Memory Management

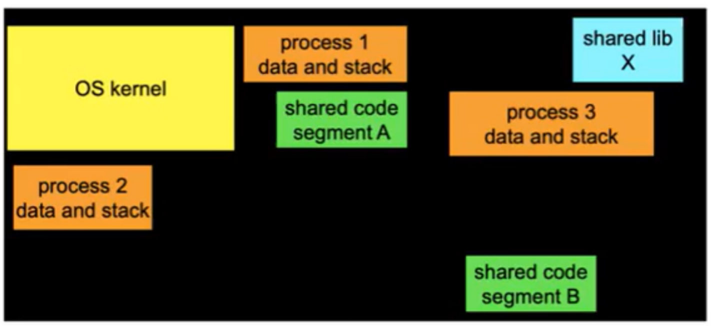
Memory Goals

- Transparency
    - Process only sees its own address space
    - Process unaware that its memory is being shared

- Efficiency
    - High effective memory utilization
    - Low run-time cost for allocation/reallocation

- Protection and isolation
    - Private data won't be corrupted
    - Private data cannot be seen by other processes

Physical Memory Allocation



Physical memory is divided between the OS kernel, process private data, and shared code segments.

- Question is how to divide the RAM

Memory Management Problem

- RAM cell has a particular physical addresses (location on a memory chip)
- Most processes can't prejudice the amount of memory they use
- Entire amount of data required by all processes could exceed the physically available memory
- Cost of memory management needs to be kept low

Fixed Partition Allocation

- Pre allocate partitions for n partitions
    - One or more per process
    - Usable only by owning process

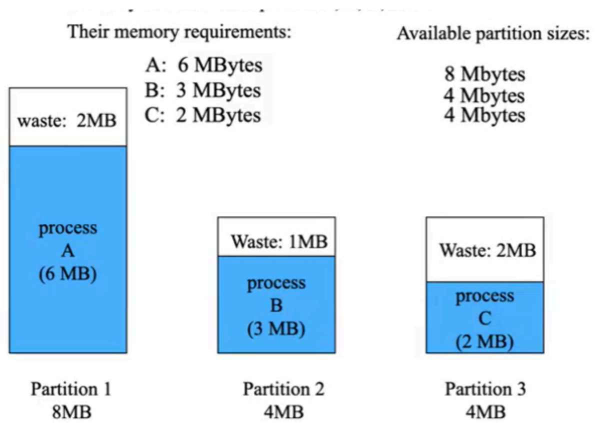
- Works for well known job mix

Memory Partitions
- Enforcing partition boundaries is need to prevent one process from accessing another's memory
- Could use hardware for this purpose
    - Special registers contain the boundaries and only accept addresses within the register values
- Basic scheme doesn't use virtual addresses

Disadvantage
- Assumes you know how much memory will be used ahead of time
- Limits the total number of processes supported to the total of theory memory requirements
- Bad for memory sharing

Internal Fragmentation
- Occurs whenever fixed sized memory is used
- Causes unallocated memory to be unusable
- Caused by inefficiency in memory allocation

    Example
    - Process A (6), B (3), C (2)
    - Available partitions: 8, 4, 4



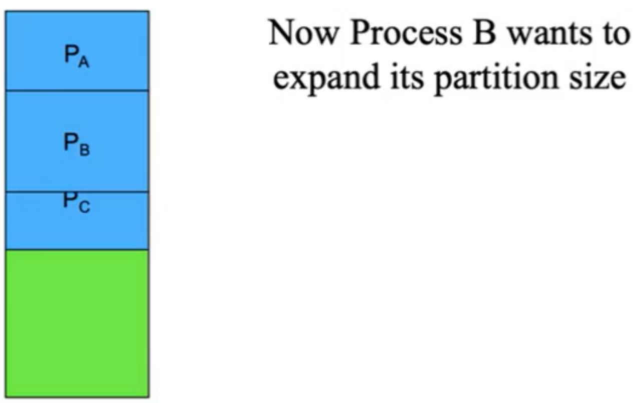
- Since all partitions have been used with remaining memory in each, there is wasted memory

Summary of Fixed
- Simple
- Inflexible
- Subject to lots of internal fragmentation
- Not used on modern systems (we don't know what our memory requirements are)

Dynamic Partition Allocation
- Variable sized partitions
- Each partition has contiguous address
- Processes have access permissions for the partitions
- Could be shared between processes

Disadvantage
- Not relocatable (once a partition is set, can't move its contents)
- Not easily expandable
- Impossible to support apps with more address space than physical memory
- Fragmentation still exists

Relocation and Expansion
- Partitions are tied to a particular address ranges
- Can't move contents to another address
  - All the pointers in the contents would be wrong
  - Don't know which memory locations contain pointers
- Hard to expand since there may not be open space nearby

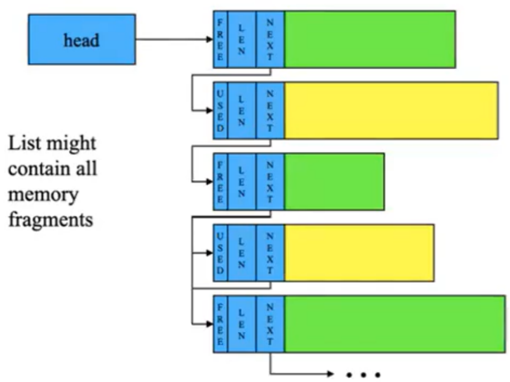


- Cannot expand since memory cannot be moved and partitions A and C prevent it from expanding locally

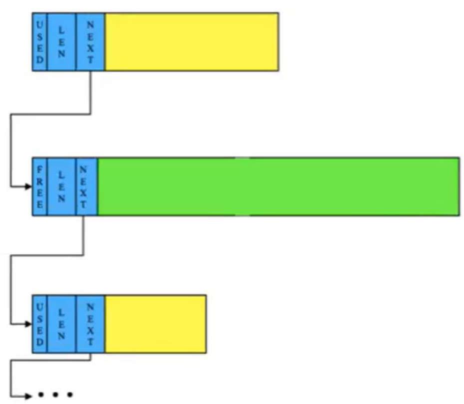
Tracking Variable Sized Partitions
1. Start with one large heap of memory
2. Maintain a free list (data structure with all pieces of unallocated memory)
3. When a process requests more memory
   - Find a large enough chunk of memory
   - Take a piece of the requested size
   - Put the remainder back on the free list
4. When a process frees memory, it goes back onto the free list

Managing the Free List
- Fixed size blocks are easy to track using a bit map of free blocks

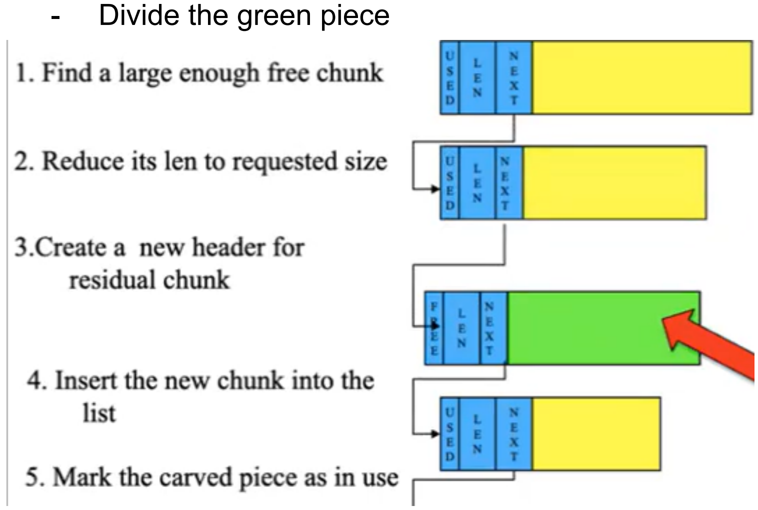
- Variable chunks
    - Linked list of descriptors, one per chunk
    - Each descriptor lists the size of the chunk and whether it is free
    - Each has a pointer to the next chunk
    - Descriptors often kept at front of each chunk (contains pointer to next chunk)



Free Chunk Carving



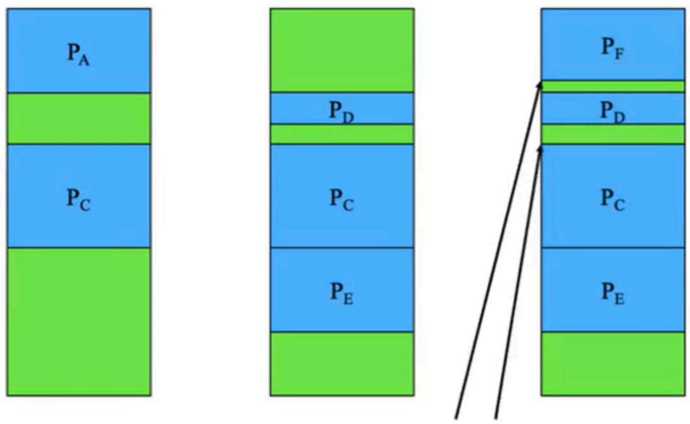
- Divide the green piece



Variable Partitions and Fragmentation

- Variable partitions not as subject to internal fragmentation
    - Unless requester asked for more than they use

External Fragmentation



We gradually build up small, unusable memory chunks scattered through memory

- As continue to allocate chunks, we are left with chunks that are too small for anyone to use
- Solution: Don't create fragments or turn these small chunks into a larger chunk

Avoiding Fragmentation

Best Fit

- Search for the best fit chunk
    - Smallest size greater than or equal to the requested size

- Advantage:
    - Might find the perfect fit

- Disadvantage:
    - Have to search the entire list every time
    - Creates very small fragments

Worst Fit

- Largest size greater than or equal to the requested size

- Advantage:
    - Creates very large fragments for some time
    - Spreads out fragmentation

- Disadvantage:
    - Need to search the entire list every time

First Fit

- Take the first chunk you find that is big enough

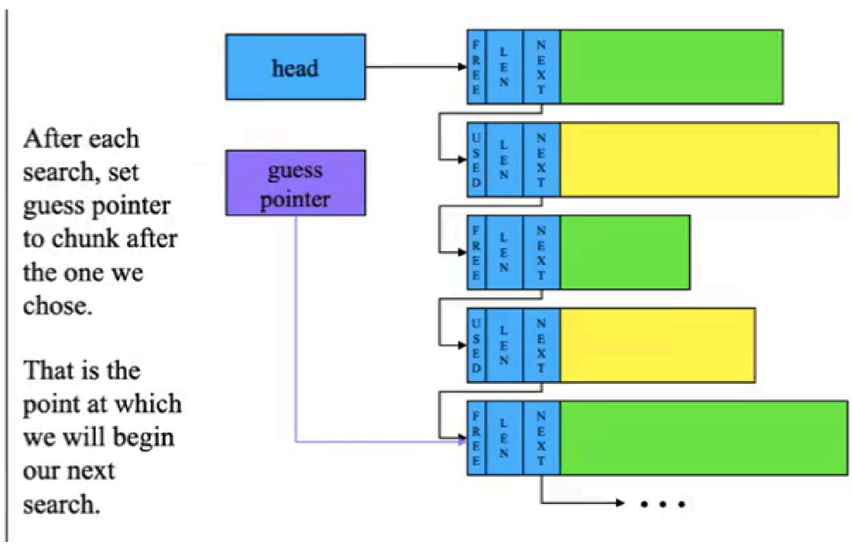
- Advantages
    - Very quick searches
    - Creates random sized fragments

- Disadvantages
    - First chunks quickly fragment, resulting in longer searches
    - Fragments just as much as best fit
    - Results in large number of fragments close to the start

Next Fit

- Tries both first and worst fit
    - Short searches
    - Spreads out fragmentation (like worst fit)

- Guess pointer
    - Set to the chunk after the one we choose
    - If it points to memory that is large enough, then it saves time
    - If it's wrong, algorithm still works



Combining Fragments
- All variable sized partition allocation algorithms have external fragmentation
- Reassemble fragments
  1. Check neighbors whenever a chunk is fixed
  2. Recombine free neighbors when possible (adjacent addresses)
  3. Free list can be ordered by chunk address
- Counters forces of external fragmentation

Issues with Coalescing
- Coalescing works better with more free space
- Chunks which are being used for a long time cannot be coalesced
- If the requested chunk size continues to change, fragmentation increases
- Fragmentation gets worse as time goes on

Variable Sized Partition Summary
- Eliminates internal fragmentation
  - Each chunk is custom carved

- Expensive implementation
  - long search times for free lists
  - carving and coalescing takes time

- External fragmentation is inevitable
  - Coalescing could counteract fragmentation

Special Case for Fixed Allocations
- There might be frequent requests to blocks of a fixed size (some sizes or requested more often than other)
- Fixed size buffers are frequently used
  - File systems, network protocols

- Account for transient use

Buffer Pools
- Popular sizes are used to from special pools of fixed size buffers
- Benefit improved efficiency
  - Simpler than variable partition allocation
  - Eliminates searching, carving, and coalescing
- Reduces external fragmentation
- Must know exactly the amount to reserve
  - Will result in internal fragmentation if done improperly
  - Can’t ask for 3k block from a 4k pool
- Once all buffers used, then returns back to normal allocation
- If too many buffers made, then memory is wasted

Using Buffer Pools

1. Process requests and uses one element from the buffer pool
2. Process frees the memory
3. Memory returns to the pool

Buffer Pool Size

- Resize it automatically

- If we run low on fixed size buffers
    - Get more memory from the free list
    - Split into more fixed size buffers

- If free buffer gets too large
    - Return some buffer to the free list

- If the free list gets low
    - Ask each major service with a buffer pool to return space

- Parameters used to determine the state of the buffer pool
    - Low space threshold
    - High space threshold
    - Normal allocation

Lost Memory

- Process is done with the buffer but doesn't free it
- If a process allocates a large area and maintains its own free space

Garbage Collection

- Searches data space and finds every object pointer and finds the address and sizes of all accessible objects
- The complement of this is what is inaccessible, which is then returned to free list

Object Oriented GC

- Object references are tagged
- Object descriptors include size information
- Have a list of all objects

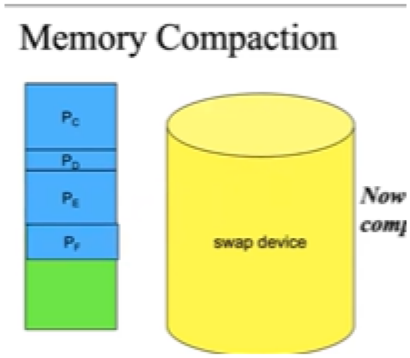
General GC

- Find all pointers in allocated memory
- Determine the amount of memory which is pointed to
- Determine what is being used

Memory Compaction

- Relocating free memory for coalescing and recombining fragments

1. Shift all processes onto a swap device
2. Move the process into consecutive memory addresses, leaving a much larger space of free memory

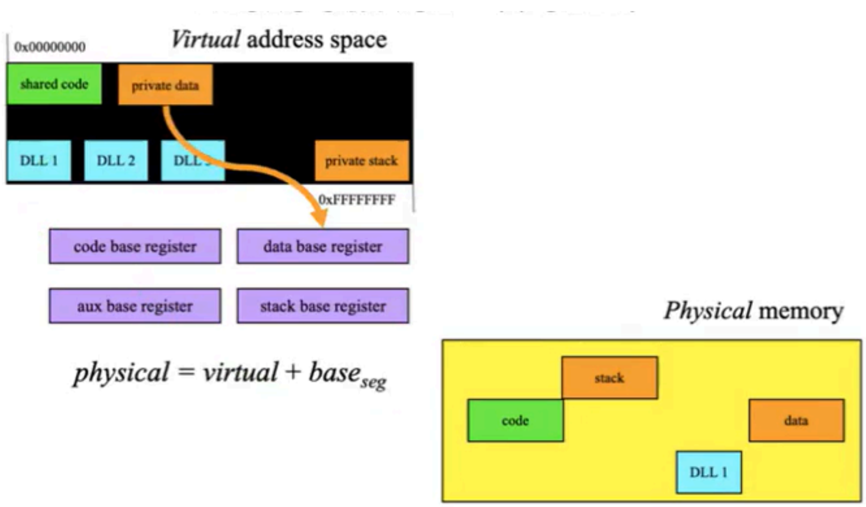


Relocation Problem

- Cannot relocate by purely manipulating the physical memory
- Process location and the physical location must be made independent

Memory Segment Relocation

- Process address space is made of multiple segments, which can be used as a unit of relocation
- Computer has special relocation registers called segment base registers
- OS uses these registers to perform virtual address translation
  - Set the base register to the start of region where the program is loaded
  - If the program is moved, reset the base registers to new location



Protection

- Prevent process from reaching outside of its allocated memory
- Seg fault

Segments

- Variable sized partitions
- Makes coalescing better
- Need to make these segments contiguous
- Removing the requirement of contiguous segments is done using virtual addresses