Asst3 Documentation
Amali Delauney (adj298) Shyam Patel (spp12)
CS 416
Sudarsun Kannan

Execution

Use the makefiles: One is for the benchmark and other for the my_vm.* files

Structures

TLB Our mutex contains the following components:

- **int mem_accesses**: Simply refers the number of times the TLB has been accessed. It is used for calculating the miss rate
- **int miss_count**: A count of all the TLB misses
- **int page_dir_nums[TLB_ENTRIES]**: An int array of the entries in the page directory that are in the TLB
- **unsigned long physical_addrs[TLB_ENTRIES]**: An array of the physical addresses which directly map to a TLB entry
- **short int age[TLB_entries]**: It refers to how many times a given TLB entry was not needed or simply its age

Globals

- **struct tlb tlb_store**: It's our tlb
- **pde_t* page_dir**: It's our page directory
- **unsigned char* physical_mem**: It's our physical memory
- **pthread_mutex_t pt_lock, vbitmap_lock, pbitmap_lock, tlb_lock**: Mutexes for the shared resource as specified in each of their names
- **int page_entries**: The number of entries in a page
- **int page_count**: Our page count in physical memory
- **int ptable_count**: Our second level pagetable count
- **valid_bit* vbitbap, pbitmap**: Simply our bitmaps for our virtual and physical pages, respectively

Functions

- **void set_physic_mem()**: Mallocs physical memory, page directory and virtual and physical bitmaps
- **int add_TLB(void* va, pte_t* pa)**: Adds an entry to the TLB. Removes oldest entry if TLB is full.
- **pte_t* check_TLB(void* va)**: Checks the TLB for a specific entry. Returns physical address if found.
- **void print_TLB_missrate()**: Prints TLB missrate
- **pte_t* translate(pde_t* pgdir, void* va)**: Takes a virtual address and returns the respective physical address.
- **unsigned int get_top_bits(unsigned int value, int num_bits)**: Simply retrieves the top bits of an address
- **unsigned int get_mid_bits(unsigned int value, int num_middle_bits, int num_lower_bits)**: Simply retrieves the middle bits of an address
- **unsigned int get_end_bits(unsigned int value, int num_bits)**: Simply retrieves the last few bits of an address
- **static void set_bit_at_index(char* bitmap, int index)**:
- **static int get_bit_at_index(char* bitmap, int index)**:
- **void update_pbitmap(int index)**: Flips the value, at index, in the physical bitmap
- **void update_vbitmap(int index)**: Flips the value, at index, in the virtual bitmap
- **int page_map(pde_t* pgdir, void* va, void* pa)**: Given the inputs it maps the virtual address to a virtual page and that virtual page to a physical page
- **void* get_next_avail_vp(int numpages)**: Returns the first index of **numpages** consecutive virtual pages in **vbitmap**
- **void* get_next_avail_pp()**: Returns the index of the first free physical page in **pbitmap**
- **void* a_malloc(unsigned int num_bytes)**: Given the **num_bytes** it will find and map available physical and virtual pages and return a virtual address
- **void a_free(void* va, int size)**: It will clear the allocated virtual pages, physical pages and TLB (if the **va** is present in the TLB) for **size** bytes
- **void put_value(void* va, void* val, int size)**: Takes a value of **size** bytes and puts **val** in **va**'s respective place in physical memory
- **void get_value(void* va, void* val, int size)**: Simply, goes to **va**'s place in physical memory and copies everything from **va**'s spot in memory and the next **size** bytes into **val**
- **void ma_mult(void* mat1, void* mat2, int size, void* answer)**: Multiplies matrices **mat1** and **mat2** and stores the result in **answer**

Design Logic

Our Multilevel Table

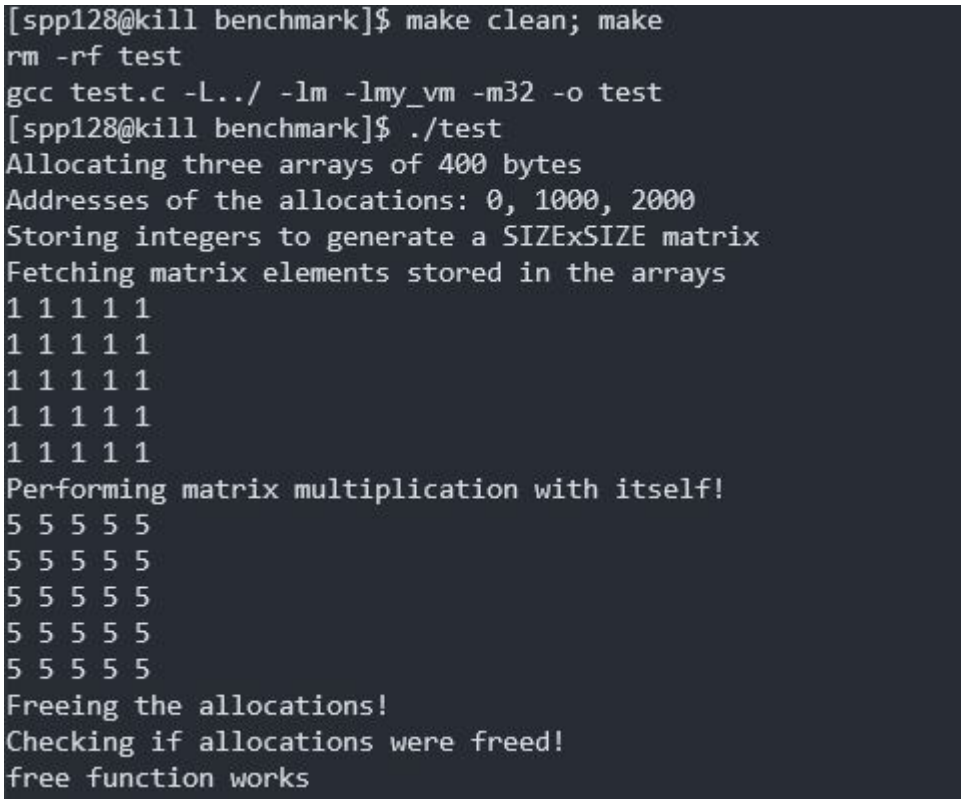
Any given entry in **page_dir** will point to a **pte_t*** which represents the start of a virtual page of size **PGSIZE**. Each entry within this page table will point to a specific place in **physical_mem**. Whenever a shared resource is accessed or modified, we call invoke a mutex for that shared resource to prevent any race conditions.

Our TLB

For our eviction policy, we are using LRU. We keep track of the age of every entry via an array of short integers. Whenever something is needed to be evicted, we replace the entry with the largest age with what needs to be added

Benchmark

When running the benchmark we receive consistent results in the matrix multiplication. When the results of the multiplication is complete, we are presented with a 5x5 matrix of all 5's. The image below shows the complete output of what occurred when running the benchmark.



Possible Issues

One possible issue we face is increased fragmentation due to the design of requiring contiguous virtual pages and either contiguous or non contiguous physical pages. One other possible issue

is if **a_free**, **get_value** or **put_value** were to be called with regards to something that spans more than 1 physical page, we may hit a segfault.