CMPSCI 577 Operating Systems Design and Implementation **Spring 2020**

Lecture 11: Minix Memory Management

Lecturer: ***Prashant Shenoy*** *Scribe:* ***Huan Wang***

11.1 Process and Memory Management

Memory management is tied to process management since processes are allocated memory and use memory. Minix functionality has evolved over time in terms of memory management support. For pre-3.1 minix, there is no separate memory manager, and we assume that a process is a collection of (stack, text, heap) segments. Each segment is allocated contiguously in RAM and memory is treated as a collection of holes and allocated segments.

Pre-3.1 Minix does not support paging, virtual memory, and demand paging. The reason for that is more sophisticated memory management techniques need hardware support. Pre-3.1 Minix is designed for older CPUs which do not support paging, so it lacks hardware support.

Post-3.2 Minix supports virtualization, paging and demand paging. Moreover it has a new VM (virtual memory) server separated from PM. We assume processes segments are contiguously laid out in virtual memory but no contiguous assumptions on physical memory. Post-3.2 Minix essentially uses segmented paging with virtual memory, which is discussed later in this lecture.

11.2 VM Server

VM manages memory (keeping track of used and unused memory, assigning memory to processes, freeing it, ..). There is a clear split between architecture dependent and independent code in VM. Virtual region, physical region and disk cache are three of the important data structures used in VM.

Virtual region : A virtual region is a contiguous range of virtual address space that has a particular type and some parameters. Virtual regions have a fixed-sized array of pointers to physical regions in them. Every entry represents a page-sized memory block. If non-NULL, that block is instantiated and points to a physical region, describing the physical block of memory.

Physical region : Physical regions exist to reference physical blocks. Physical blocks describe a physical page of memory.

Disk cache : holds pages of disk blocks.

11.2.1 Call Stucture

VM gets calls from user space (allocating memory for heaps), PM (fork exit) and the kernel. A typical flow of control is (1)Receive message in main.c (2)Do call-specific work in call-specific file, e.g. mmap.c, cache.c (3)Update data structures (region.c) and pagetable (pagetable.c). An example is mmap (memory map), which is a system call that maps a file into memory.

11.2.2 Handling Absent Memory

There are situations in which processes try to access a page or memory address that is not in memory, so we get a page fault. VM can handle absent memory. The two cases are (1)pagefault in anonymous memory (2)pagefault in a file-mapped region, in this case VM will query cache else go to VFS.

11.2.3 Bitmaps and Linked List

We can use two data structures, bitmaps or linked list, to keep track of used and unused memory

11.2.4 Memory Allocation Algorithms

First fit: Use first hole big enough
Next fit: Use next hole big enough
Best fit: Search list for smallest hole big enough
Worst fit: Search list for largest hole available
Quick fit: Separate lists of commonly requested sizes
Early Minix used these method for physical memory allocation. Later Minix versions uses holes and allocation for allocating a process in virtual memory.

11.3 PM Server

PM (process manager) can share text segment across processes (for starting multiple copy of the same process, shared library), so the OS doesn't have to load it twice.

PM implement some system calls such as fork and exec that interact with memory management. The three important calls in VM are alloc_mem, free_mem, and mem_init. They are used for requesting a block of memory of given size, returning memory that is no longer needed, and initializing free list when PM starts running. One thing we need to be careful When we allocate memory is we should fill the memory block with zeros or rewrite random bits for security reason, so other processes cannot see what was in that memory region that was put in there by a previous process.