David Merrick

CS 411

20 May, 2013

Individual Writeup for Project 4

1. *What do you think the main point of this assignment is?*

   The main point of this assignment is to gain a better understanding of how memory allocation in the kernel works and the advantages/disadvantages of the algorithms associated with doing that. The point is also to learn how to use system calls.

2. *How did you approach the problem? Design decisions, algorithm, etc.*

   Background:

   The memory management layer is the part of the kernel that services all memory allocation requests. To handle memory requests that are less than a full page, the kernel currently gives a choice of three different allocators: SLAB, SLUB, and SLOB. SLAB and SLUB (the most recent of these) are complex allocation frameworks for use in resource-rich systems (desktop computers, for example). They are designed to reduce internal fragmentation of memory and to permit efficient reuse of freed memory.

   SLOB (Simple List Of Blocks), on the other hand, is designed to be a small and efficient allocation framework for use in small systems such as embedded systems. Unfortunately, a major limitation of the SLOB allocator is the high degree to which it suffers from internal fragmentation. One likely cause for SLOB's high fragmentation rate is the fact that it uses a simple first-fit algorithm for memory allocation. SLOB is capable of allocating as little as 2 bytes, but most architectures will require at least 4 bytes (32-bit) and 8 bytes (64 bit). SLOB allocates using a heap. Inside this heap is a linked list of pages. To allocate from the heap, the algorithm searches for a page with sufficient free blocks (next-fit-like approach) followed by a first-fit scan of the page. To deallocate, SLOB inserts objects back into the free list in address order (effectively an address-ordered first fit). The SLOB heap is a linked list of pages. Heap pages are segregated into 3 separate lists. One list contains objects less than 256 bytes, another contains objects less than 1024 bytes, and the third contains all other objects.

   Memory fragmentation. Memory fragmentation is the inability to find large contiguous chunks of available memory. Internal fragmentation is when the unusable memory is within an allocated region (in this case, a page).

   Design and Implementation: The default implementation of SLOB is the first-fit approach mentioned in the "Background" section. Our task was to implement a best-fit algorithm. This meant that, for each request, the SLOB allocator had to find the smallest block of memory that it would fit in. We decided that we could do this by looping through all of the requests with a temp pointer. This pointer would hold the current smallest block of memory that would fit the request. If a block of memory was found that would fit the current request and was smaller than this block, the temp pointer would be reassigned to reference this new block. This algorithm would guarantee that the temp pointer was always referencing the smallest block of memory that the request would fit in. Once outside the loop, the request would be stored in the memory pointed to by temp.

   In order to get the new kernel to use SLOB, we had to modify the .config file in the kernel directory. We changed CONFIG_EMBEDDED to "y" (SLOB is designed for embedded systems) and added CONFIG_SLOB=y near where CONFIG_SLAB was.

   Testing:

   We decided to test the fragmentation of our new slob best-fit algorithm by defining system calls for determining the number of SLOB blocks claimed and the number of free blocks. We could use these to determine the amount of fragmentation suffered by SLOB. We got this idea from Paul Paulson's instructions for his class when he taught this class (http://classes.engr.oregonstate.edu/eecs/fall2011/cs411/proj02.pdf). We created two system calls. The first, "sys_get_slob_amt_claimed," returned the amount of memory not served by an allocation request. We did this by multiplying the number of used slob pages by the number of slob units per page. The second system call, "sys_slob_free," returned the number of free units within each used page. We did this by looping through each of the three linked lists of slob pages and adding the free units to a running total. We were able to use both of these system calls to calculate the internal fragmentation by dividing the number of free units by the total number of units.

We assumed that "efficiency" meant the time complexity of each algorithm. We decided to test this by timing some malloc() calls for varying sizes of memory and printing the average number of microseconds they took. Because the best-fit algorithm iterated through all of the slob pages while the first-fit did not, we predicted that the best-fit would be less efficient by these standards. And this is exactly what we found.

Defining System Calls. To do this, we first had to learn how to define system calls. Fortunately, there was an entire section in the book dedicated to this on page 78. The steps involved are as follows: write the system call as a function. Add an entry for this system call in the end of the system call table for each architecture that supports it. This table is located in asm/unistd.h. Give the system call a number for each architecture. This number is used to invoke the system call later. Finally, compile the system call into the kernel image. It can then be invoked with syscall(syscall_number), where syscall_number is the number it was defined with.

3. *What did you learn?* I learned how to write system calls. I learned about various memory allocation schemes in the kernel. I re-learned how to do microsecond timing of operations and function calls.