Information and Analysis

Project 3: User-level Memory Management Library

Design

Libraries

The math.h, stdio.h, sys/mman.h, and pthread.h functions have been added to the header file to support the implementation of the project. As a result, '-lm' must be included when compiling for math.h. Per assignment instructions, the '-m32' flag is included as well.

Defines, Globals, and Structs

The page, memory, and TLB sizes are all defined in the header to allow easy modification.

Void pointers, like *va, are global pointers to keep track of the virtual address and page table structures. Unsigned long global variables hold the values of the address offset, VPN, and page directory information. A pde_t pointer is a global variable for the page directory that manages the pages. Global flag bools and ints are also used in many functions, relying on the correct use of locks to function correctly.

The tlb struct is comprised of an array of struct tlb_entry. Struct tlb_entry contains an unsigned long virtual and physical address. The tlb_entry arrayin struct tlb is of size TLB_SIZE, which is defined in the header.

Threads

In order to support threading, functions like m_alloc() and a_free() utilize mutex locks. The locks protect the critical sections of the code, which occur when accessing global virtual address pointer, va.

Implementation

M_alloc() first checks for initialization, if uninitialized SetPhysicalMem() is called. SetPhysicalMem() calculates the virtual memory address size and number of bits of the address needed for the offset and VPN using the header defined values for page size and size of the address space. Using these calculations, the pages and page directory are allocated and initialized. M_alloc() then calls get_next_avail() to return a pointer to the pages that are needed for the m_alloc request. M_alloc() calls PageMap() for each page that was found in get_next_avail(). PageMap() error checks the virtual and physical addresses before setting the page table entry and adding a new virtual address if necessary.

A_free() releases the valid memory pages used by the given virtual address, utilizing the function Translate() which takes the virtual address and returns the physical address—part of the page table functionality. This functionality includes the TLB which is checked in Translate()

for a TLB hit. If it is a miss, Translate() does the address translation, and updates the TLB and multi-level page table. If the address is in the TLB by the end of the function, a_free() removes the references to the address from the TLB to avoid invalid memory references.

MatMult() populates a result matrix using the functions GetVal() and PutVal(). GetVal copies the contents of the page at a given virtual address, PutVal() copies data to the physical address corresponding to the given virtual address. The result matrix is the sum of each corresponding row and column value added together, then stored as an element in the result matrix.

Multi-level Page Table

The multi-level page table is an array-like structure which add indirection, compact use of memory, and ease of implementation to page tables. The page directory contains page entries, in which the virtual and physical address lists are implemented with linked-lists.

Translation Look-aside Buffer

The page directory's "cache of sorts" was implemented using struct tlb and struct tlb_entry (see Defines, Globals, and Structs for more information). It is an array which holds the most recent memory references. The function Translate() utilizes the TLB by checking for a TLB hit before going to the page table. Translate() also adds the memory reference to the TLB to keep the TLB updated. A TLB hit increases the speed of the program by anticipating memory access patterns and providing a quicker access point when those patterns occur. Instead of double accessing in the multilevel page table, the TLB provides a single access for the memory reference. However, a TLB miss is more inefficient because it requires an additional memory access to update the TLB and the multi-level page table.
A_free also utilizes the TLB, see Implementation for more information. Helper functions check_in_tlb and put_in_tlb are used to find and manage the data in the TLB as well.

Method

Contributors

Dhaval Patel; dp918

Pre-planning phase: research lecture notes and relevant textbook chapters
Coding phase: m_alloc(), a_free(), get_next_avail(), Translate(), PageMap(), TLB[check_in_tlb and put_in_tlb]
Testing phase: test cases

Deanna Miceli; dm1053

Pre-planning phase: research lecture notes and relevant textbook chapters
Coding phase: SetPhysicalMem(), MatMult(), GetVal(), PutVal()
Testing phase: documentation, test cases