Assignment 5

Memory Management

Wiki Components Due: 1:00 pm, Monday, April 18, 2005
Final Patch Due: 11:00 pm, Wednesday, April 27, 2005

In this assignment you will be finishing the implemention a modern paged virtual memory system for OS/161. You begin with much of the work of implementing a VM system already complete. The system already handles page faults, manages the MIPS processor's TLB, and maintains mappings both from frames of physical memory to logical pages in programs and back again. It even supports swapping pages out to disk. The VM system is incomplete, however. Several important routines either just contain the comment "`/* NOT IMPLEMENTED */`" or consist of calls to `panic`. Your task is to complete those routines, which will also require you to understand the new codebase.

Preliminaries

Recall that you *must* have your path correctly set to undertake all CS 182 assignments. If you haven't done so already, put the appropriate line in your `.login` or `.bashrc`.

You no longer need your code from Assignment 4. You may delete your entire source tree (although you may wish to save your patch and any other changes for posterity).

For this assignment, check out and `setup` your `assign5` directory in the usual way, then configure and build the kernel using the `HW5` configuration files. You will also probably wish to create an additional configuration file, `HW5-RANDPAGE`, that turns on the configuration option for random page replacement.

Code Reading

In addition to the "kent" patch, which provides implementations of the MLF and Random schedulers, the codebase now contains an entirely new VM system. A human-readable version of the VM system patch is available on the course Wiki site. You should read over that patch and make sure that you understand its various components, which include

- Data structures that describe physical memory
- Data structures that describe the logical address space of processes
- TLB handling
- Page fault handling
- Page eviction
- Swapping pages to disk

You should make sure that you understand how all these components work together to provide a VM system. The codebase is discussed further in the exercises on the course Wiki.

Design and Implementation Requirements

In this assignment you will need to complete the VM system, allowing processes to allocate memory dynamically and have virtual memory larger than the amount of RAM physically present in the machine.

As usual, your code needs to be simple and easy to follow. You *may not* use any significant new data structures beyond the data structures provided—instead, where necessary, you should extend existing data structures in obvious ways.

Random Page Replacement

You should implement a version of `page_replace` that selects a random page for replacement. It is sensible to code and debug random page replacement before developing more complex page-replacement code.

Remember that you will need to use the `HW5-RANDPAGE` configuration file you created to build a version of the kernel that uses this algorithm.

LRU-Approximation Page Replacement

You should also implement page replacement based on the clock algorithm. In order to implement the clock algorithm, you will need to add additional fields to some of the VM data structures to track whether a given page has been referenced since the last time the clock hand has passed. Your algorithm should be tunable in some way so that you can experiment to see how varying parameters of the algorithm influences performance.

You do not need to worry about giving preference to clean pages over dirty ones.

Heap Support

Your should also extend the VM implementation to provide support for the heap by providing processes with an additional memory area (in addition to the code area, static data area, and stack) to hold heap data, and an `sbrk` system call to manipulate this area. In particular, the heap data area should be able to expand and contract.

You should ensure that the heap and stack cannot be contiguous in a process's address space (to prevent stack overflows from accidentally corrupting a large heap). You are free to choose how to implement this feature; you may either provide it in userspace or in the kernel.

Malloc Support

You should also write an implementation of `malloc` and `free` that uses the `sbrk` system call to request additional memory from the kernel and, when possible, release it. You may adapt code from CS 105 if you wish, but you should provide proper attribution if you do so.

Performance Analysis

Perform a qualitative and quantitative analysis of the performance of the page-replacement algorithms you have implemented and the TLB replacement algorithms. The provided code already tracks a number of performance measures that should help in making an assessment. You may add any other measures that you believe will help in evaluating your algorithms.

Choose some small number of VM-intensive programs and determine which of the algorithms makes them perform best, and, where applicable, how those algorithms can be tuned to achieve better performance. There are some programs in the `testbin` directory that may be helpful (e.g., `huge`) but you are also free to write or port your own tests.

Place your performance analysis in the file `performance.txt`.

Patch Submission

As usual, you will also need to submit a patch, which should contain sufficient documentation of your techniques for other members of the class to adequately understand your code. Your patch should be called `vm.patch`.

Going Further

There are several ways that you can go beyond the requirements of the assignment. If you would like to do so, please see me first to discuss the feasibility of your enhancements.