COMP 530H
Fall 2014

Honors Assignment 5
Module to implement a vmlogger service
Date assigned: October 31, 2014
Date due: November 14 , 2014

Implement a service module that provides dynamic logs of page fault behavior. Each Linux process has a dynamically allocated set of VMAs, each represented by one vm_area_struct data structure. The set of VMAs for a process is held in a linked list rooted in the mm_struct for the process. When vmlogger is called, your module should initiate logging (with printk to /var/log/kernel.log) of all page faults occurring in the VM areas belonging to the process. Limit the number of processes using your module to one at a time. Page activity logging can be implemented using a technique based on virtual function calls similar to the technique used for assignment 4. The vm_area_struct includes a set of function pointers (vm_operations_struct *vm_ops) that reference specialized functions to implement generic operations on virtual memory areas (e.g. different classes of memory areas have different page fault handling algorithms). An example of the generic call to a specific fault handler can be found function __do_fault() in mm/memory.c.

For page faults, your module should provide a wrapper function that calls the real fault handling code for the VMA and, when it returns, creates a log entry that records the address of the owning mm_struct, the virtual page number of the faulting logical address, the offset of the logical page in the VMA, the real page frame number (pfn) of a page mapped for the faulting address (see page_to_pfn), and the elapsed time in nanoseconds for the page fault handler to execute (see ktime_get and ktime_to_ns; note that time in nanoseconds is unsigned 64 bits or u64). The generic calls other than fault should be passed through to the normal implementations. Note that not all vm_area_structs for a process will have a non-NULL vm_ops pointer. Similarly, not all function pointers in a vm_operations_struct will be non-NULL. The important kernel files related to this assignment are include/linux/mm_types.h, and include/linux/mm.h,

Your module should deal with any errors that can occur (but not be burdened with tests for errors that cannot occur). The calling program should receive sufficient information so that it can determine if the call has been successful or has failed.

The most significant part of this assignment will consist of conducting experiments using the logging functions to gain insights into the paging behavior of carefully designed test cases. The recommended approach for this is to write test programs that use the Linux mmap() function to map a very large (~50 MB) file into memory. Once the mapping is done, you can then use pointers or array indexing to experiment with various access patterns to locations in the mapped memory. For example, you could use both sequential and random patterns of access. Note that the file mapping must take place before the program calls your module so its VMA will be allocated, and that the access patterns should take place after so they will be logged. A code fragment that gives an example of mapping a big file and accessing it will be placed on the course web page

From the logs generated you could plot time series of faulting virtual page numbers or the corresponding page frame numbers that are mapped. You can also use the timestamps to plot a time series of duration of fault handler executions. These suggestions are not meant to be exhaustive – use your curiosity and creativity to get more insights about paging behavior Because page caching in memory will make handling of those faults when the page is in the cache fast while faults that require disk access will be slow, you should experiment with runs after a boot of your host and VM (cold cache) and successive runs without an intervening reboot (warm cache). Also note that when a page must be fetched from disk, the fault handler is usually called twice for the same faulting address. The first call includes the operations to read the file from disk into the page cache. The second call actually finishes mapping the page into page tables. These two calls can be distinguished because the first will not have assigned a page. They will also differ substantially in the elapsed time to the fault handler implementation.

In addition to submitting your program as usual, also submit a report that describes your experimental methods and gives the quantitative results you obtained from experiments. Give careful consideration to how you will test your module and run your experiments. There will be no demonstrations required for this assignment – only the program submission and report.