Project 3: User-level Memory Management

CS416 – Operating System Design

Professor Badri Nath

Project Report

Sami Munir (sm2246)

Kelvin Sorto (kas739)

I. Detailed Logic of Virtual Memory Library Functions

- ***set_physical_mem():*** this function uses *malloc()* to allocate a large contiguous block of memory that represents the physical memory available for page allocation.
- ***translate():*** this function performs a two-level page table lookup to translate a virtual address to a physical address.
  - We calculate the page directory and page table indices based on the virtual address and page size. This involves using bit manipulation to isolate the relevant bits for the page directory index, page table index, and offset within the page.
  - Once we have the indices, we use them to access the corresponding entries in the page directory and page table.
  - The page directory entry points to the relevant page table, and the page table entry holds the physical page frame number for the virtual page.
  - For part 2, we check the TLB for the translation before resorting to the page table lookup for faster access.
- ***page_map():*** this function walks the page directory to check if a mapping already exists for the provided virtual address.
  - If no mapping exists, it allocates a new page table (if needed) using memory allocated by *set_physical_mem()*.

    - It updates the page directory entry and the bitmaps to reflect the new mapping. The bitmaps keep track of which physical pages are free and allocated.
- ***t_malloc():*** this function allocates virtual memory using *malloc()*.
    - It calculates the required number of pages based on the allocation size and page size.
    - It uses *get_next_avail()* to find free physical pages from the bitmaps and updates the bitmaps to mark those pages as allocated.
    - It creates entries in the page table to map the allocated virtual address space to the corresponding physical pages.
- ***t_free():*** this function frees virtual memory using *free()*.
    - It iterates through the virtual pages to be freed based on the provided arguments.
    - It clears the corresponding entries in the page table to remove the mapping.
    - It updates the bitmaps to mark the freed physical pages as available for future allocation.
    - It handles freeing non-contiguous physical pages by iterating through the virtual address range and updating the page table and bitmaps accordingly.
- ***put_value():*** this function stores data in the virtual memory.
    - It translates the virtual address to a physical address using *translate()* or by checking the TLB.
    - It performs error checks (e.g., invalid virtual address) before copying the data using *memcpy()* to the corresponding physical memory location.
- ***get_value():*** this function retrieves data from the virtual memory.
    - It translates the virtual address to a physical address using *translate()* or by checking the TLB.
    - It performs error checks (e.g., invalid virtual address) before reading data using *memcpy()* from the corresponding physical memory location.
- ***mat_mult():*** this function performs matrix multiplication on the given parameters.

- It uses nested loops to iterate through the elements of the matrices stored in virtual memory.
- Inside the loops, it uses *get_value()* and *put_value()* to access and update elements within the matrices based on their virtual addresses.

II. Support for Different Page Sizes

- The number of bits used for the offset will vary depending on the page size $\log_2$[page size].
- The remaining bits in the virtual address will be used to calculate the page directory and page table indices based on the chosen page size.

III. Possible Issues/Drawbacks in our Implementation

- Internal fragmentation: Our code might not handle internal fragmentation efficiently. For example, allocating a full page for a small data element can waste memory (not required per project description).
- Thread safety: If our code is not thread-safe, then it could lead to race conditions when multiple threads access the shared memory structures (page tables, bitmaps, etc.) concurrently.

IV. Testing & Makefile

We used *test.c* and *Makefile* to compile, run, and test our program. We used the *main()* function in *test.c* to call each of the custom function in *test.c* which tested corresponding library functions in *my_vm.c*.

```
> make clean
> make
> ./test
```

V. Important Notes

- We noticed that the *translate()* function works just fine on its own, meaning it translates the right address contents. However, when it used within function *mat_multi()* it does not work correctly (does not translate the right address contents).
  - This may result in the inaccuracy of functions *put_value()* and *get_value()*.
- We added a *pthread lock* to only the *mat_mult()* function (assuming this is what we will be tested on using threads). This is because if it was to be added

to the rest of the functions, it could result in a potential deadlock.

VI. Collaboration & References

- Project write-up (CS416 Canvas)
- Class slides/notes (CS416 Canvas)
- CS416: Project 1 (threads & bit manipulation)
- Virtual memory address translation (YouTube)