Memory Management

Today
- Basic memory management
- Swapping
- Kernel memory allocation

Next Time
- Virtual memory
Memory management

- Ideal memory for a programmer
  - Large
  - Fast
  - Non volatile
  - Cheap

- Nothing like that → memory hierarchy
  - Small amount of fast, expensive memory – cache
  - Some medium-speed, medium price main memory
  - Gigabytes of slow, cheap disk storage

- Memory manager handles the memory hierarchy
Basic memory management

- Simplest memory abstraction – no abstraction at all
  - Early mainframes (before ‘60), minicomputers (before ‘70) and PCs (before ‘80)
  - MOV REG1, 1000 moves content of physical memory 1000 to register 1
  - Logically, only one program running at a time Why?
  - Still here, some alternatives for organizing memory
Multiprogramming w/ fixed partitions

- With a bit of hardware – Multiprogramming – while one process waits for I/O, another one can use the CPU.
- Two simple approaches
  - Split memory in \( n \) parts (possible \(!=\) sizes)
  - Single or separate input queues for each partition
  - ~IBM OS/360 – MFT: Multiprogramming with Fixed number of Tasks

![Diagram of memory partitioning](image-url)
Two problems w/ multiprogramming

- Protection and relocation
  - Keep a process out of other processes’ partitions
    - IBM OS/MFT - modify instructions on the fly
      - Split memory into 2KB blocks
      - Add key/code combination (4 bit)
      - The PSW kept the key
    - Don’t know where a program will be loaded in memory
      - Address locations of variables & code routines
      - IBM 360 – modify program at loading time (static relocation)

- A new abstraction: Address space
  - Address space – the set of addresses a process can use to address memory
    - Each process has its own address space

- Other examples of address spaces
  - Phones, IP addresses, .com Internet domains
Two problems w/ multiprogramming

- Use base and limit values (CDC 6600 & Intel 8088)
  - Address locations + base value $\rightarrow$ physical address
  - Ideally, the base and limit registers can only be modified by the OS
  - A disadvantage – Comparisons can be done fast but additions can be expensive

![Diagram of address checking](attachment:address_checking_diagram.png)
Not enough memory for all processes?

- **Swapping**
  - Simplest
  - Bring each process entirely
  - Move another one to disk
  - Compatible Time Sharing System (CTSS) – a uniprogrammed swapping system

- Virtual memory (your other option)
  - Allow processes to be only partially in main memory
Swapping

- How is different from MFT?
  - Much more flexible
    - Size & number of partitions changes dynamically
  - Higher memory utilization, but harder memory management

- Swapping in/out creates multiple holes
  - Fragmentation …

Space for A is available, but not as a single piece.
Fragmentation

- External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous

- Reduce external fragmentation by compaction
  - Shuffle contents to group free memory as one block
  - Possible only if relocation is dynamic; done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers

- Too expensive (1GB machine that can copy at 4B/20nsec will take 5 sec to compact memory!)
How much memory to allocate?

- If process’ memory doesn’t grow – easy
- In real world, memory needs change dynamically:
  - Swapping to make space?
  - Allocate more space to start with
    - Internal Fragmentation – leftover memory is internal to a partition
  - Remember what you used when swapping
- More than one growing area per processes
  - Stack & data segment
  - If need more, same as before
Memory management

- With dynamically allocated memory
  - OS must keep track of allocated/free memory
  - Two general approaches - bit maps and linked lists

- Bit maps
  - Divide memory into allocation units
  - For each unit, a bit in the bitmap
  - Design issues - Size of allocation unit
    - The smaller the size, the larger the bitmap
    - The larger the size, the bigger the waste
  - Simple, but slow
    - find a big enough chunk?

Memory management with lists

- Linked list of allocated/free space
- List ordered by address
- Double link will make your life easier
  - Updating when a process is swapped out or terminates

Keeping track of processes and holes in the same list
Picking a place – different algorithms

If list of processes & holes is ordered by addresses, different ways to get memory for a new processes …

– First fit – simple and fast
– Next fit - ~ First fit but start where it left off
  • Slightly worst performance than First fit
– Best fit – try to waste the least but …
  • More wasted in tiny holes!
– Worst fit – try to “waste” the most (easier to reuse)
  • Not too good either

– Speeding things up
  • Two lists (free and allocated) – slows down deallocation
  • Order the hole list – first fit ~ best fit
  • Use the same holes to keep the list
  • Quick fit – list of commonly used hole sizes
    N lists for N different common sizes (4KB, 8KB, …)
    Allocation is quick, merging is expensive
Kernel memory allocation

- Most OS manage memory as set of fixed-size pages
- Kernel maintains a list of free pages
- Page-level allocator has
  - Two main routines: e.g. `get_page()` & `freepage()` in SVR4
  - Two main clients: Paging system & KMA

Provides odd-size buffers to various kernel subsystems

Diagram:

- Kernel memory allocator
- Page-level allocator
- Paging system

Network Buffers
Proc structures
inodes, file descriptors
User processes
Block buffer cache
Kernel memory allocation

- KMA’s common users
  - The pathname translation routine
  - Proc structures, vnodes, file descriptor blocks, ...
- Since requests $\ll$ page $\rightarrow$ page-level allocator is inappropriate
- KMA & the page-level allocator
  - Pre-allocates part of memory for the KMA
  - Allow KMA to request memory
  - Allow two-way exchange with the paging system
- Evaluation criteria
  - Utilization memory – physical memory is limited after all
  - Speed – it is used by various kernel subsystems
  - Simple API
  - Allow a two-way exchange with page-level allocator
KMA – Resource map allocator

- Resource map – a set of <base, size> pairs
- Initially the pool is described by a single pair
- ... after a few exchanges ... a list of entries per contiguous free regions
- Allocate requests based on
  - First fit, Best fit, Worst fit
- A simple interface

  ```
  offset_t rmalloc(size);
  void rmfree(base, size);
  ```

```
256,128
576,448

rmalloc(256)  rmalloc(320)

rmfree(256,128)
```
Resource map allocator

**Pros**
- Easy to implement
- Not restricted to memory allocation
- It avoid waste (although normally rounds up requests sizes for simplicity)
- Client can release any part of the region
- Allocator coalesces adjacent free regions

**Cons**
- After a while maps ended up fragmented – low utilization
- Higher fragmentation, longer map
- Map may need an allocator for its own entries
  - *How would you implement it?*
- To coalesce regions, keep map sorted – expensive
- Linear search to find a free region large enough
KMA – Simple power-of-two free list

- A set of free lists
- Each list keeps free buffers of a particular size ($2^x$)
- Each buffer has one word header
  - Pointer to next free buffer, if free or to
  - Pointer to free list (or size), if allocated
KMA – Simple power-of-two free list

- Allocating(size)
  - allocating (size + header) rounded up to next power of two
  - Return pointer to first byte after header
- Freeing doesn’t require size as argument
  - Move pointer back header-size to access header
  - Put buffer in list
- Initialize allocator by preallocating buffers or get pages on demand; if it needs a buffer from an empty list …
  - Block request until a buffer is released
  - Satisfy request with a bigger buffer if available
  - Get a new page from page allocator
Power-of-two free lists

- **Pros**
  - Simple and pretty fast (avoids linear search)
  - Familiar programming interface (malloc, free)
  - Free does not require size; easier to program with

- **Cons**
  - Rounding means internal fragmentation
  - As many requests are power of two and we loose header; a lot of waste
  - No way to coalesce free buffers to get a bigger one
  - Rounding up may be a costly operation
Coming up …

- The nitty-gritty details of virtual memory …
- But first, this time for real, the midterm!