Memory Managment

Memory Hierarchy

- what is the memory hierarchy?
    - different levels of memory
    - some are small and fast
    - others are large and sloe
- What levels are usually included?
    - cache:small amount of fast expensive memory
        - L1 cache: usually on the CPU chip
        - L2: may be on or off chop
        - L3 cache: off-chip, made of SRAM

Basic Memory management:

- components include:
    - OS
    - single process
- Goal: lay these out in memory
    - memory protection may not be an issue
    - flexibility may still be useful
- No swapping or paging

Fixed partitions: multiple programs:

- fixed memory partitions
    - divide memory into fixed spaces
    - assign a process to a space when its free
- Mechanisms
    - separate input queues for each partition
    - single input queue

How many processes are enough?

- several memory partitions
- lots of processes wanting to use the CPU
- Tradeoff:
    - more processes utilize the CPU better
    - fewer processes use less memory

Modeling multiprogramming

- More I/O wait means less processor utilization
- This means that the OS should have more processes if they're I/O bound
- More processes -> memory management and protection

Multiprogrammed system performance

- More I/O wait means less processor utilization
- This means that the OS should have more processes if they're I/O bound
- More processes -> memory management and protection...

Memory and multiprogramming

- memory needs two things for multiprogramming
    - relocation
    - protection
- The OS cannot be certain where a program will be loaded in memory
    - variables and procedures can't use absolute locations in memory
    - several ways to guarantee this
- The OS must keep processes memory separate
    - protect a process from other processes reading or modifying its own memory
    - protect a process from modifying its own memory in undesirable ways (such as writing to program code)

Base and limit registers

- special CPU registers: base and limit
    - access to the registers limited to system mode
    - registers contain:
        - base: start of the processes memory
        - limit: length go the processes memory partition
- Address generation
    - physical address: location in actual memory
    - logical address: location from the process's point of view
    - physical address = base + logical address
    - logical address..

Swapping:

- memory allocation changes as

    - processes come into memory
    - processes leave memory
        - swapped to disk
        - complete execution
- Gray regions are unused memory

Swapping: leaving room to grow:

- need to allow for programs to grow
    - allocate more memory for data
    - larger stack
- Handled by allocating more space than is necessary at the start
    - inefficient:wastes memory that's not currently in use
    - what if the process requests too much memory?

Tracking memory usage:

- operating system needs to track allocation state of memory
    - regions that are available to hand out
    - regions that are in use...

Tracking memory usage: bitmaps

- keep track of free/allocated memory regions with a bitmap
    - one bit in map corresponds to a fixed-size region of memory
    - bitmap is a constant size for a given amount of memory regardless of how much is allocated at a particular time
- Chunk size determines efficiency
    - at 1 bit per 4KB chunk, we need just 256 buts per MB od memory
    - for smaller chunks we need more memory for the bitmap

Tracking memory: linked lists

- each entry in the list corresponds to a contiguous region of memory
- entry can indicate either allocated or free
- may have separate lists for free and allocated areas
- efficient if chunks are large
    - fixed wise representation for each region
    - more regions == more space needed for free lists

Allocating memory

- search through region list to find a large enough space
- Suppose there are several choices: which one to use?
    - First fit: the first suitable hole on the list
    - Next fit: the first suitable after the previously allocated hole
    - best fit: the smallest hole that is larger than the desired region
    - worst fit: the largest available hole
- option: maintain separate queues for different size holes

Freeing memory

- allocation structures must be updates when memory is freed
- easy with bitmaps: just set the appropriate bits in the bitmap
- linked lists: modify adjacent elements as needed
    - merge adjacent free regions into a single region
    - may involve merging two regions with the just-freed area

Buddy allocations

- allocate memory in powers of two
    - good for objects of varied sizes
- spilt larger chunks to create two smaller chunks
- when chunk is freed, see if it can be combined with its buddy to rebuild a larger chunk
    - this is recursive

Slab allocation

- use slabs (each 1 + page long) to allocate and manage a large number of equal-sized objects
    - avoid internal fragmentation
    - better performance: faster allocation / deallocation
- carve slabs into objects
    - free a slab only when all of its objects are freed
    - keep track of free objects and hand out again when needed

Limitations of swapping

- problems with swapping
    - process must fit into physical memory (impossible to run larger processes)

        - Memory becomes fragemented
            - external fragmentations:: lots of small free areas
            - compaction needed to reassemble larger free areas
        - processes are either in memory or on disk: half and half doesn't do any good
- overlays solved the first problem
    - bring in pieces of the process over time
    - still doesn't solve the problem of fragmentations or partially resident processes

Virtual Memory

- basic idea: allow the OS to hand out more memory than exists on the system
- keep recently used stuff in physical memory
- move less recently used stuff to disk
- keep all of this hidden from processes

Virtual and Physical addresses

- program uses virtual addresses
    - addresses local to the process
    - hardware translates virtual addresses to physical addresses
- Translation done by the Memory Management Unit
    - usually on the same chip as the CPU
    - Only physical addresses leave the CPU/MMU chip
- Physical memory indexed by physical addresses