User-level Thread Library and Scheduler

Akshith Dandemraju (Acd218), Abhinav Acharya (Aa2372)

April 10, 2024

Detailed Logic of Virtual Memory Functions Implementation

1. **set_physical_mem():**
   - Allocates physical memory using malloc based on the product of the page size (PGSIZE) and the number of frames (num_frames).
   - Initializes the physical memory using memset to zero and sets up a bitmap (physical_memory_bitmap) with 1 bit per frame to track free/used frames in physical memory.
   - Calculates the number of bits required for the page offset, page table, and page directory based on the page size and total addressable bits (32 bits).
   - Allocates memory for the page directory in physical memory and initializes it.
   - Initializes the virtual memory bitmap to track mapped/unmapped virtual addresses.
   - Allocates memory for the TLB (Translation Lookaside Buffer) to store virtual to physical address mappings.
2. **get_page_table(page_directory_entry *pd):**
   - Checks if the page table for a given page directory entry (pd) is already allocated.
   - If not allocated, it searches for a free frame in physical memory using the physical memory bitmap.
   - Allocates and initializes the page table in physical memory and updates the physical memory bitmap.
   - Returns a pointer to the page table.
3. **translate(unsigned int vp):**
   - Checks the TLB for the virtual page (vp) to physical page (pp) mapping. If found, directly returns the physical memory address.
   - If not in TLB, calculates the page directory index, page table index, and page offset from the virtual address.
   - Checks if the virtual address is mapped in the virtual memory bitmap.
   - Retrieves the frame number from the page table and calculates the physical address.
   - Adds the virtual to physical mapping to the TLB for future lookups.
   - Returns the physical memory address corresponding to the virtual address.
4. **page_map(unsigned int vp, int size):**
   - Maps a virtual page (vp) to a physical frame in memory.
   - Calculates the page directory index and page table index from the virtual address.
   - Searches for a free frame in physical memory using the physical memory bitmap.
   - Updates the page table entry with the frame number and sets the virtual memory bitmap accordingly.
   - Returns 0 if successful, 1 if no free physical memory is available or page is already mapped.
5. **t_malloc(size_t n):**

- Allocates a contiguous block of virtual memory of size n.
- Checks for available free pages in virtual memory using a helper function.
- Maps the required number of pages to physical frames using page_map().
- Returns the starting virtual address of the allocated memory block.

6. **t_free(unsigned int vp, size_t n):**
   - Frees a contiguous block of virtual memory starting from virtual address vp and of size n.
   - Checks if the pages are valid and mapped in virtual memory.
   - Clears the corresponding bits in virtual memory bitmap and physical memory bitmap.
   - Returns 0 if successful, -1 if the virtual address is not mapped or invalid.
7. **put_value(unsigned int vp, void *val, size_t n):**
   - Writes a value to a contiguous block of virtual memory starting from virtual address vp and of size n.
   - Checks if the virtual address is valid and mapped in virtual memory.
   - Translates the virtual address to physical address using translate().
   - Copies the value to the physical address using memcpy().
   - Returns 0 if successful, -1 if the virtual address is not mapped or invalid.
8. **get_value(unsigned int vp, void *dst, size_t n):**
   - Reads a value from a contiguous block of virtual memory starting from virtual address vp and of size n.
   - Checks if the virtual address is valid and mapped in virtual memory.
   - Translates the virtual address to physical address using translate().
   - Copies the value from physical address to destination using memcpy().
   - Returns 0 if successful, -1 if the virtual address is not mapped or invalid.
9. **mat_mult(unsigned int a, unsigned int b, unsigned int c, size_t l, size_t m, size_t n):**
   - Implements a basic matrix multiplication using virtual memory addresses a, b, and c.
   - Checks if the virtual addresses are valid and mapped in virtual memory.
   - Reads values from virtual addresses a and b, performs matrix multiplication, and writes the result to virtual address c.
10. **add_TLB(unsigned int vpage, unsigned int ppage):**
    - Adds a virtual to physical address mapping to the TLB.
11. **check_TLB(unsigned)**
    - Check if the TLB has the vp with its physical frame.
    - If it exists increment hits and return 0. If it doesn't increment misses and return -1.

Support for different page sizes (in multiples of 8K).

1. My program works with different page sizes by calculating the page dir bits and page table bits at runtime.
2. page offset bits are calculated using log base 2 of the page size. (defined in header file)
3. page table bits would be calculated by log base 2 of the page size / sizeof(page_table_entry)
4. page directory bits are the rest of the bits remaining from our 32 bit address.

Possible issues in your code (if any).

My code does not have any page replacement and there is going to be internal fragmentation.

Collaboration and References: State clearly all people and external resources (including on the Internet) that you consulted. What was the nature of your collaboration or usage of these resources?

We used help from the TAs and Piazza.