<u>Project 3 - User-level Memory Management (100 points)</u>

CS 416 - OS Design

DEADLINE: April 3rd, 2019, 11:59 pm

Assume you are building a new startup for Cloud Infrastructure (Amazing Systems); a competitor of Amazon Cloud Services. In the class, we discussed the benefits of keeping memory management in hardware vs. moving them to software. As the CEO of the company, you decide to move the page table and memory allocation to software. In this project, you will build a user-level page table that can translate virtual and physical address using multi-level page tables and reduce translation using TLBs. We will test out for different page sizes.

1. Description

While you must have used malloc () several times, you might not have thought about how virtual pages are translated to physical pages and how they are managed. The goal of the project is to implement ***"a_malloc()"*** (Amazing malloc) which will return a virtual address that maps to "physical memory." The physical memory refers to a large region of contiguous memory which can be allocated using mmap() call and provide your page table or memory manager as the illusion of physical memory. For simplicity, we will imagine that we have a 32-bit address space; hence that total memory that we can support is 4GB. The sizes of memory that will be used to test your implementation will be variable, so you should be very careful making the physical memory size configurable (a parameter to mmap). The project details are described in detail below:

set_physical_mem: This function would be responsible for allocating the memory buffer creating an illusion of physical memory using mmap (provided by Linux http://man7.org/linux/man-pages/man2/mmap.2.htm). Feel free to use malloc for allocating other data structures.

translate: This function will take input a page directory (address of the outer page table), a virtual address and return a physical address. You have to work with two-level page tables with 10 bits, 10 bits, 12 bits split. Return NULL if you don't find that address is allocated.

page_mapdir: This function will take a virtual and a physical address along with your root directory and store the physical address at the appropriate place in the page table.

a_malloc: This function would take the number of bytes you want to allocate and return a virtual address. Now, this part is a tricky but important part. Because you are using a 32-bit virtual address space, you are responsible for address management and you must keep track of what virtual addresses you have allocated and what you have not. To make things simple, assume that you do allocations in page granularity. For example,

Assume on your first call of a_malloc you return address is 0x1000. Now when you call a_malloc, you have the choice to return 0x1001 (if application asked for 1byte) or any of the above valid addresses that fall within page size. So simply keep track in terms of pages. In other words, after the first call, the next malloc will return 0x2000 or above. One thing to note about this simple management scheme is that all your virtual addresses are page aligned.

You will also have to keep track of what physical pages were allocated and for tracking, use bitmaps, where each bitmap will represent a page. Note that you must implement bitmaps efficiently (allocating only one bit for each page) to avoid space wastage (https://www.cprogramming.com/tutorial/bitwise_operators.html)

a_free: this call will take in a virtual address, and the number of bytes (int) and only free pages starting from the page of a given address until the page number for the number of bytes given. For example, a call of a_free(0x1000, 5000) will free two pages starting with virtual addresses 0x1000 and 0x2000 respectively. Also, you have to check if the call is for some address that hasn't been allocated yet. Even if a single of the pages in this range does not have a valid allocation, do not free any of them.

put_value: This function will take a virtual address returned by your library, and a virtual address which can be directly de-referenced in c (int x; &x => c knows about this) and put the contents of later in the former upto a given number of bytes which is your third argument. Again, you have to check for the validity of the library's virtual address.

get_value: This function will take the same arguments as the previous one but put the data in memory at your virtual address to the memory reference by the c interpretable address.

mat_mult: This function is for the testing of the above two functions. This will take two one dimensional arrays which will contain the contents of a square matrix, multiply them and store them inside a third memory location which is again passed as a library managed virtual address. The indexing for matrices in a single dimension array is as follows:
A[i][j] = A[(i * size_of_rows * value_size) + (j * value_size)]

BONUS (10 points): If you have finished the page table design, implement a direct-mapped TLB. The length for this TLB would be configurable and should be specified in the my_vm.h file as a TLB_SIZE constant. You will get bonus points only if the page table design works correctly with threading support and your TLB works correctly.

Important Note: Your code must be thread-safe and your code will be tested with multi-threaded benchmarks.

2. Suggested Steps

Step 1. Design basic data structures for your memory management library.
Step 2. Implementing set_physical_mem(), translate() and page_map(). Make sure they works.
Step 3. Implementing a_malloc() and a_free().
Step 4. Test your code with matrix multiplication.
Step 5. Implementing direct-mapped TLB with LFU for a bonus if Step 1 to 4 works fully for the multi-threaded case.

3. Submission

You need to submit your project 3 code and report in a compressed tar file on Sakai. You MUST use the following command to archive your project directory.

```
tar -zcvf <your_net_id>.tar.gz project3
```

4. Compiling and Benchmark Instructions

Please only use the given Makefiles for compiling. Also to compile the benchmark first, you have to compile the project code first code using the given Makefiles. Also, right now, benchmark will not display the correct results. You should also get some hints from the benchmark on how to test your code.

Please Note: Your grade will be reduced if your submission does not follow the above instruction.

*Created by Baber Khalid and modified by CS 416 staff.