Assignment 4

***Student**: Alicia Rodriguez - 5162522*

***Due Date**: 07/22/2017 by 11:55pm*

Part 1: Memory Management Statistics System Call

getconf PAGESIZE: 4096 B

Current number of free pages: Free pages can be found in kB `cat /proc/meminfo` as `MemFree`. Since the page size is 4096 B, then to verify the number, you need to get the `MemFree` value and divide it by 4.

Current number of pages used by slab allocator: Slab pages can be found in cat /proc/vmstat as nr_slab

Current number of pages in the active list: Active list can be found in kB in `cat /proc/meminfo` as `Active`. Since the page size is 4096 B, then to verify the number, you need to get the `Active` value and divide it by 4.

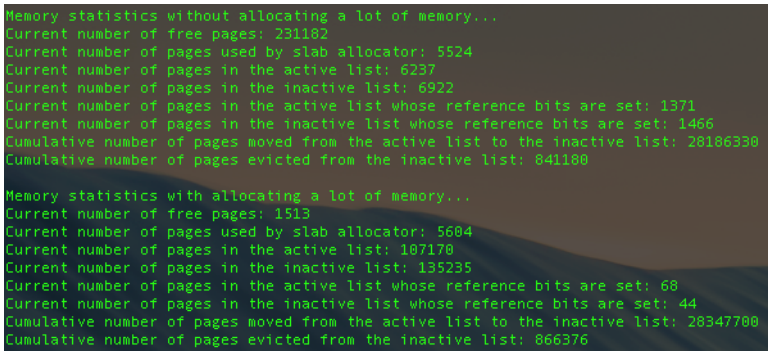
Current number of pages in the inactive list: Inactive list can be found in kB in `cat /proc/meminfo` as `Inactive`. Since the page size is 4096 B, then to verify the number, you need to get the `Inactive` value and divide it by 4.

Current number of pages in the active list and inactive list whose reference bits are set: By using the `test_bit` function and `PG_referenced`, you are able to get the pages which have been referenced or not.

Cumulative number of pages moved from the active list to the inactive list: This number is the same number found in `cat /proc/vmstat` as `pgdeactivate`. This makes sense because those are the pages that have been moved from the active list to the inactive list; hence, the ones that have been deactivated.

Cumulative number of pages evicted from the inactive list: This number is the same number found in `cat /proc/vmstat` as `pgsteal_dma32`.

The following image shows a dramatic change in the numbers described when allocating a lot of memory:



KERNEL COMPONENTS MODIFIED:

- `/usr/src/linux/arch/i386/kernel/syscall_table.S`
- `/usr/src/linux/include/asm-i386/unistd.h`
- `/usr/src/linux/include/asm-x86_64/unistd.h`
- `/usr/src/linux/include/linux/syscalls.h`
- `/usr/src/linux/Makefile`
- `/usr/src/linux/mm/vmscan.c`
- `/usr/src/linux/mm/page_alloc.c`
- `/usr/src/linux/include/linux/mmzone.h`

KERNEL COMPONENTS CREATED:

- `/usr/src/linux/include/linux/memstats.h`
- `/usr/include/memstats.h`
- `/usr/src/linux/memstats`
- `/usr/src/linux/memstats/memstats.c`
- `/usr/src/linux/memstats/Makefile`

REFRENCES:

- Implementing a System Call on Linux 2.6 for i386 <http://tldp.org/HOWTO/html_single/Implement-Sys-Call-Linux-2.6-i386/#AEN19>
- /proc/vmstat <http://linuxinsight.com/proc_vmstat.html>

Part 2: Counter-Based Clock Page Replacement Algorithm

STEPS USED TO IMPLEMENT THE COUNTER-BASED CLOCK ALGORITHM AND HOW IT WORKS

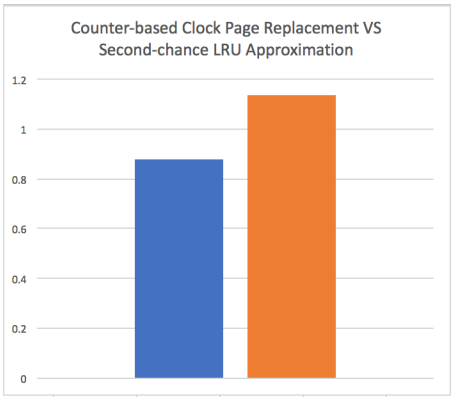
1. Add `reference_counter` variable to the `struct page` definition in `mm.h`
2. In `page_alloc.c`, `reference_counter` is initially set to 0. This is the same file where other initializations are made.
3. In the `shrink_active_list()` function in `vmscan.c`:
    a. Add reference bit value to the `reference_counter` and clear the reference bit (at the same time)
    b. If `reference_counter` reaches maximum value, do not overwrite the value when adding the reference bit to it.
    c. Check the `reference_counter`: if 0, evict page; otherwise, decrement `reference_counter` by 1 and move the frame to the back of the list (as the original already does).
4. Scan the frames periodically (as part of the timer interrupt)
5. Add the reference bit value to the `reference_counter` and clear the reference bit (at the same time)

How does the counter-based clock page replacement algorithm work? By following the steps above, it essentially uses the `reference_counter` as a way to track activity, as opposed to using the referenced bit.

Why is it useful for comparing the page replacement implementations? To demonstrate that the counter-based clock page replacement algorithm is more efficient than the second-change LRU approximation algorithm. Additionally for checking if the rate of page faults has been minimized as well as how effective it is in evicting pages.

Experiments conducted: By triggering on and off the algorithm implemented on the `vmscan.c` file and ran the `test_algorithms.c` test program in order to compare the CPU times of both algorithms. The experiments conducted can be seen in `test/test_algorithms_results.xlsx`.

Measurements obtained: Obtained the amount of seconds it would take to allocate a log of memory for both the algorithms. As expected, the counter-based clock page replacement algorithm takes less time than the second-chance LRU approximation algorithm. This can be shown by the performance results chart below. The blue bar is the average CPU time used for counter-based clock page replacement algorithm while the orange bar is the second-change LRU approximation. More statistical results can be found in `test/test_algorithms_results.xlsx`.



KERNEL COMPONENTS MODIFIED:

- `/usr/src/linux/include/linux/mm.h`
- `/usr/src/linux/mm/page_alloc.c`
- `/usr/src/linux/mm/vmscan.c`

REFRENCES:

- Calculating Time <https://www.gnu.org/software/libc/manual/html_node/CPU-Time.html>