Alejandro Palacios
PID: 5156050

Memory Management in Linux Kernel

Each kernel modification has the comment 'ADDED BY ALEJANDRO PALACIOS' so you can easily find the modifications in mm.h, mmzone.h, and vmscan.c by searching for 'ALEJANDRO PALACIOS' in the document.

Part 1: Add a Static System Call (sys_memstats) to report on memory management statistics

For this part of the assignment, the instructions in assignment 2 were followed to add the sys_memstats system call to the kernel. The more interesting portion are the means in which each statistic is retrieved from kernel structures.

1.) The current number of free pages.
    a.) For this item, it was necessary to iterate over each zone. Each zone has a variable named 'free_pages' which conveniently gives the number of free pages in that zone. The value is accumulated over each zone to give the result.
2.) The current number of pages used by the slab allocator.
    a.) To get this item, it was necessary to iterate over each zone. A function called 'zone_page_state' could then be used as follows to get the value: zone_page_state(current_zone, NR_SLAB). This was accumulated over each zone.
3.) Current number of pages in the active list.
    a.) The zone structure contains a list head structure to the active list. To get this item, you could simply iterate over the list structure and increment for each value traversed.
4.) Current number of pages in the inactive list.
    a.) Similar to number 3, the zone structure also had a list head structure containing the tip of the inactive list. The same method was used.
5.) Current number of pages in the active list whose reference bits are set.
    a.) While iterating over each page in the active list, the function 'PageReferenced' was used to increment the counter if the condition was true.
6.) Current number of pages in the inactive list whose reference bits are set.

    a.) Similar to number 5, the ‘PageReferenced’ function was used when iterating over the inactive list.

7.) Cumulative number of pages moved from the active to inactive list.

    a.) This item required some modification to the kernel. Firstly, a long variable called ‘active_inactive_count’ was added to the zone struct. The value is mutated in the ‘shrink_active_list’ function in vmscan.c by adding ‘pgdeactive’ to its current value.

8.) Cumulative number of pages evicted from the inactive list.

    a.) This item also required modification to the kernel source. A long variable called ‘inactive_evicted_count’ was added to the zone struct. The value is mutates in the ‘shrink_inactive_list’ function in vmscan.c by adding ‘nr_reclaimed’ to its current value.

<u>Results</u>

The results before virtual memory is used were as follows:

Number of free pages: 77151
Number of pages in slab allocator: 9522
Number of active pages: 41663
Number of inactive pages 121531
Number of active pages with set reference bits: 15798
Number of inactive pages with set reference bits: 43199
Number of pages sent from the active to inactive list: 0
Number of pages sent from inactive from evicted: 0

Returned 0

To compare with the cat /proc/meminfo | less command, we must divide its results by 4 since the pagesize is 4096 Kb.

MemTotal: 1023956 kB
MemFree: 308596 kB / 4 = **77149 pages**
Buffers: 34652 kB
Cached: 528176 kB
SwapCached: 0 kB
Active: 166796 kB / 4 = **41,699 pages**
Inactive: 486116 kB / 4 = **121542 pages**

```
HighTotal:          0 kB
HighFree:           0 kB
LowTotal:      1023956 kB
LowFree:        308596 kB
SwapTotal:     2064376 kB
SwapFree:      2064376 kB
Dirty:              0 kB
Writeback:          0 kB
AnonPages:       90124 kB
Mapped:          18988 kB
Slab:            38092 kB / 4 = 9,525 pages
PageTables:       6408 kB
NFS_Unstable:       0 kB
Bounce:             0 kB
CommitLimit:   2576352 kB
Committed_AS:   200196 kB
VmallocTotal: 34359738367 kB
VmallocUsed:     18272 kB
VmallocChunk: 34359718903 kB
HugePages_Total:     0
HugePages_Free:      0
HugePages_Rsvd:      0
Hugepagesize:     2048 kB
```

The negligible differences in results can be attributed to the fact that the programs were not called at the exact same time. Aside from that, it can be concluded that the results are accurate.

When the stress system call is used, virtual memory kicks in and we can see the results for items 7 and 8 change:

Note stress_system.c is intensive on memory and you may have to forcibly terminate using Ctrl+C because it might take a very long time to finish. Regardless, it obtains the intended results of forcing virtual memory management to occur.

```
Number of free pages: 232775
Number of pages in slab allocator: 5808
```

Number of active pages: 358
Number of inactive pages 10844
Number of active pages with set reference bits: 174
Number of inactive pages with set reference bits: 173
Number of pages sent from the active to inactive list: **973054**
Number of pages sent from inactive from evicted: **472967**

Part 2: Counter-based clock page replacement algorithm

There were several modifications to the kernel structure in this assignment. Firstly, mm.h’s page structure had to be modified to include a reference counter named ‘ref_count’. The primary targets for modification are shrink_active_list and shrink_page_list in vmscan.c.

Here is how the algorithm was modified in shrink_active_list to replace the old algorithm. We have to mutate the reference count using my function called ‘mutate_ref_count’ to increment it if it is referenced before doing the check. The if statement to potentially send the page is modified to use my function called ‘is_page_referenced’ which decrements the reference count and returns 1 (true) if referenced. If referenced, the page will be re-added to the active list.

```
 //ADDED BY ALEJANDRO PALACIOS
        while (!list_empty(&l_hold))
          {
              cond_resched();
              page = lru_to_page(&l_hold);
              list_del(&page->lru);
              //Mutates the reference count before determining to send to inactive list or
not.
              mutate_ref_count(page);
              if (page_mapped(page))
              {
                     if (!reclaim_mapped ||
                         (total_swap_pages == 0 && PageAnon(page)) ||
                         is_page_referenced(page)/*new check that replaced the old
page_referenced function*/)
                     {
                            list_add(&page->lru, &l_active);
                             trace_mm_pagereclaim_shrinkactive_a2a(page);
```

```
                continue;
            }
        }
        list_add(&page->lru, &l_inactive);
    }
```

Here is how the algorithm is modified in the shrink_page_list function. If all the checks go through before this modification, the same ‘is_page_referenced’ function is used. If the page is not referenced and all the other original conditions hold, evict it.

```
 //ADDED BY ALEJANDRO PALACIOS
            if (PagePrivate(page))
        {
                //Adds reference bit to inactive page if referenced.
            mutate_ref_count(page);
            if (!try_to_release_page(page, sc->gfp_mask))
                goto activate_locked;
            //Check added to see if page counter > 0.
            if (!mapping && page_count(page) == 1 && !is_page_referenced(page))
                goto free_it;
        }
```

The timer was added to the memstats.c along with the system call and mutates the reference count every 30 seconds.

To check the effectiveness of this new algorithm, I timed how long it took for the new clock-based algorithm to complete the stress test in stress_system.c vs the original algorithm.

Test run	New algorithm	Old algorithm
1	13 seconds	9 seconds
2	10 seconds	9 seconds
3	7 seconds	9 seconds

Here are the results from my system call after running the tests:

New algorithm

Number of free pages: 236415
Number of pages in slab allocator: 5578
Number of active pages: 6792
Number of inactive pages 1106
Number of active pages with set reference bits: 195
Number of inactive pages with set reference bits: 241
Number of pages sent from the active to inactive list: 1418651 (sends more to inactive)
Number of pages sent from inactive from evicted: 629804 (evicts less pages)

Old algorithm

Number of free pages: 235326
Number of pages in slab allocator: 5877
Number of active pages: 2444
Number of inactive pages 6819
Number of active pages with set reference bits: 163
Number of inactive pages with set reference bits: 146
Number of pages sent from the active to inactive list: 802604 (sends less to inactive)
Number of pages sent from inactive from evicted: 636322 (evicts more pages)

On average, the old algorithm did perform better. However, the last run was interesting because the new algorithm performed better. The trend seemed to have been that the new algorithm did better after each subsequent run, while the old one had the same execution time.

I hypothesize that this might have to do with the new algorithm not evicting pages as often as the old algorithm due to it having a reference counter rather than a reference bit that it relies on. Keeping the pages in memory might improve the performance between runs.