Group member's names: Michael Zhang
NetIDs: mbz27
ILab Used: kill.cs.rutgers.edu

1 Logic:

a_malloc()

The function is responsible for allocating pages in the physical memory and returning the corresponding virtual address. It will check if there is enough space to allocate the memory in the physical memory space then it will allocate a physical page/pages that will accommodate the given number of bytes (num_bytes). Then it will get the next available virtual page and map the virtual page to the physical page address and it will do this for multiple pages if needed. It will finally return the starting virtual address.

a_free()

The function is responsible for releasing memory pages given a virtual address and the number of bytes to free. It will first check if the virtual address has an offset (if it does then it will not free the memory) and it will check if the virtual address(es) are free'd/valid. Then the function will iterate through all of the virtual addresses (there can be multiple virtual addresses since the given number of bytes to free can span multiple pages) and set the valid bits to 0 and free their respective physical pages. If the free operation leaves a page table empty then that page directory entry will be freed too.

set_physical_mem()

This function will initialize the physical valid bitmap, the physical memory, and will calculate the offset bits, page table bits, and page directory bits. It will also reserve/allocate page(s) for the page directory and set the valid bits for each page directory to 0.

translate()

The function will first check if the virtual address is in the TLB and returns the page directory address if it is present in the TLB. In the case the virtual address is not in the TLB the function will access the page directory entry (checks if the entry is valid) and get the page table index. Then it will access the page table entry (checks if the entry is valid) and constructs the page directory address with offset. It will add the virtual address and page directory address to the TLB and return the page directory address.

page_map()

The function takes a virtual address and maps a page directory address to it. The function will first check if there exists a page directory. If the page directory does not exist it will check if there is enough space to allocate a new page table and reserve it in physical memory. Then it will add the newly created page table to the page directory and map the page directory address to the corresponding page table entry that's represented in the given virtual address. Of course if the page directory exists then it will simply map the page directory address to the virtual address (and will check corresponding valid bits).

get_next_avail()
The function returns the beginning virtual page that proceeds the given number (num_pages) of contiguous virtual pages. It will iterate through all of the page tables entries beginning at virtual page address 0x0 and will check if there are enough contiguous virtual pages to accommodate the given number of pages (num_pages).

put_value()
The function copies the given number bytes (size) from a given pointer (val) to a given virtual address. It will first check if the virtual addresses are valid (there can be multiple virtual addresses since the given number of bytes can span multiple pages). Then the function will translate the virtual addresses into the physical page address (by using translate()) and will copy the data given by the pointer byte by byte (by casting the pointer to char* and will access it char by char) into the physical memory.

get_value()
The function will fetch the given number of bytes (size) from the given virtual address and will copy the bytes into the given pointer (val). It will first check if the virtual addresses are valid (there can be multiple virtual addresses since the given number of bytes can span multiple pages). Then the function will translate the virtual addresses into the physical page address (by using translate()) and will copy the bytes from the physical page address byte by byte (by casting the pointer to char* and will access it char by char) into the given pointer (val).

add_TLB()
Simply adds the virtual address and page directory address into the TLB and evicts the TLB entry if there is any existing entry. The index in which the TLB entry is placed is calculated by taking the modulus of the virtual address index and the number of TLB entries.

check_TLB()
Simply checks if the virtual address is in the TLB and if it's valid. If the virtual address is in the TLB the page directory address is simply returned and the number checks gets incremented. But if the virtual address is not in the TLB then it will return -1 and increment the number of checks and the number of misses.

print_TLB_missrate()
Simply calculates the miss rate which is the total number of misses over the total number of checks and prints out the rate.

mat_mult()
The function simply multiplies the two given matrices (mat1, mat2) and outputs the calculated values to the given output matrix (answer).

2 Benchmarks:

multi_test.c output - Miss rate 0.001230 or 0.122951%

Allocated Pointers:
0 2000 4000 6000 8000 c000 a000 e000 10000 12000 14000 16000 18000 1a000 1c000
initializing some of the memory by in multiple threads
Randomly checking a thread allocation to see if everything worked correctly!
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
Performing matrix multiplications in multiple threads threads!
Randomly checking a thread allocation to see if everything worked correctly!
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
Gonna free everything in multiple threads!
Free Worked!

test.c output - Miss rate 0.007500 or 0.750000%

Allocating three arrays of 400 bytes
Addresses of the allocations: 0, 1000, 2000
Storing integers to generate a SIZExSIZE matrix
Fetching matrix elements stored in the arrays
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
Performing matrix multiplication with itself!
5 5 5 5 5
5 5 5 5 5

5 5 5 5 5
5 5 5 5 5
5 5 5 5 5
Freeing the allocations!
Checking if allocations were freed!
free function works

3 Page Size Support

Yes my program supports different page sizes (in multiples of 4K)

README

The my_vm.c file needs to be compiled with the flag -lm to include the math functions. This might be the case with the benchmark makefile too.

Just to make sure I added the makefile for the benchmark program that I used to compile the c files.