CS 3204
Operating Systems

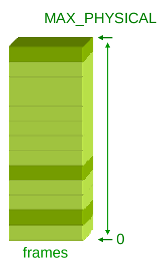
Lecture 18

Godmar Back

Physical Memory Management

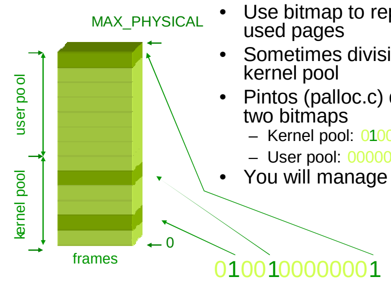
Physical Memory Management

- Aka frame table management
- Task: keep efficiently track of which physical frames are used
- Allocate a frame when paging in, or eager loading
- Deallocate a frame when process exits or when page is evicted (later)



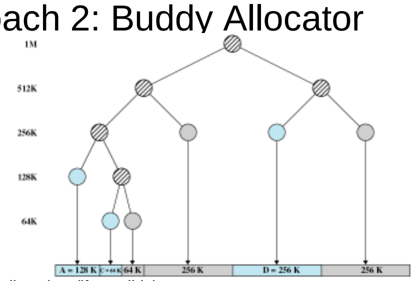
Approach 1: Bitmaps

- Use bitmap to represent free, used pages
- Sometimes division in user & kernel pool
- Pintos (palloc.c) does that: keeps two bitmaps
  - Kernel pool: 010010
  - User pool: 0000001
- You will manage user pool only



Approach 2: Buddy Allocator

- Logically subdivide memory in power-of-two blocks
- Round up on allocation to next power of 2
- Split block on allocation (if necessary)

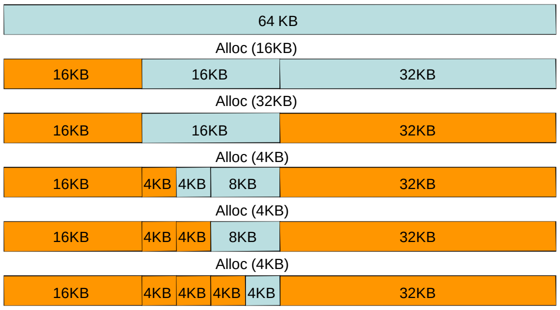


- Coalesce on deallocation (if possible)
  - Coalescing can be delayed
- Used in Linux: allocation requests are always multiple of pages, max blocksize is 4MB

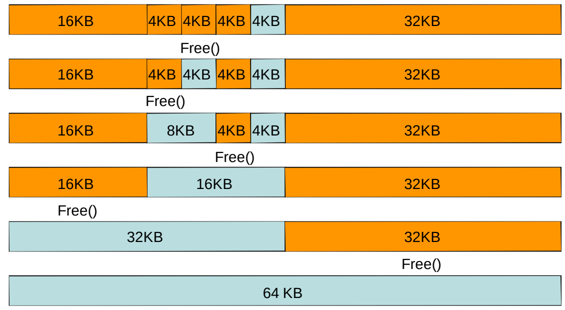
Buddy Allocator Freelists

- Note: tree view is conceptual, not tree is actually built in implementation
- Implementation uses n freelists for free blocks of order k, representing allocation sizes of 2^(k+B) bytes.
  - E.g., Linux uses orders 0-10 for 4KB, 8KB, … 4MB or 2^(0+12) … 2^(10+12) bytes.
  - Note that free list is kept entirely inside unused pages – unlike bitmaps, no additional memory is needed

Buddy Example - Allocation



Buddy Example - Deallocation

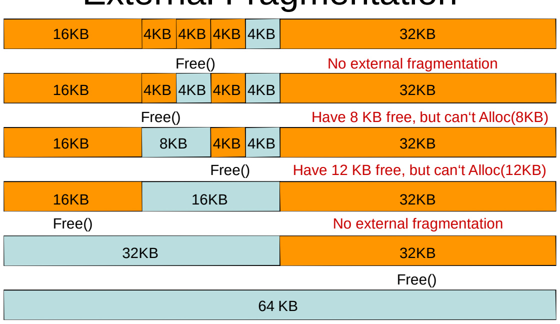


Fragmentation

- Def: *The inability to use memory that is unused.*
- Internal fragmentation:
  - Not all memory inside an allocated unit is used; rest can't be allocated to other users
- External fragmentation:
  - Impossible to satisfy allocation request even though total amount of memory > size requested

Internal Fragmentation

External Fragmentation



Buddy Allocator & Fragmentation

- Q.: what is the average internal fragmentation (per allocated object) for
  - buddy allocator with size 2^n?
  - in bitmap allocator for objects of size n*s, where each bit represents a unit of size s?
  - in first-fit allocator from project 0?
- Q.: what external fragmentation can you expect from buddy allocator scheme?
- Q.: what's a good way to measure fragmentation in general?

Page Size & Fragmentation

- How should a system's architect choose the page size? – Trade-Off
- Large pages:
  - Larger internal fragmentation
  - (not an issue if most pages are full…)
  - Page-in & write-back cost larger
- Small pages:
  - Higher overhead to store page table (more entries to maintain)
- Modern architectures provide support for "super pages" – 2MB or 4MB

Page Replacement

Page Replacement Policies

- Goal: want to minimize number of (major) page faults (situations where a page must be brought in from disk.)
  - Also: want to reduce their cost (ideally, evict those pages from their frames that are already on disk – save writeback time)
- Possible scopes in which replacement is done:
  - Global replacement policies
    - Treat frames used by all processes equally
  - Local replacement polices
    - Pool frames according to user or process when considering replacement
  - Hybrids
- Algorithms can be applied in these scopes

Replacement Algorithms

- Optimal:
  - "know the future"
  - Obviously impractical, just a benchmark for comparison/analysis
- FIFO – evict oldest page
- LRU – evict least recently used page
- Clock algorithm ("NRU")
  - Enhanced versions of clock

Optimal or MIN Replacement

- To analyze algorithms, consider stream of accesses; each access falls into a given page, e.g.
  2 3 2 1 5 2 4 5 3 2 5 2
- Optimal (also known as MIN, or Belady's algorithm)
  - Replace the page that is accessed the farthest in the future, e.g. that won't be accessed for the longest time
- Problem: don't know what the future holds

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	4	4	4	2	2	2
	3	3	3	3	3	3	3	3	3	3	3
			1	5	5	5	5	5	5	5	5

FIFO

- Evict oldest page:
  - Problem: completely ignores usage pattern – first pages loaded are often frequently accessed

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	5	5	5	5	3	3	3	3
	3	3	3	3	2	2	2	2	2	5	5
			1	1	1	4	4	4	4	4	2

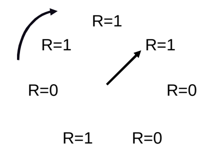
LRU

- Evict least-recently-used page
- Great if past = future: becomes MIN!
- Major problem: would have to keep track of “recency” on every access, either by timestamping, or move to front of a list
  - Infeasible to do that at pipeline speed

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	2	2	3	3	3	3
	3	3	3	5	5	5	5	5	5	5	5
			1	1	1	4	4	4	2	2	2

Clock

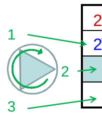
- Also known as NRU (Not Recently Used) or 2nd Chance
- Use access (or reference bit)
  - R=1 was accessed
  - R=0 was not accessed
- Hand moves & clears R
- Hand stops when it finds R==0
- Two ways to look at it:
  - Approximation of LRU
  - FIFO, but keep recently used pages



Clock Example

- In this example, assume hand advances only when allocation requires eviction (as you can do for Pintos P3)
- To avoid running out of frames, one could use clock daemon that periodically scans pages and resets their access bits (done in Solaris)
  - Q.: what if clock daemon scans too fast? – all pages appear unused
  - Q.: what if too slow? – many pages appear used
- Or start scanning when #free pages falls below some watermark (as done by kswapd in Linux)

* means R=1 (page was accessed since last scan)



2	3	2	1	5	2	4	5	3	2	5	2
2*	2*	2*	2*	5*	5*	5*	5*	3*	3*	3*	3*
	3*	3*	3*	3	2*	2*	2*	2	2*	2	2*
			1*	1	1	4*	4*	4	4	5*	5*

Variations on Clock Algorithm

- 2-handed Clock
  - If lots of frames, may need to scan many page frames until one is found – so introduce second hand
    - Leading hand clears ref bits
    - Trailing hand evicts pages
- Enhanced Clock: exploit modified (or “dirty”) bit
  - First find unreferenced & unmodified frames to evict
  - Only if out of those, consider unreferenced & modified frames
  - Clear reference bit as usual

N-bit Clock Algorithm

- 1-bit says was recently used or wasn’t
  - But how recently?
- Idea: associate n-bit counter with page frame
  - “age” or “act_count”
  - have R-bit as before (provided by hardware)
- When hand passes page frame
  - act_count >>= 1 aging
  - act_count |= (R << (n-1)) recent access
- Replace page frame with lowest act_count