Jesse Osiecki

COMP 530H

Prof. Smith

November 25, 2014

VMlogger module – Page fault analysis on the Linux 2.6.32 Kernel

In this assignment a module utilizing debugfs for psuedo-system calls was created. When called, it replaces all of the page fault handler functions for the VMAs of a given process with our own; in this case to simply log data about each page fault. Our function then calls the original fault handler to collect metrics. The kernel then prints to the log a line such as:

“Nov 25 00:35:10 localhost kernel: vmlogger: MM c16e5580 PAGE 2790 PAGE_OFFSET 161 PFN 510593 TIME 1875”

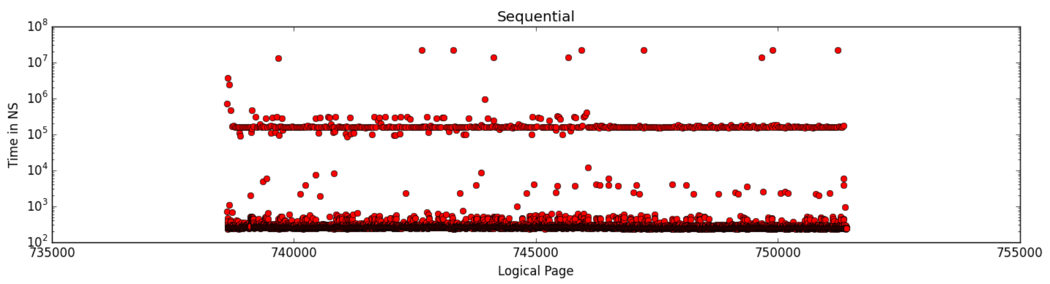
where we have the Memory management unit's address, the Logical Page, the Page offset, the Page Frame Number, and the time it took to execute the original fault handling function.

To test the effect of different access types on how the fault handler performs, we create three files using “dd if=/dev/urandom” of a size roughly 50 MB.[1] Using the data collected, three graphs for each were created, the first plotting the logical page that was faulting against the time to execute the fault handler (in nanoseconds), then the page offset against the same time, and then the page frame against time. This allows us to explore different access schemes effect on both virtual and physical memory.

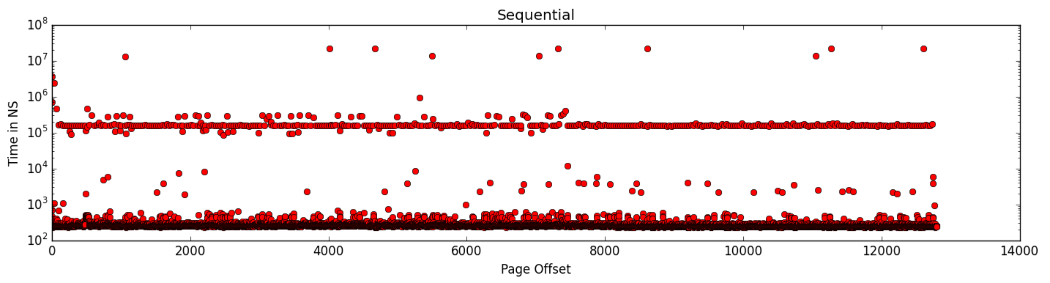
Sequential Access:

1 Note that three files were used as opposed to one to ensure that the OS wasn't caching the file in subsequent tests

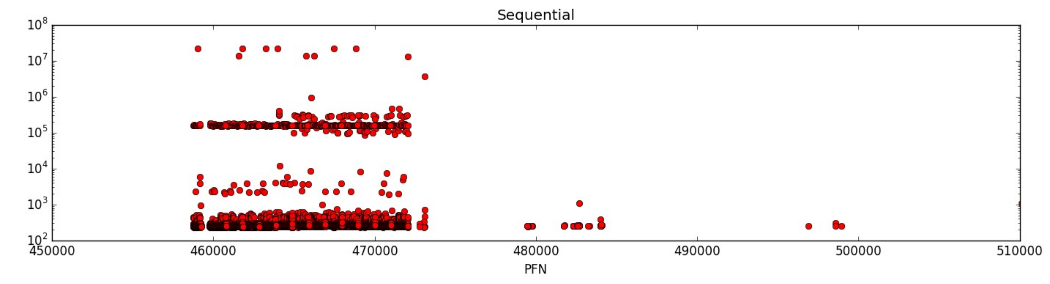
Page faults and the respective handling time



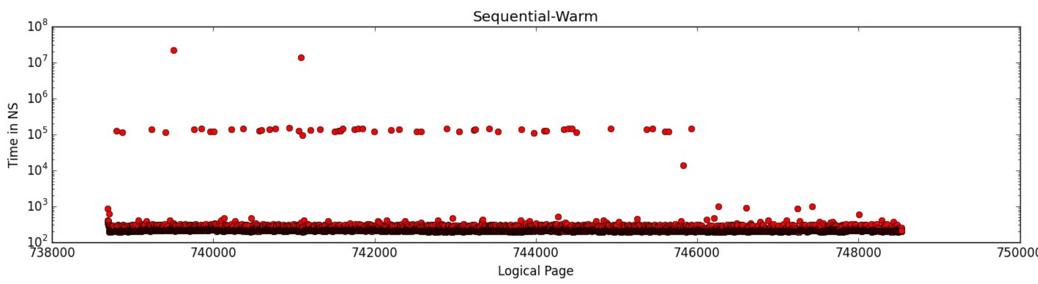
Page faults and the respective handling time



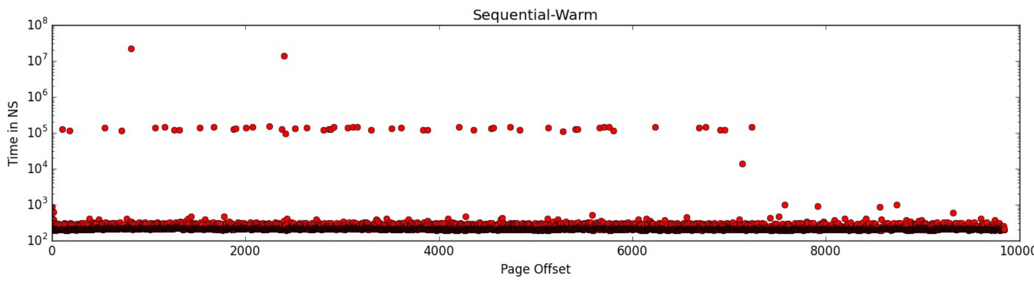
Page faults and the respective handling time



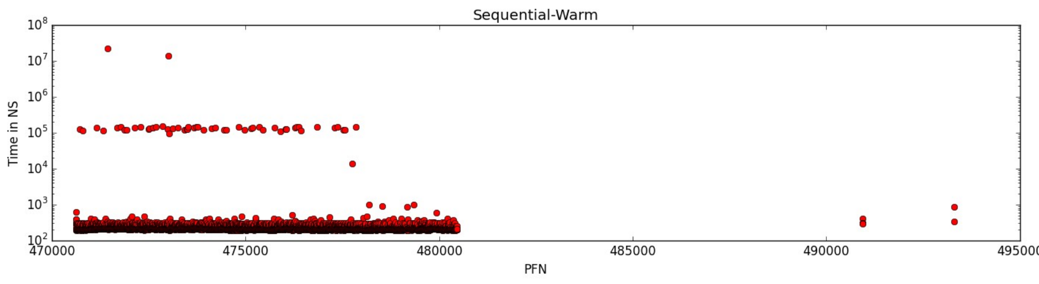
Page faults and the respective handling time



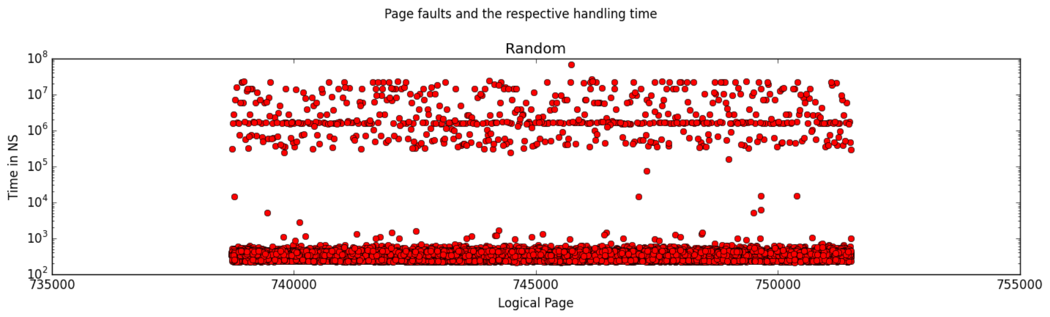
Page faults and the respective handling time

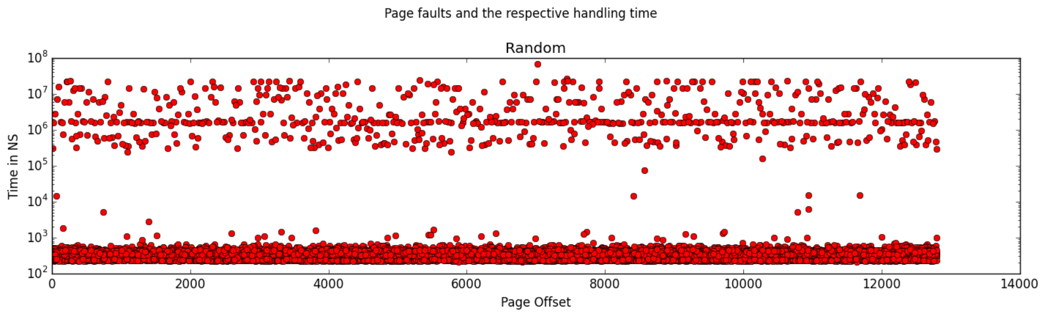


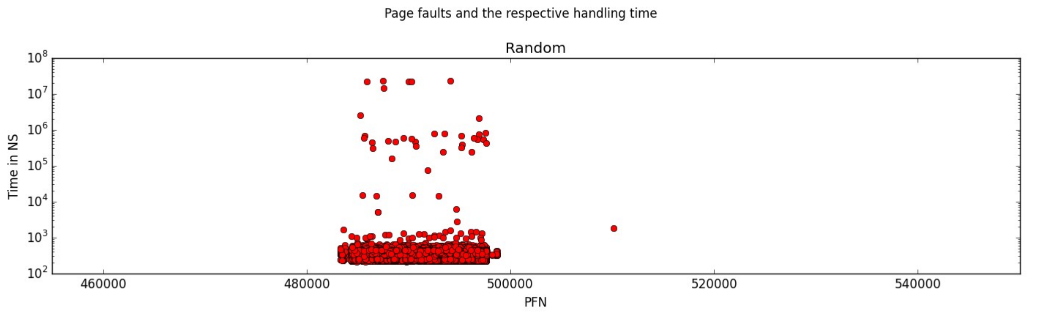
Page faults and the respective handling time

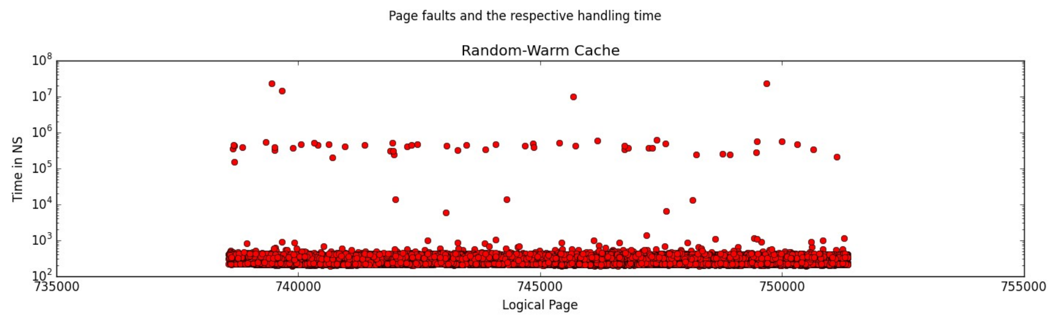


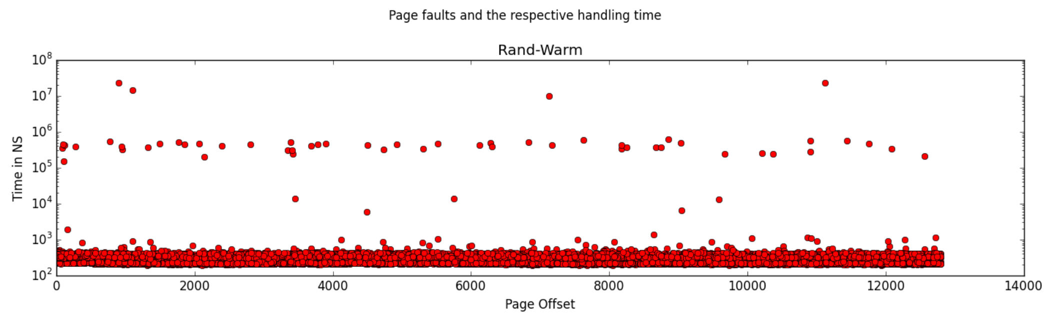
Random Access:

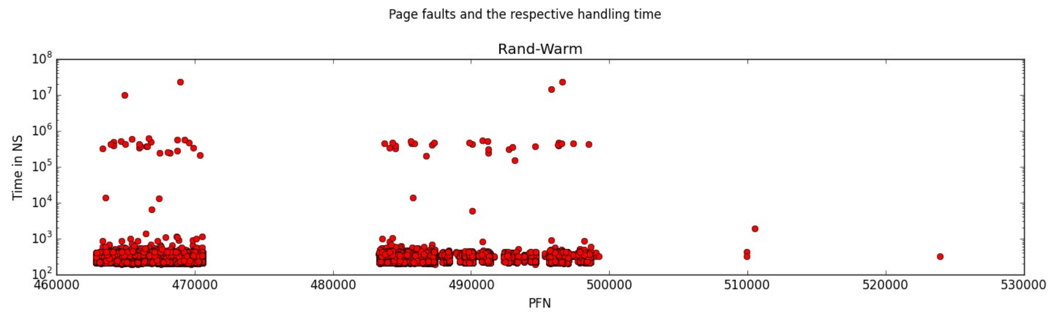




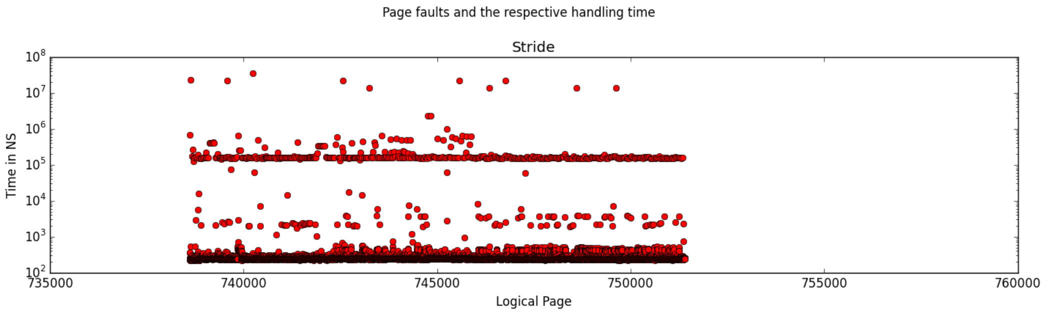


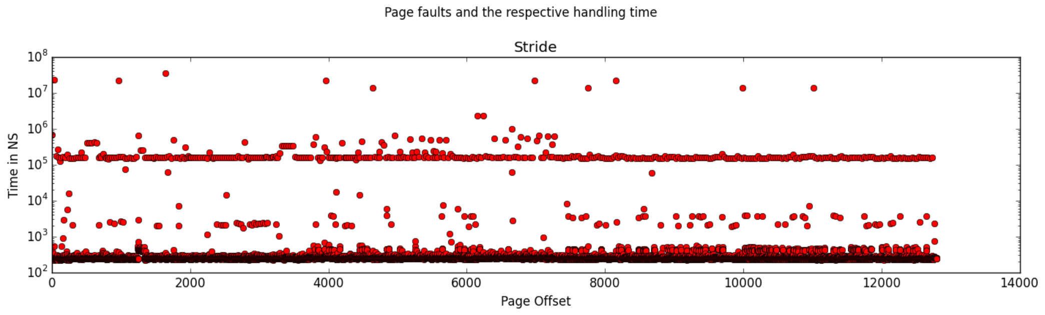

Page faults and the respective handling time
Random-Warm Cache
Time in NS
Logical Page
10^2 10^3 10^4 10^5 10^6 10^7 10^8
735000 740000 745000 750000 755000

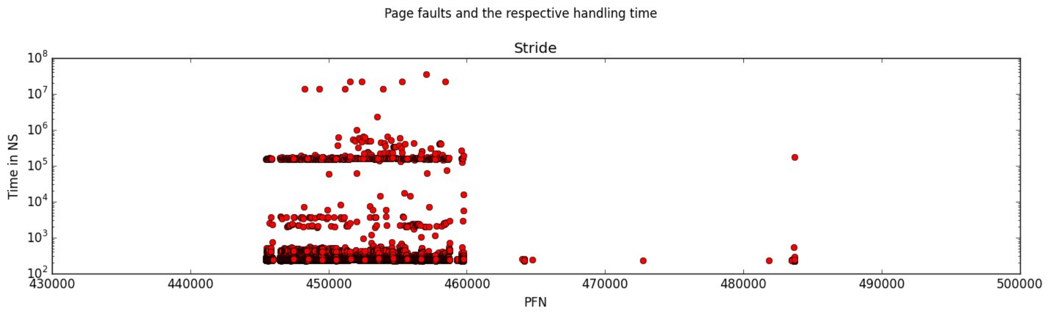

Page faults and the respective handling time
Rand-Warm
Time in NS
Page Offset
10^2 10^3 10^4 10^5 10^6 10^7 10^8
0 2000 4000 6000 8000 10000 12000 14000


Page faults and the respective handling time
Rand-Warm
Time in NS
PFN
10^2 10^3 10^4 10^5 10^6 10^7 10^8
460000 470000 480000 490000 500000 510000 520000 530000

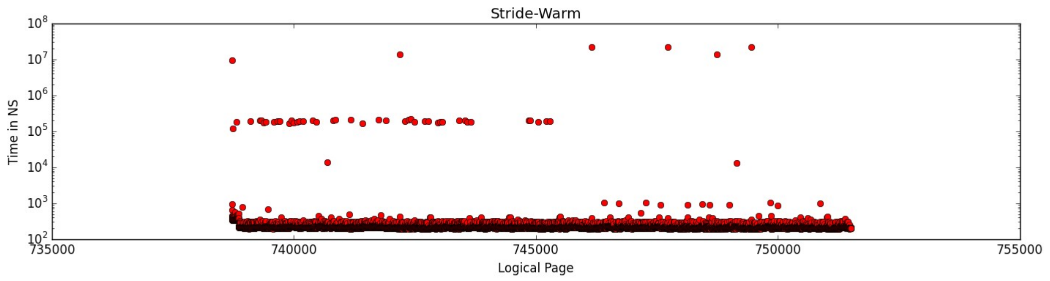
Stride access (Sequential access in large step offsets):



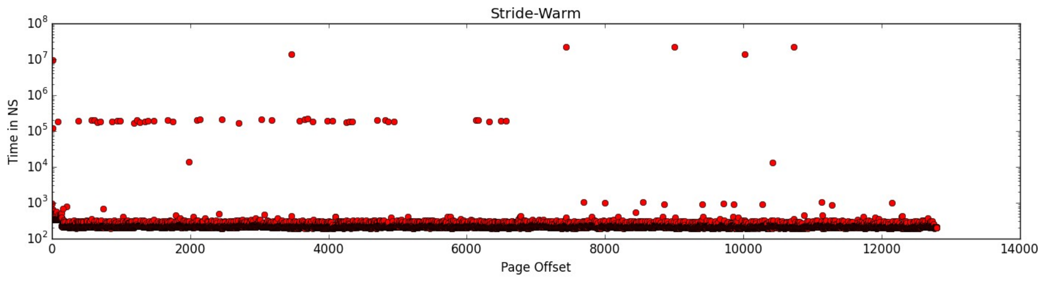


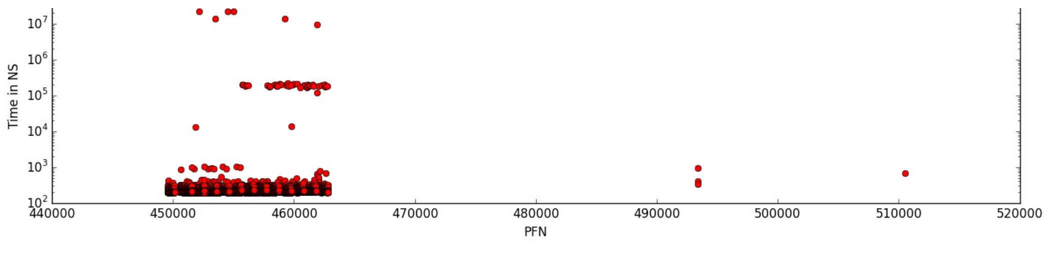


Page faults and the respective handling time



Page faults and the respective handling time





Looking at the time to do a fault from any metrics here, it can be seen that clearly the warm-cache is faster than the cold cache (as expected). This suggests that Linux is actually keeping some of these pages in memory when the file access is out of scope.

The Random access patterns seemed to give the most variation on fault handling times that likely used disk io. This could because of the random nature; having to access a file in random rather than sequential blocks would cause the disk to do more seeks (or less optimal ones), possibly accounting for this difference. Furthermore, the average access time for Random seemed to be to the order of 10^6 ns whereas sequential was closer to 10^5 ns, further enforcing such hypothesis.

Something worth noting is that in the Random-Warm test, it can be seen in the PFN v. Time graph that there is a large gap in the enumeration of Page Frames. This could have been caused by a preemption in the execution of the mmap (maybe something else's logical pages were then mapped to those frames before our process resumed execution).