TCSS 422 — Computer Operating Systems
Winter 2015 — Homework Assignment 4

Due Date: Thursday, Feb. 26

Guidelines

Homework should be electronically submitted to the instructor by the end of the day on the due date. A submission link is provided on the course Canvas page for this assignment.

Assignment Description

This assignment is intended to strengthen your understanding of paged memory management. In this assignment you will complete the implementation of a simulator for managing the mapping between virtual address space and physical address space. Starter code for this assignment is available on the course Canvas page for this assignment.

Implementation Specifications

The provided `page_manager.c` source file contains skeletons for the two functions you're to implement, `initialize_page_manager` and `access_memory`. The `initialize_page_manager` function is called to set up the memory manager in a specific configuration, while the `access_memory` function is called to simulate a single memory operation at a specific virtual address. This simulation is of the memory system's address translation behavior, not the contents of main memory, caching, paging files, or other aspects of real memory managers. Consequently, this simulator doesn't include a simulated main memory nor differentiate between read and write memory operations.

The `main.c` source file provides a simple driver with four different simulations, i.e., memory configurations and memory access patterns. The macros `FIRST_SIMULATION_TO_RUN` and `LAST_SIMULATION_TO_RUN` select the range of simulations to execute. As is, only the first simulation is executed. The execution of a simulation generates output illuminating the internal operation of your memory manager, which may be helpful for debugging. You're encouraged to test your memory manager with the four provided simulations as well as create new simulations of your own.

`void initialize_page_manager(memory_config mc)`

The `initialize_page_manager` function receives memory configuration information necessary to simulate the internal operation of the memory manager. The information includes the sizes of the physical and virtual address spaces, the page size, and the number of processes that will be accessing memory (see `page_manager.h`). Since at least some of this information may be needed in the `access_memory` function as well, the provided code saves the configuration information in a global variable named `config`, should you wish to use it. You will undoubtedly need to add further initialization code to this function. After this function is called, the page tables of all simulated processes are considered to be empty (no valid entries) and no physical pages hold any virtual pages.

`access_result access_memory(unsigned int pid, unsigned int virtual_address)`

The `access_memory` function simulates a single memory operation, i.e., a specific process accessing a specific virtual address, based on the current state of the memory manager. PID numbers are non-negative integers in the range [0, $n - 1$], where n is the number of processes in the preceding call to `initialize_page_manager`. This function returns a structure containing information about the internal operation of the memory manager for that memory operation, including the virtual page number accessed, the physical page number that virtual page maps to, the complete physical address the virtual address maps to, and whether accessing that virtual page corresponded to a page fault (the page not existing in physical memory before the memory operation). Some of this information is not only needed to simulate the memory operation, but will also be displayed by the simulation framework in `main.c`. The majority of the memory manager will be implemented in this function (or helper functions, depending on how you organize your code).

The memory manager needs to implement the following two policies regarding page faults. 1) If one or more unused physical pages are available, the unused physical page with the lowest page number should be selected to hold the virtual page in question. For simplicity, all physical pages can be used by the memory manager (ignoring the reality that some portions of physical memory don't hold paged data). For example, at the beginning of a simulation, just after `initialize_page_manager` is called, the first page fault should result in physical page 0 holding the first virtual page accessed. 2) If no unused physical pages are available, a used physical page must be selected to hold the virtual page in question. The used physical page that belongs to the same process and was accessed least recently (the further back in the past) should be selected.

Deliverables

The following items should be submitted to the instructor. Failure to correctly submit assignment files will result in a 10% grade penalty.

1) The completed `page_manager.c` source file.
2) Any new supporting files, e.g., additional source code, needed by `page_manager.c`.

Do not include any extraneous files, such as Eclipse IDE files, object files, or subversion files.