CSCI315 – Operating Systems Design

Department of Computer Science
Bucknell University

Other Issues in Virtual Memory Examples

Ch 10.5, 10.8. 10.9

This set of notes is based on notes from the textbook authors, as well as L. Felipe Perrone, Joshua Stough, and other instructors.
Xiannong Meng, Fall 2021.

Overview

- Reclaiming free memory
- Virtual memory for kernel process(es)
- Virtual memory examples: Linux and Windows

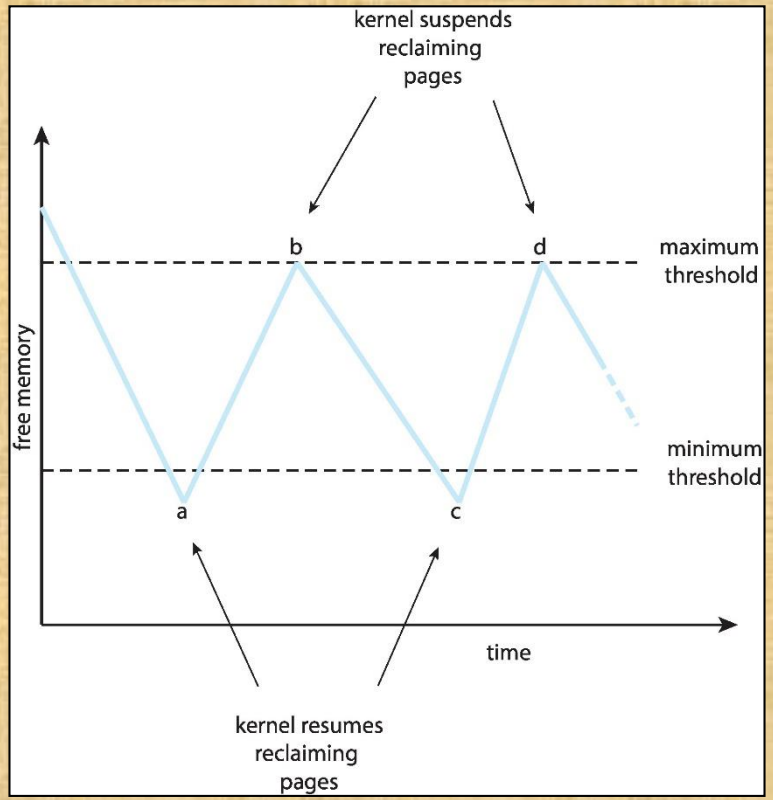
Reclaiming Free Memory

- The issue of reclaiming free memory:
  - If we wait until a major page fault, it is too costly!
  - The MMU tries to avoid as much as possible any major faults. One practical solution is to reclaim free memory when possible.
- In a global page-replacement policy, the MMU periodically reclaims free frames from all processes under certain given condition(s).

Reclaiming Pages

- A strategy to implement global page-replacement policy
  - All memory requests are serviced by the global free-frame list.
  - Rather than waiting for the free list to drop to zero before we begin selecting pages for replacement, page replacement is triggered when the list falls below a certain threshold.
- This strategy attempts to ensure there is always sufficient free memory to satisfy new requests.

Reclaiming Pages Example



Allocating Kernel Memory

- Algorithms so far are applied to user processes. Kernel processes need special consideration.
- Often allocated from a separate free-memory pool
  - Kernel requests memory for structures of varying sizes
  - Some kernel memory needs to be contiguous
    - i.e., for device I/O

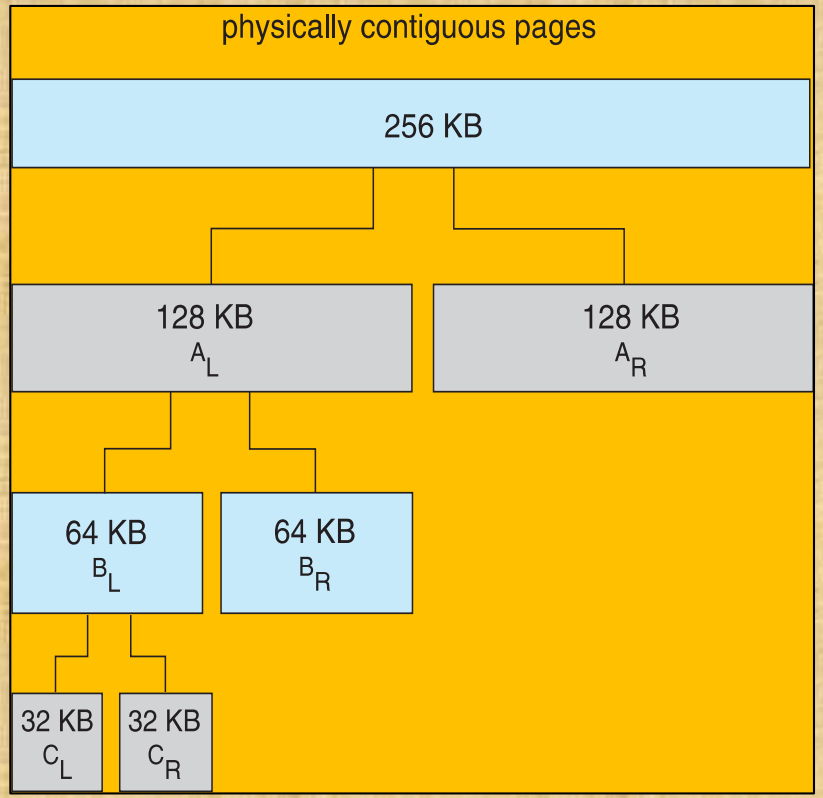
Buddy System

- Allocates memory from fixed-size segment consisting of physically-contiguous pages
- Memory allocated using **power-of-2 allocator**
  - Satisfies requests in units sized as power of 2
  - Request rounded up to next highest power of 2
  - When smaller allocation needed than is available, current chunk split into two buddies of next-lower power of 2
    - Continue until appropriate sized chunk available

Buddy System Example

- For example, assume 256KB chunk available, kernel requests 21KB
  - Split into A_L and A_R of 128KB each
    - One further divided into B_L and B_R of 64KB
      - One further into C_L and C_R of 32KB each – one used to satisfy the given request
- Advantage – quickly **coalesce** unused chunks into larger chunk
- Disadvantage - fragmentation

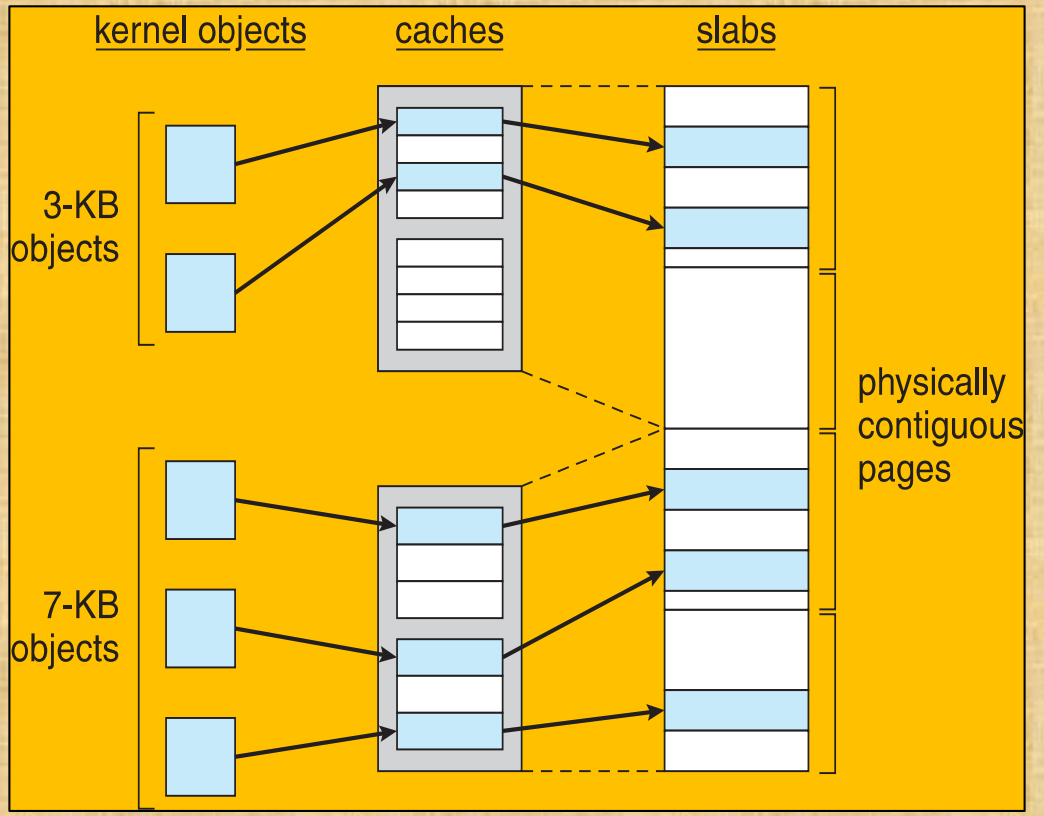
Buddy System Allocator



Slab Allocator

- Alternate strategy
- **Slab** is one or more physically contiguous pages
- **Cache** consists of one or more slabs
- Single cache for each unique kernel data structure
  - Each cache filled with **objects** – instantiations of the data structure
- When cache created, filled with objects marked as **free**
- When structures stored, objects marked as **used**
- If slab is full of used objects, next object allocated from empty slab
  - If no empty slabs, new slab allocated
- Benefits include no fragmentation, fast memory request satisfaction

Slab Allocation



Slab Allocator in Linux

- For example process descriptor is of type `struct task_struct`
- Approx 1.7KB of memory
- New task -> allocate new struct from cache
  - Will use existing free `struct task_struct`
- Slab can be in three possible states
  1. Full – all used
  2. Empty – all free
  3. Partial – mix of free and used
- Upon request, slab allocator
  1. Uses free struct in partial slab
  2. If none, takes one from empty slab
  3. If no empty slab, create new empty

Slab Allocator in Linux (Cont.)

- Slab started in Solaris, now wide-spread for both kernel mode and user memory in various OSes
- Linux 2.2 had SLAB, now has both SLOB and SLUB allocators
  - SLOB for systems with limited memory
    - **S**imple **L**ist **O**f **B**locks – maintains 3 list objects for small, medium, large objects
  - SLUB is performance-optimized SLAB removes per-CPU queues, metadata stored in page structure
    - **SLUB**: Unqueued SLAB allocator

https://events.static.linuxfound.org/sites/events/files/slides/slaballocators.pdf

https://lwn.net/Articles/229096/

VIRTUAL MEMORY SYSTEM EXAMPLES

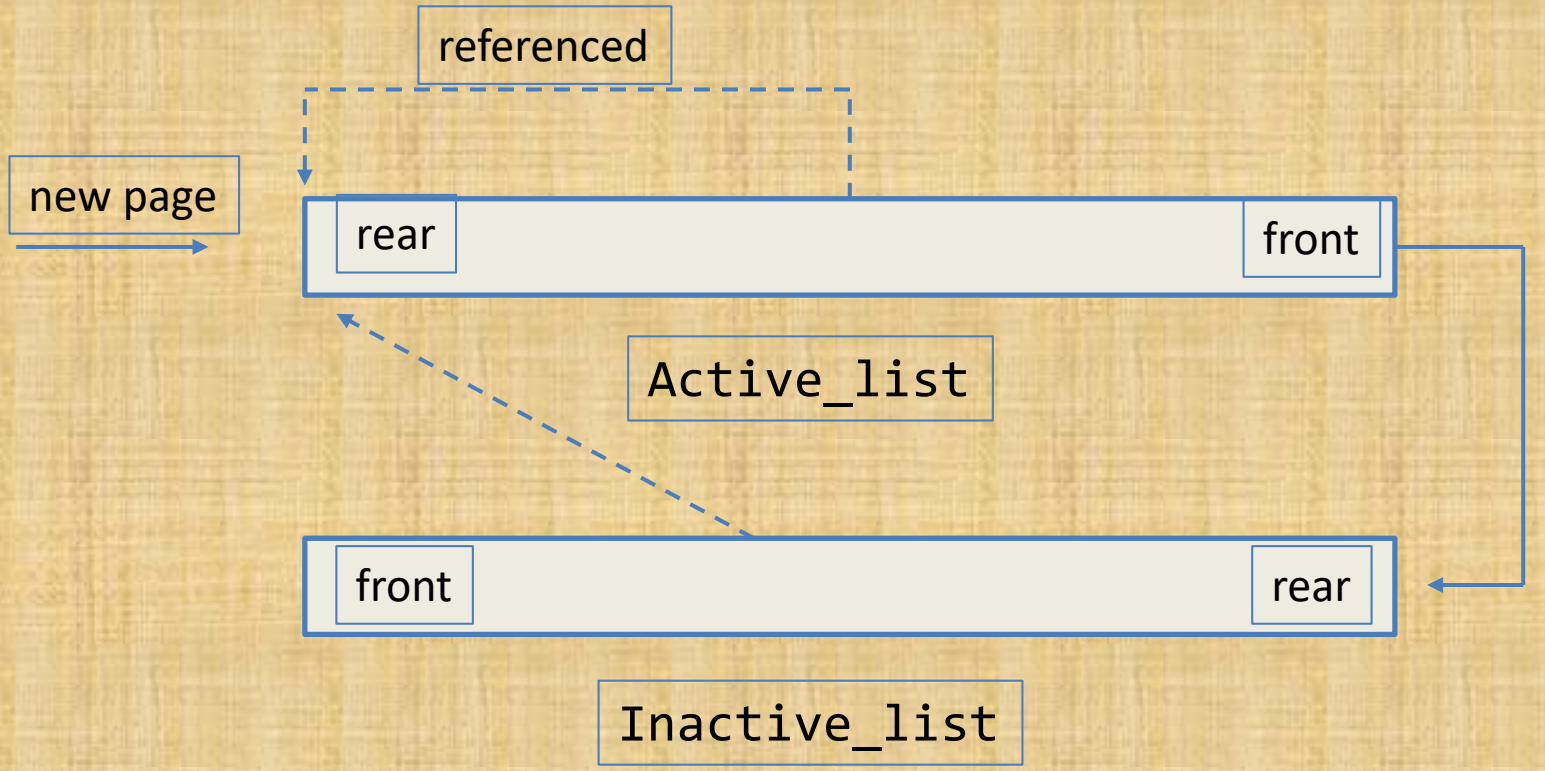
Linux

- Use demanding paging, allocating pages from a list of free frames.
- Employ a global page-replacement policy similar to LRU-approximation clock algorithm.
- Maintain two types of page lists: an **active list** containing pages that are in use, and an **inactive list** containing pages that have not been used recently and are eligible to be reclaimed.

Maintaining the Two Lists

- Each page has an **accessed** bit that is set whenever the page is referenced.
- When a page is first allocated, its access bit is set, the page is added to the **rear** of the **`active_list`**.
- Whenever a page is referenced, its access bit is set, the page is moved to the **rear** of the **`active_list`**.
- Periodically the access bit of pages in the **`active_list`** are reset.
- Over time, the least recently used pages will be at the **front** of the **`active_list`**.
- When the reclaiming algorithm is run, some pages are moved from the **front** of the **`active_list`** to the rear of the **`inactive_list`**.
- Note that the use and the meaning of "**front**" and "**rear**" in these two lists.
- See an illustration on next slide.

Active and Inactive Lists



Linux Virtual Memory References

- Some relevant references
  - **Better active/inactive list balancing** https://lwn.net/Articles/495543/
  - **Understanding memory information on Linux systems** https://linux-audit.com/understanding-memory-information-on-linux-systems/
  - **Page Frame Reclamation** https://www.kernel.org/doc/gorman/html/understand/understand013.html

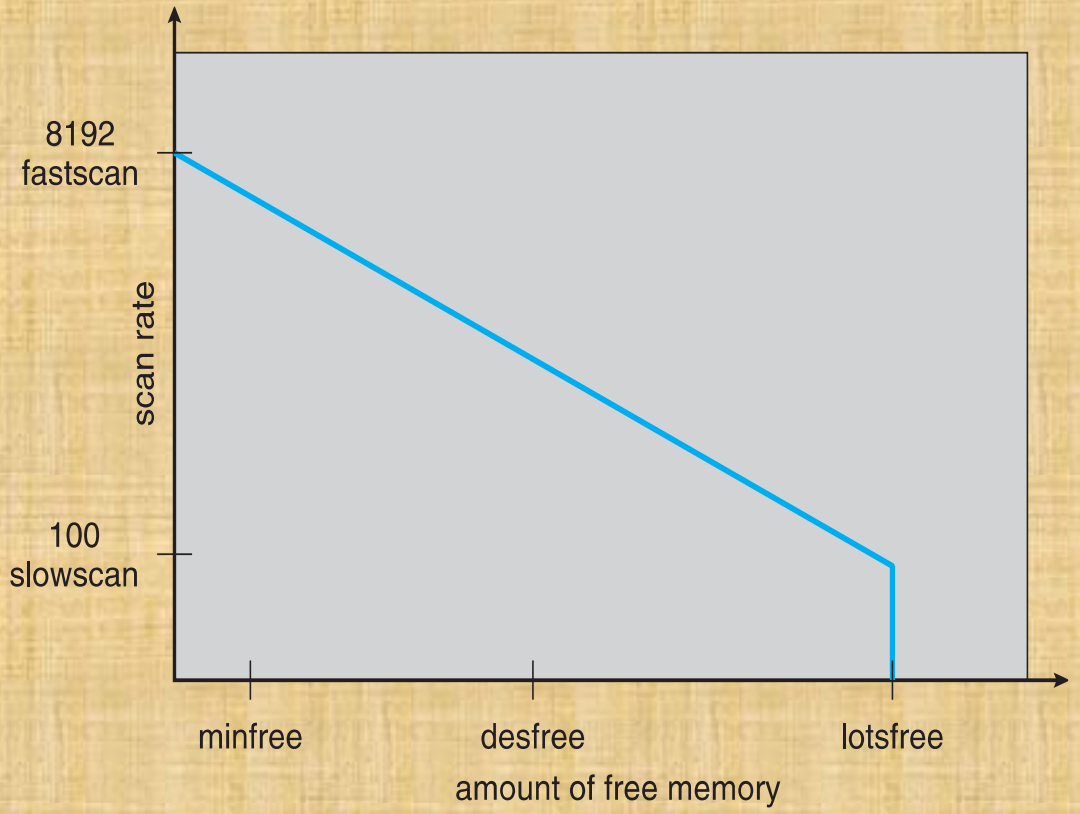
Windows

- Uses demand paging with **clustering**. Clustering brings in pages surrounding the faulting page
- Processes are assigned **working set minimum** and **working set maximum**
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory
- A process may be assigned as many pages up to its working set maximum
- When the amount of free memory in the system falls below a threshold**, automatic working set trimming** is performed to restore the amount of free memory
- Working set trimming removes pages from processes that have pages in excess of their working set minimum

Solaris

- Maintains a list of free pages to assign faulting processes
- **`lotsfree`** – threshold parameter (amount of free memory) to begin scanning at a slow rate (high threshold)
- **`desfree`** – desired amount of free frames (mid point)
- **`minfree`** – threshold parameter to begin swapping (low threshold of free frames)
- Paging is performed by the **`pageout`** process
- **`pageout`** scans pages using modified clock algorithm
- **`scanrate`** is the rate at which pages are scanned. This ranges from **`slowscan`** to **`fastscan`**
- **`pageout`** is called more frequently depending upon the amount of free memory available
- ***Priority paging*** gives priority to process code pages

Solaris 2 Page Scanner



Solaris page reclaiming