CS416 Project 3 Report

Mario Barsoum (mwb96) and Ryan Lee (rsl82)

Project was tested to be fully functional and fulfilling all requirements on iLab machine cp.cs.rutgers.edu.

1. Detailed logic of how you implemented each API function and the scheduler logic.

- *void set_physical_mem()*

First, I initialized mutex with pthread_mutex_init(), memory with 1GB, bitmap for memories, and page directory. Bitmap was made of unsigned char* with the size of page number/8. For part 2, I initialized TLB bitmap, and bitmap structures.

- *add_TLB(void* va, void* pa)*

Find the unassigned TLB entry using TLB bitmap, and put in data of address into the TLB. If all entries are already assigned, evict the first TLB since there was no mention of the way of evicting TLB.

- *delete_TLB(void * va)*

Find the TLB that has data matches with va. It will be used at the free function.

- *pte_t* check_TLB(void* va)*

Find the pa from va in TLB. It will also update numChecks and numMisses variables to calculate missrate.

- *void print_TLB_missrate()*

Calculate the numMisses/numChecks and print it.

- *pte_t* translate(pde_t* pgdir, void* va)*

First, check if va is in TLB. From va, extract each layer's bits. Go through the page directory and page table, get pa and return it. If we could find the pa, return NULL.

- *int page_map(pde_t *pgdir, void* va, void* pa)*

Similar to translate, but we store pa to the page table. If successful return 1, if not return 0. I return 0 instead of NULL since the return type was int.

- *void* get_next_avail(int num_pages)*

Using virtual bitmap, find available pages to assign pages for malloc. If we could find the proper pages, return the index of the starting page. If not, return myNull=INTPTR_MAX.

- *void* get_next_avail2()*

Find the proper one page, to use it in the malloc using a physical bitmap. It is different from get_next_avail since it allocates real memory.

- *void* t_malloc(unsigned int num_bytes)*

if memory is not initialized, initialize it. calculate the number of pages from num_bytes. Find a virtual page with get_next_avail. And find pa to store the data, calculate virtual address with virtual page and physical memory. Store the physical address into the page table using page_map, set the bit to 1 of bitmaps and then return the virtual address.

- *void t_free(void*va, int size)*

Calculate inner and outer bit from virtual address. Set the page table to NULL value, clear the bit to 0 of bitmaps. If the page number is more than 1, do the same thing for all pages.

- *int put_value(void* va, void*val, int size)*

get pa from va using translate. Copy the data to pa using memcpy function. If the page number is more than 1, loop until all data are copied into pa.

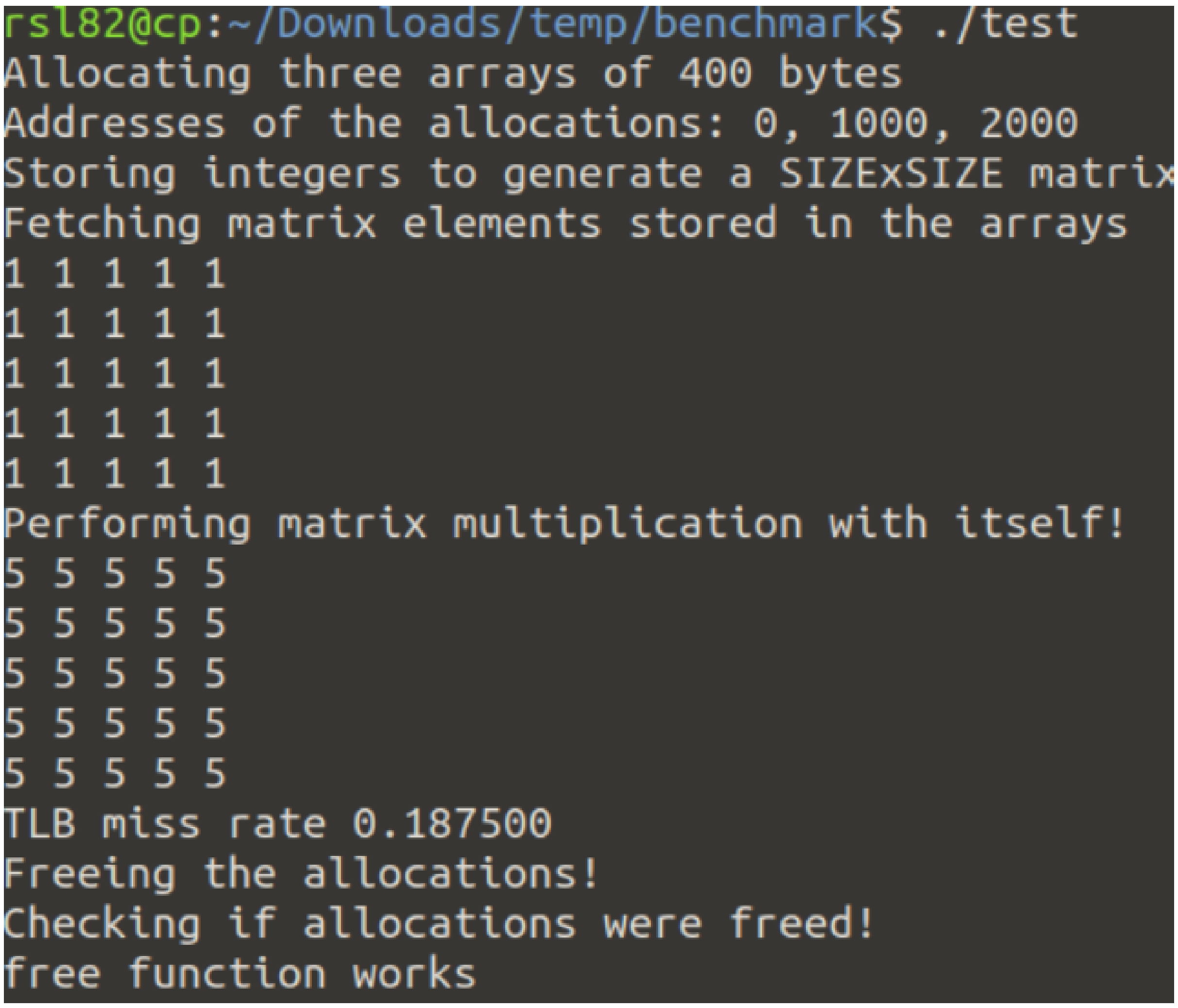
- *void get_value(void* va, void*val, int size)*

Basically, same as put_value. But copy the data from pa to val using memcpy function. I used a different approach to translate the pa from put_value, but it works the same.

- void mat_mult(void* mat1, void* mat2, int size, void* answer)

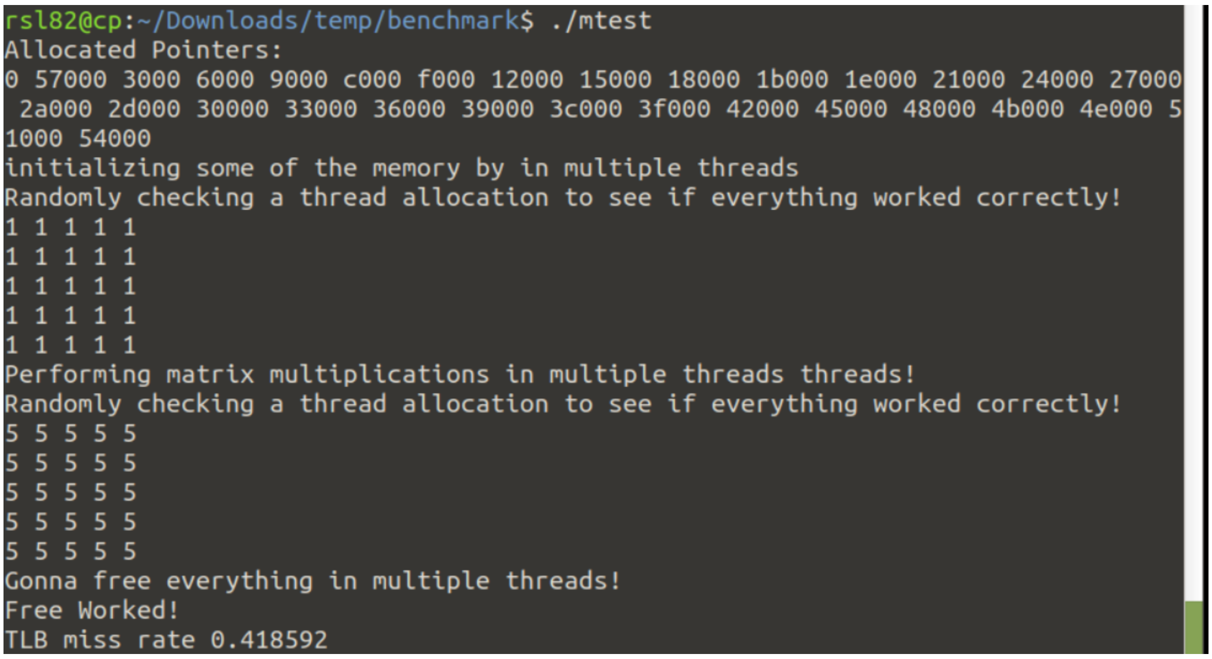
I used 3 loops to calculate the matrix multiplication. In the innermost loop, I get values from matrices using the get_value function and calculate the multiplication. At the middle loop, I put my calculation data into the answer pointer.

2. Benchmark output for Part 1 and the observed TLB miss rate in Part 2.



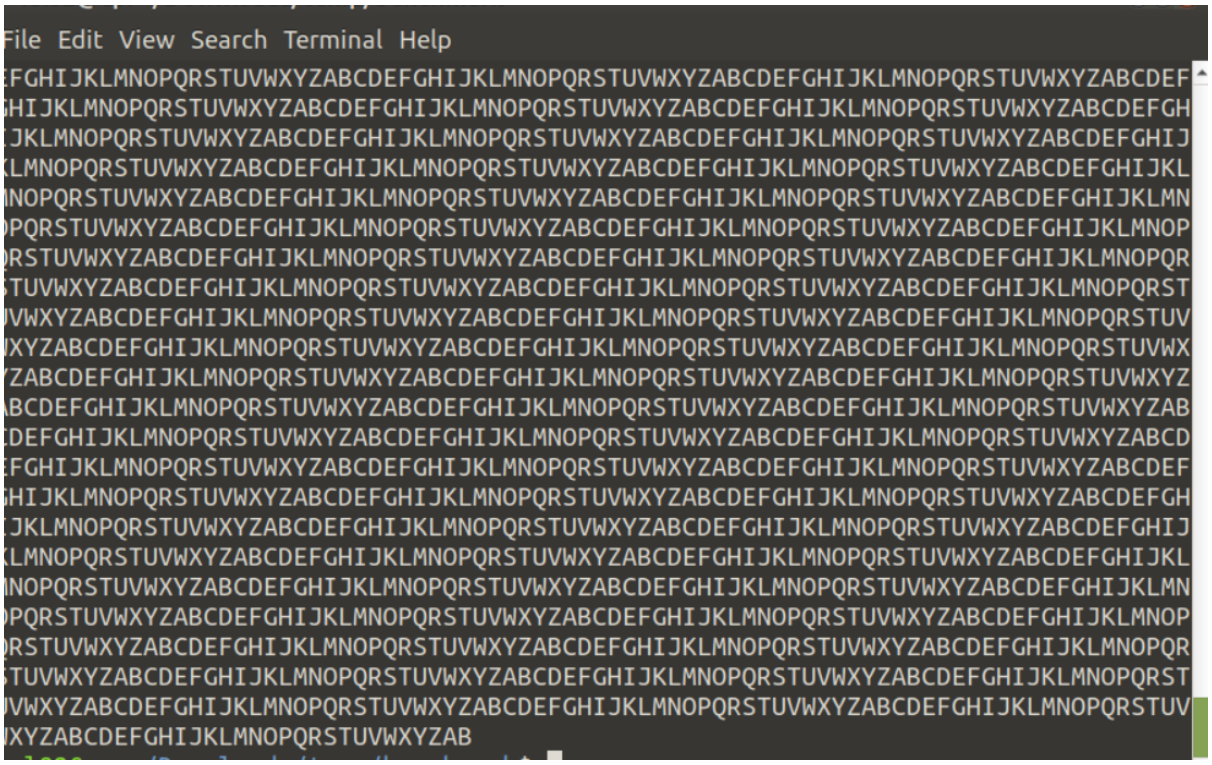
```
rsl82@cp:~/Downloads/temp/benchmark$ ./test
Allocating three arrays of 400 bytes
Addresses of the allocations: 0, 1000, 2000
Storing integers to generate a SIZExSIZE matrix
Fetching matrix elements stored in the arrays
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
Performing matrix multiplication 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
TLB miss rate 0.187500
Freeing the allocations!
Checking if allocations were freed!
free function works
```

This is my result for ./test. The multiplication worked as respected, and TLB miss rate was 0.1875. Free function also worked as I expected.



```
rsl82@cp:~/Downloads/temp/benchmark$ ./mtest
Allocated Pointers:
0 57000 3000 6000 9000 c000 f000 12000 15000 18000 1b000 1e000 21000 24000 27000
 2a000 2d000 30000 33000 36000 39000 3c000 3f000 42000 45000 48000 4b000 4e000 5
1000 54000
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
Performing matrix multiplications in multiple threads threads!
Randomly checking a thread allocation to see if everything worked correctly!
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
Gonna free everything in multiple threads!
Free Worked!
TLB miss rate 0.418592
```

./mtest also worked as I expected. One thing interesting about this is that the TLB miss rate has increased a lot even though I used the same function.



```
FGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEF
HIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGH
JKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJ
LMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKL
NOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMN
PQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOP
RSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQR
TUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRST
VWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUV
XYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWX
ZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ
BCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZAB
DEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCD
FGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEF
HIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGH
JKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJ
LMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKL
NOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMN
PQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOP
RSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQR
TUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRST
VWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUV
XYZABCDEFGHIJKLMNOPQRSTUVWXYZAB
```

This is ./mytest to see if the data larger than 1 page also works. This is 8k+1 numbers of characters and I copy it into t_malloc using put_value and get_value. It also worked and the pages worked as I expected.

3. Support for different page sizes (in multiples of 4K).

I tested 4 versions of PGSIZE: 4K,8K,12K,16K with my code. They all worked the same, and I could not find any problems with it. So, I assume my code also works for different page sizes.