CSCE 611: Operating Systems

Fall 2022

Machine Problem 4: Virtual Memory Managment and Memory Allocation

Name: Rajesh Satpathy

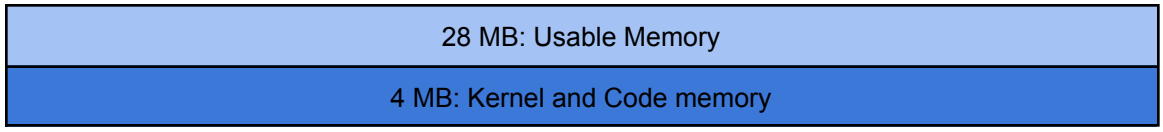
UIN: 931006158

Goal:

Implement virtual memory management and memory allocation system with Page Table Entry and Page Directory Entry lookups for the implemented paging system.

Assumptions shared about the given Kernel:

- Total machine memory: 32 MB
- Memory Layout:



- About the first 4 MB:
  - Is reserved for the kernel.
  - The first 1 MB address space is reserved for IDTs, GDTs, video memory, etc.
  - The address range from 1MB to 2MB consists of kernel code and stack space.
  - The kernel frame pool that is implemented in the mp2 assignment is located between 2 MB and 4MB.

Scope

- Page Table management is extended to support larger numbers and sizes of address spaces.
- Process memory pool's virtual memory is utilized for page tables instead of the kernel memory pool's direct address references.
- Virtual memory pool with new and delete capabilities is introduced.

Implementation:

The paging system is updated to support virtual memory with a process memory pool within page_table.C. Updated and new functions with their changes are mentioned below:

- **Constructor:** Initializes a page table with the first frame pool from the process frame pool and initialize the vmpool size as 0.
- **handle_fault:** Handles page faults when the page requested is not present in page table or if a page is not present at index. The fault is handled here by finding the legitimate page in the vmpool and then referring the entries using Page Tabel Directory and Page Table Entries at the address of the thrown page fault. Entries are made after the resolution of PTE and PDE into physical address indices.
- **PDE_address:** Sends the pointer to the Page Directory entry.
- **PTE_address:** Sends the pointer to the Page Table entry.
- **free_page:** Frees the page at the given page number by referring the PTE, PDE entries, and page by | X:10 | Y:10 | offset:12 |
- **register_pool:** Fill the vmpool instance at the vmpool list entry.

The virtual memory pool is implemented to support a larger number of pages. The implementation and functions within vm_pool.c is as follows:

- **Constructor:** Initializes the data structures needed for managing this virtual-memory pool.
- **allocate:** Allocates a region of _size bytes of memory from the virtual-memory pool and sends the start of the allocated region of memory, returns 0 if unsuccessful.
- **release:** Releases previously allocated memory region, given the start address of the region.
- **is_legitimate:** Checks if the address is within the virtual memory pool size and returns a boolean based on it.

Design Decisions:

The design decisions are made as follows:

1. A max size of 100 is provided for the virtual memory pool list. This is a static variable that can be changed at page_table.H
2. The size of vm_pool is tracked using a vmpool_size variable to reduce the number of iterations when the number of entered vm_pools is small. Is defined at page_table.H
3. A struct, alloc_space for allocating regions is introduced which holds the values for the allocated region's start address and size defined in vm_pool.H
4. The index of the allocated spaces is kept track in space_iter defined in vm_pool.H
5. The max allocated space is defined as 256 at vm_pool.C

Output from testing: