Linux Memory Allocation

Page Descriptor

- One for each page of memory
- Quite large — at least 28 bytes *plus* a `list_head`
- Many flags — locked, dirty, accessed, active/inactive, being reclaimed, and more
- Reference counter — how many page tables is this page part of?

Non-Uniform Memory Access

- For some CPU types, some forms of memory are more expensive to access than others
- Memory organized into *nodes*
- Not needed for Pentiums, but complicates the rest of memory architecture

Memory Zones

- Not all memory is equally addressable
- Different types of memory have to be used for different things
- Linux uses different *zones* to handle this
- ZONE_DMA: Some older I/O devices can only address memory up to 16M
- ZONE_NORMAL: Regular memory up to 896M
- ZONE_HIGHMEM: Memory above 896M

4 / 33

High Memory

- On Pentiums, the Linux kernel can't address memory over 1G. Why not?
- Not enough address space available, given Linux's addressing strategy — where the kernel is, where the user programs start, etc.
- Memory up to 896M is addressed directly
- Virtual addresses between 896M and 1G are constantly changed, to address physical memory at higher addresses
- On 64-bit machines, this isn't an issue; all memory is directly addressable

5 / 33

Slab Allocator

- There are certain kinds of data structures that are frequently allocated and freed
- Instead of constantly asking the kernel memory allocator for such pieces, they're allocated in groups and freed to per-type linked lists
- To allocate such an object, check the linked list; only if it's empty is the generic memory allocator called
- To free such an item, just put it back on the list
- If a set of free objects constitute an entire page, it can be reclaimed if necessary

6 / 33

Linux Page Frame Reclaiming Algorithm 7 / 33

Principles

- Linux does not predetermine how memory is used
- All pages can be used for all purposes
- Well, most. . .
- Memory will be consumed until there's none left
- The Linux *Page Frame Reclaiming Algorithm* (PFRA) has to make sure that there is some free

8 / 33

Types of Pages

Type	Description	Action
Unreclaimable	Locked pages, kernel mode stacks, free pages, etc.	Impossible
Swappable	Anonymous user-mode pages	Write to swap area
Syncable	Parts of files; mapped user mode pages	Write to file if necessary
Discardable	Unused pages	Do nothing

9 / 33

Mapped versus Anonymous Pages

- A *mapped* page is part of a file
- It corresponds to a given block in the file system
- Very often, such pages are read-only and hence can't be dirty
- An *anonymous* page does not correspond to any file — it may be part of a program's data area or stack — and has to be written to the swap area

10 / 33

Reclamation Principles

- First reclaim pages not associated with any process (no page table changes needed)
- Virtually all user-mode pages are reclaimable
- For shared pages, clear all page table entries simultaneously
- Only reclaim "unused" pages, i.e., those that haven't been referenced recently

11 / 33

When to Reclaim Memory

- Low on memory
- Hibernation (part of laptop power management — out of scope)
- Periodically — make sure that memory will be available whenever it's needed

12 / 33

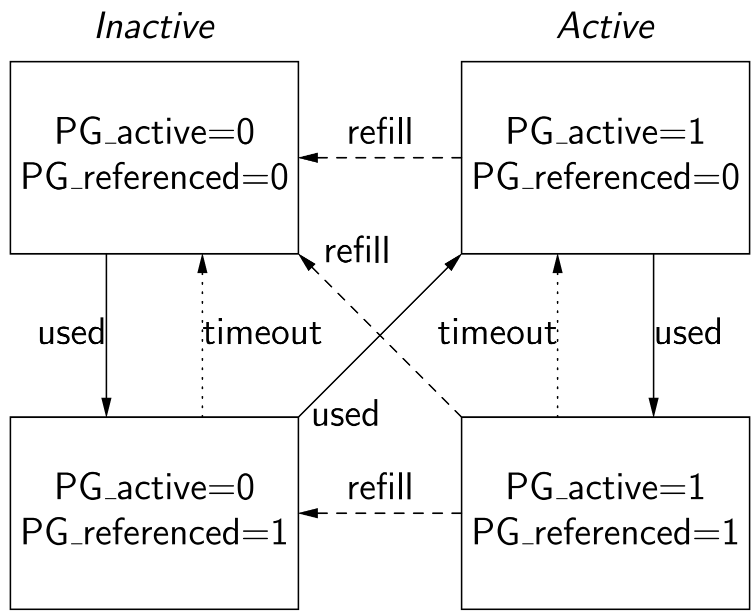
The LRU Lists

- Two linked lists of pages for each process: the *active list* and the *inactive list*
- Clearly, we reclaim pages from the inactive list first
- If a page hasn't been referenced lately, it moves to the inactive list
- If a page is referenced, it *doesn't* get moved to the active list immediately

Double Accesses

- The first time a page is accessed, the `PG_referenced` flag is set
- The next time it's accessed, it's moved to the active list
- That is, it takes two accesses for a page to be declared active
- More precisely, it takes two accesses *in different scans* for a page to become active
- If the second access doesn't happen soon enough, `PG_referenced` is reset

Access States



Access States

- When a page is referenced, it moves to a more active state
- After two accesses, it moves to the active list
- When a page hasn't been used for a while, it moves a less active state
- After two timeouts, it moves to the inactive list
- If memory is getting low, pages can be demoted regardless of activity state

Refilling Memory

- Periodically try to make pages reclaimable
- If too aggressive, too many active pages are reclaimed
- If too timid, not enough memory will be free
- Works adaptively

Adaptive Rate

- Start by scanning a few pages to see if they should be reclaimed
- If memory is running low, increase the rate
  Controlled by `prority` field
- Conversely, reduce rate if ok
- If more memory needed, tend to swap out process' pages

Equations

- `distress` is a trouble measure: 0 is good, 100 is great trouble:
  `distress = 100 >> prev_priority;`
- How much memory is currently mapped?
  `mapped_ratio = (nr_mapped * 100) / total_memory;`
- Should we tend to swap out process pages?
  `swap_tendency = mapped_ratio / 2 + distress + vm_swappiness;`
- Note: `vm_swappiness` is tunable by the administrator. High values mean it will swap user mode pages more readily; 0 means it almost never will.

19 / 33

Writing Dirty Pages

- Normally, not many dirty pages are written at any one time
- You don't want to clog up the disk bandwidth, since the write may be useless — if anticipatory, the page may never actually be reclaimed, or it may be dirtied again before reclamation
- However, if memory is low — that is, if an allocation request has failed — a large burst can be written

20 / 33

Periodic Reclaiming

- A kernel thread kswapd runs periodically to reclaim memory
- Some *must* be free at all times — often need to allocate memory at interrupt level, when sleeping is impossible
- If there's too little free memory in the zone (less than pages_min), it tries to find more; if more than pages_high, it does nothing
- After reclaiming 32 pages, kswapd yields the CPU and calls the scheduler, to let other processes run

21 / 33

The Out-of-Memory Killer

- If there's no free and no reclaimable memory, the system is in trouble
- Last resort: kill some process
- Find a process that fits the following criteria: big (including its children), hasn't run too long, low priority, non-root

22 / 33

The Swap Token

- To prevent thrashing, one process at a time can hold the *swap token*
- Pages belonging to the swap token owner are (almost) never reclaimed
- The idea is to force other process' pages out of memory, to let one process run
- Ideally, the token should be held for a considerable amount of time, possibly even minutes
- The token is a pointer to the process' memory data structure; it simply skips any memory structure if it equals the token

The Swapper

- On Linux, the swapper writes pages out to the swap area and reads in other pages
- Also manage swap areas on disk
- Keeps track of mapping between per-process virtual address and disk block address

The Swap Area

- Frequently a separate disk partition
- Can also be a regular file on disk, but that's slower
- Swap area starts with some magic values to identify the partition as a swap area
- Also keeps track of bad blocks

Distribution of Pages

- Try to keep pages in contiguous page slots in the swap area
- Improves performance — minimize seek time
- (That's why files are bad for swapping)
- With multiple swap areas, prioritize them by access speed
- Among swap areas of equal speed, use round-robin

The Swap Cache

- Several race conditions possible
- Example: two processes simultaneously try to swap in the same shared page
- Example: a process may try to swap in a page that is currently being swapped out
- The *swap cache* is used as an intermediate owner of pages
- All attempts to change a page's status must go through the cache
- This provides the requisite locking

28 / 33

vmscan.c: Where Decisions Are Made

- The page frame reclamation algorithm
- Decides what pages to swap out
- Implements the LRU algorithm

29 / 33

struct scan_control

Policy and state of PFRA decisions

nr_to_scan	Scan this many pages
nr_scanned	Number actually scanned thus far
nr_reclaimed	Number reclaimed
nr_mapped	Number of mapped pages found
nr_to_reclaim	Number to reclaim
priority	Priority
may_writepage	Disk queue too full for writing?
may_swap	Can pages be swapped out here?

30 / 33

try_to_free_pages()

- Called when it's time to find some free memory
- Iterates until enough pages are found
- Writes a group, but not too many; it then sleeps
- If enough pages are reclaimed, it returns

31 / 33

refill_inactive_zone()

- Moves pages from active list to inactive
- Looks at distress level, mapped ratio, etc.

shrink_list() – the Heart of PFRA

- The caller (`shrink_cache()` moves a group of pages from the LRU list to a temporary list
- `shrink_list()` tries to reclaim each page on this list
- It removes the page from the list; if it can't reclaim it, it puts it back
- Pages that appear to be locked for a long time are put on the active list
- Others are kept on the inactive list, to be freed next time
- There is no bias against reclaiming dirty pages; however, they can be written out here