Michael Elkhouri (Mre66), Akash Pathuri (Arp229)
Project 3 Report
User-level Memory Management
4/12/22
Machines: less.cs.rutgers.edu

Set_Physical_Mem():

This function is responsible for allocating memory for all data structures and initializing global variables for threads to use. We initialize bitmaps and bit pointers to help us throughout the system to calculate indices. We use mmap to allocate and initialize physical memory that we use throughout the project. We initialize the thread mutex to ensure thread safety for a multithreaded environment.

Translate():

Given a virtual address pointer and the page directory entry we can calculate the physical address using the global variables we initialized in set_physical_mem() to find the indexes of the entry we are looking for. The outer index is for the directory page level and the inner index is for the frame and using the frame, directory entry, physical memory, and PGSIZE the physical address is calculated. We then add the va to pa translation into the TLB to speed up translations in the future and return.

Page_Map():

Using similar methods from translate we traverse the page table checking if the table we are told to check from the arguments is empty or not. If it is the case we must initialize default values to each entry in the table and then set the index we want to the frame given from the physical address. Once the table is initialized, we put the physical address into the table.

T_Malloc():

To be thread safe, we lock the global mutex variable and then check if set_physical_mem has been called by any thread. Then we calculate the pages required given the bytes sent in by dividing and then checking the remainder for any extra pages to account for. Calling get_next_avail will give us the next available virtual address for the amount of pages we require. Calculating the inner and outer indexes we can check if the bit in the directory bitmap is empty or not. If it is, we are free to use it. Lastly, in a for loop we find the indexes of the free pages and we call page_map() to map the virtual addresses to physical addresses. Unlock the mutex and return va.

T_Free():

To begin freeing we need the amount of pages we are freeing, to calculate this we use the size passed in divided by the defined PGSIZE again checking for the remainder. Now looping through the page tables, we want to calculate the offsets, inner, and outer indexes to later use to get the individual frame. We set the value at frame we want to free equal to -1 to denote it is empty or cleared and then we call clear_bit to free. Unlock the mutex variable and repeat the loop. If there were no bit at the index we are freeing then that would point to an unused page and does not need to be freed.

Put_Value():

The thread locks the mutex and then calculates offset and translates the virtual to a physical address. If the va is invalid we unlock and return. If the size of val is enough to fit in what is remaining of the page then we fit it in otherwise we fit as much as we can and traverse the address space to fit the rest. The loop runs till all the full pages are copied and exits. Finally any remaining partial page memory is written to the value and return with a mutex unlocked. The coping of memory is done using memcpy from the value given to the designated physical address.

Get_Value():

We repeat almost every step identical to put value however the memcpy statements have a reverse order. We are storing the value from pa into the variable “val” instead of the opposite like in put_value.

TLB & Debugging:

We designed the TLB to hold virtual address and physical address void pointers of course for quick translation of commonly used addresses. We included an “age” attribute to denote whether or not the entry in the tlb is not being used as often as it should be, specifically for the eviction policy that was implemented in add_tlb. We had the freedom to choose what data structure and we chose to keep it simple and go with an array of structs. Linked lists were giving issues so we aborted that approach. We began to come across errors once the TLB was implemented due to slight errors in loop logic and pointer traversal. These were fixed and we encountered no more problems with the test benchmark. However we soon realized that our program was not multithread safe so we began debugging and found that the issue resided when we initialized the mutex variable for the first time. We saw that multiple threads were attempting to run set_physical_mem() multiple times so we added a base case where a thread can only call set_physical_mem if and only if physical_memory was null. This solved our problem. Lastly we tested different page sizes (multiples of 4096) to see if any issues came but we coded in a way where PGSIZE was dynamically involved.

BenchMarking results

Test using 4kb page:

```
Allocating three arrays of 400 bytes
Addresses of the allocations: 1000, 2000, 3000
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
TLB miss rate 0.013333
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
```

Multi Test using 4 kb page and 45 threads:

```
Allocated Pointers:
1000 7000 4000 a000 d000 10000 13000 16000 19000 1c000 1f000 22000 25000 28000
 2b000 2e000 31000 34000 37000 3a000 3d000 40000 43000 46000 49000 4c000 4f000
 52000 55000 58000 5b000 5e000 61000 64000 67000 6a000 6d000 70000 73000 76000
 79000 7c000 7f000 82000 85000
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 0.008491
```

Test using 8kb page:

```
Allocating three arrays of 400 bytes
Addresses of the allocations: 2000, 4000, 6000
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 1 1 1 1 1
1 1 1 1 1 1
TLB miss rate 0.009259
Performing matrix multiplication with itself!
6 6 6 6 6 6
6 6 6 6 6 6
6 6 6 6 6 6
6 6 6 6 6 6
6 6 6 6 6 6
6 6 6 6 6 6
Freeing the allocations!
Checking if allocations were freed!
free function works
```

Multi Test using 8kb page and 45 threads:

```
Allocated Pointers:
2000 e000 12000 6000 a000 16000 1a000 1e000 22000 26000 2a000 2e000 32000 3600
0 3a000 3e000 42000 46000 4a000 4e000 52000 56000 5a000 5e000 62000 66000 6a00
0 6e000 72000 76000 7a000 7e000 82000 86000 8a000 8e000 92000 96000 9a000 9e00
0 a2000 a6000 aa000 ae000 b2000
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 0.008491
```

Test using 16kb page:

```
Allocating three arrays of 400 bytes
Addresses of the allocations: 4000, 8000, c000
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 1 1 1 1 1
1 1 1 1 1 1
TLB miss rate 0.009259
Performing matrix multiplication with itself!
6 6 6 6 6 6
6 6 6 6 6 6
6 6 6 6 6 6
6 6 6 6 6 6
6 6 6 6 6 6
6 6 6 6 6 6
Freeing the allocations!
Checking if allocations were freed!
free function works
```

Multi Test using 16kb page and 45 threads:

```
Allocated Pointers:
10000 4000 18000 14000 8000 c000 1c000 20000 24000 28000 2c000 38000 30000 340
00 3c000 40000 44000 48000 4c000 50000 54000 58000 5c000 60000 64000 68000 6c0
00 70000 74000 78000 7c000 80000 84000 88000 8c000 90000 94000 98000 9c000 a00
00 a4000 a8000 ac000 b0000 b4000
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 0.008491
```