Mourya Vulupala (MV638)
Naman Bajaj (NB726)
CS416 – Operating Systems
Project 3

1. Below is how we implemented each memory function:
   set_physical_mem: This function is called by t_malloc as a set up function. In this function, we initialize the physical memory, virtual and physical bitmaps, calculate the 1st level directory mask, the 2nd level directory mask, and offset bits, in addition to setting additional variables up.

   translate: We first check the TLB, if its present, we return. If not, then we walk through the page directory and return the physical address, if present.

   page_map: We again calculate the location of what we want mapped in our page directory, and initializing the table if it isn't present. We then set the virtual address corresponding to the physical address in the provided page directory.

   t_malloc: This is the heart of the program. We first check to see if the physical memory is initialized. We calculate the number of required pages, and get the address of the next available page. We do this until we don't need any more pages.

   t_free: We calculate the virtual page number from the virtual address given to us. We also calculate the number of pages we may need to free. We get the page table entry using translate, clear the bits in both of the bitmaps, invalidate the TLB entry, then check if the program is exiting.

   put_value: Hand off control to the perform IO function.

   get_value: Hand off control to the perform IO function.

   perform_IO: We implemented this helper function to help with put and get value, since the 2 functions were very similar. It's just a matter of calculating the physical address, and then writing the required number of bytes or reading the required number of bytes into or from the physical address, and returning if needed.

   mat_mult: Pre implemented

2. Benchmark output for part 1 seems to be working fine. In both test and mtest, using multiple different thread amounts, we were consistently seeing that "Free works" in both programs. In part 2, we noticed that with the default configuration of 512 TLB entries, we had a TLB miss rate of 18.6% in the single threaded program and 21.1%

in the multi threaded program. Below is a table of TLB entries we tested and their calculated miss rates in each program (plus any additional info).

	TLB_Entries	TLB Miss Rate (%)
./test	32, 64, 512, 1024	20, 20, 18, 18
./mtest (thread count = 5)	32, 64, 512, 1024	74, 69, 47, 20
./mtest (thread count = 10)	32, 64, 512, 1024	74, 78, 67, 20
./mtest (thread count = 25)	32, 64, 512, 1024	89, 85, 85, 20
./mtest (thread count = 50)	32, 64, 512, 1024	96, 91, 90, 40

3. We can confirm our code works with multiple page sizes. We implemented using the default 4096, but we tested with 8192, 12288, etc., and it was still working as expected.

4. One possible issue we were having was that sometimes, testing our program with multiple threads resulted in a segfault. This might still be a reoccurring issue, however we were able to somewhat mend it by ensuring that data is consistent across threads. However, we feel as if the problem still isn't fully fixed, as the fix we implemented is somewhat janky, so we fear there still may be a chance of a crash.

5. N/A

6. We mainly collaborated with each other, the project writeup, and Piazza. We occasionally used Google and StackOverflow for syntactic problems (such as multiple definitions), however they were not used for the actual implementation of our Virtual Memory System nor TLB.