

Swapping



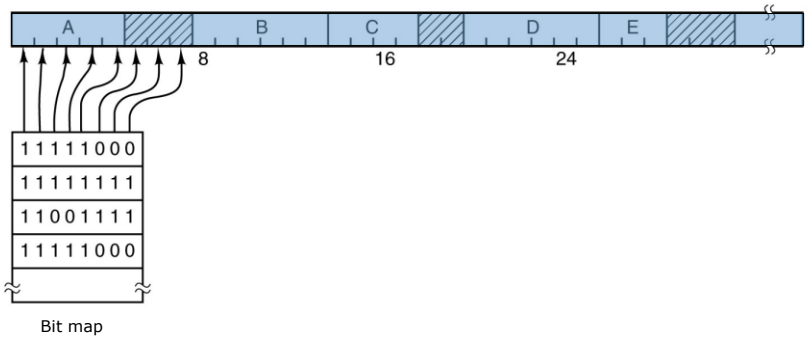
Free Memory Space Management

- OS need keep memory space information available or occupied.
- There are two possible way to keep these memory information
  - Bitmap
  - Free List

Free Memory Space Management with Bitmap

- Operating system keep a table called Bitmap to keep memory space information.
- Memory is divided up into allocation units (words ~ k bytes)
- Corresponding to each allocation unit is a bit in the bitmap
  - If the unit is free, corresponding bit = 0 and
  - If unit is occupied, corresponding bit = 1.

Free Memory Space Management with Bitmap



Bit map

Free Memory Space Management with Bitmap

- Size of the allocation unit is an important design issue
  - The smaller the allocation unit size, the larger the bitmap.
  - The larger the allocation unit size, the smaller bitmap – But memory may be wasted in the last unit of the process if the process size is not an exact multiple of the allocation unit.

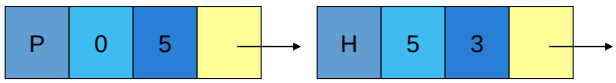
    Ex) a unit size = 10, a process size is 11, OS need assign two unit for the process. 90% of second unit are wasting space

Free Memory Space Management with Bitmap

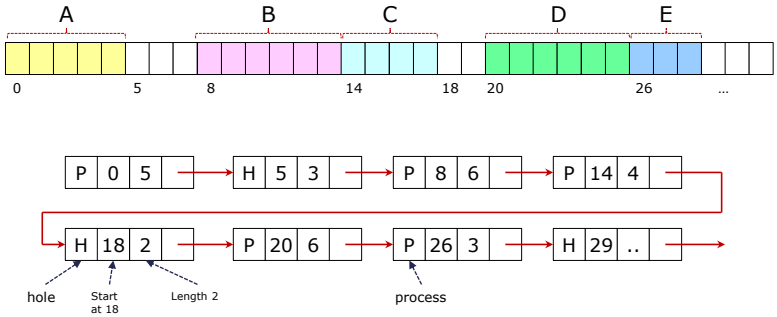
- Advantage with Bitmap
  - Simple way to keep track of memory words in a fixed amount of memory since the size of bitmap depends only on the size of memory and size of the allocation units.
- Disadvantage with Bitmap
  - To allocate memory for the process with k unit size, first, the memory manager (part of operating system) need to search the bitmap to find out k consecutive 0 bits in the map.

Free Memory Space Management with Free-list

- With free-list, Operating System maintain a linked list of allocated and free memory segments.
- Each entry in the list keeps four information: **hole or process**, **starting address**, **length and a pointer** to the next entry

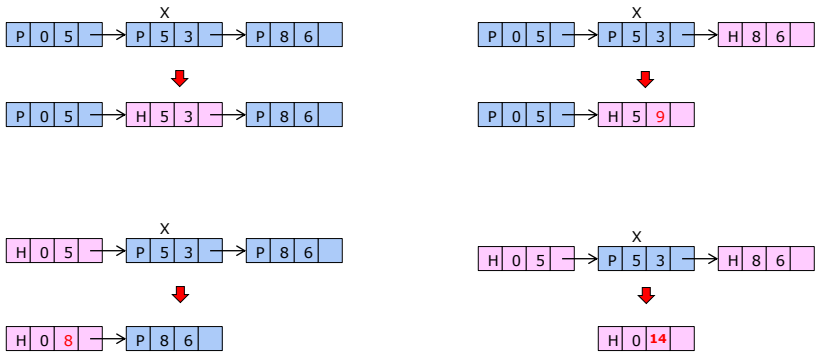


Free Memory Space Management with Free-list



Free Memory Space Management with Free-list

If the segments list is kept sorted by address, it is simple to update the list when a process termites or swapped out from memory.



Free Space management with bit map or free list

- The 128-MB memory is allocated in units of 2KB. For the linked list, let's assume that memory is currently consists of an alternating sequence of segment and holes, each 64 KB. Also assume that each node in the linked list needs a 32-bit memory address, a 16-bit length, and a 16-bit next node field.
- Bitmap:
  - #of allocation unit = 128MB/2KB = $(128 \times 2^{20})/(2 \times 2^{10}) = 2^{27}/2^{11} = 2^{16}$ units
  - Size of the bitmap = 2^{16} bits = **2^{13} byte**
- Free list:
  - number of node for linked list= 128 MB /64KB = $2^{27}/2^{16}$ or 2^{11} nodes.
  - size of each node = 32+16+16 = 64 bit = 8 byte =2^3 bytes
  - Total size of linked list = number of node × size of a node = $2^{11} \times 2^3$ bytes = **2^{14} bytes**.

Free Memory Space Management with Free-list

- Algorithms for allocating memory for a process (Assume that the segments list is kept sorted by address).
  - **First Fit** – The memory manager scans the list of segments from the beginning until it finds a hole that is big enough.
  - **Next Fit** – It works the same ways as first fit. But next time it is called, it starts searching the list from the place where it left off last time.
  - **Best Fit** – It search entire list and take the smallest hole that fit for the process.
  - **Worst Fit** – It always take the largest free hole

Memory Space Management with Free-list

- Four algorithms can be speed up by maintaining two lists: one for holes and another for processes.
  - The list of holes can be maintained with sorted by size. Best fit does not need search entire list.
  - But still need extra effort updating two lists based on the allocation and deallocation of memory.

Memory Space Management with Free-list

Quick Fit

- Maintains separate lists for holes based on the size of hole.
- It might have a table with n entries. Each entry is a pointer to the head of a certain size of holes list. Ex) First entry point to the head of list 4 KB hole, second entry point to the head of list 8 KB hole….
- Quick to finding a hole of required size
- Expensive to maintain the separate lists for hole based on the deallocation of memory