CS411 Project 4

Trevor Bramwell, Brian Cox, Matthew Okazaki

Abstract

This document describes the design, implementation, testing, and obstacles faced in implementing best-fit slob for the Linux Kernel. These challenges were faced by Group 13 of the Fall 2012 CS411 class with the assistance of Squidly 13. This group consists of Brian Cox, Matthew Okazaki, Trevor Bramwell, and Squidly.

Design

At the beginning of this project we already understood several things about the project and about the kernel:

- The slob was located in the mm directory
- The slab, located in the same directory as the slob, was the current memory management system
- The slob currently implemented first-fit and we wanted best-fit
- We needed a way to show that the best-fit slob used more memory, was less efficient, than first-fit slob.

Using the same general concept that everything is referenced in the kconfig, we used the config menu to disable the slab and enable the slob. this had minor tediousness to it, but we understood what to do. Reading through Tannenbaum we found descriptions of best-fit, worst-fit, last-fit, and first-fit. Comparing first-fit to any of the others showed there was only a minor difference between them, all versions of memory management, except first-fit, must iterate through all the pages. With implementing best-fit, we need to change the slob so it didn't allocate as soon as it found a page it fit in, and we needed to find free space as close to the object's size as possible. Lastly, we had to set up a pair of system calls and a program. The system calls would tell us how much space was being used and how much was free, while the program would take the information from the system calls and find the percentage of free space and used space.

Implementation

We implemented the project by changing the slob_alloc function in the slob.c file. The function would already loop through the link list in order to find the first page that could fit and allocate

the memory. We modified this in order to loop through all the pages in the linked list and find the page that would best fit the unit. We also added system calls within the project in order to retrieve information about the memory allocation. Because of this, we ended up modifying several other files and creating two system call functions in slob.c. The other files that were modified were the unistd.h, the syscall.h, and the syscall_table_32.s. These files were necessary in order to correctly implement the system call.

Testing

Our testing consisted of making sure the kernel ran and using the system calls and program we made to check the memory. Implementing two system calls, sys_get_slob_amt_free and sys_get_slob_amt_claimed, these two system calls would return the amount of free space and the amount of total space. The program would make the call to the two system calls and process the different values to display the percentage free, percentage used, and total volume.

Difficulties

while working on the project we were uncertain if we needed to find the best fitting page or the best fitting block. After a short burst of research we found that only worrying about the page would suffice. Following that, the changes to the kernel would not boot. Thanks to the large number of hours Trevor put into it, he managed to isolate the problem and the kernel was up and running. Lastly the syscalls used for getting the memory information gave extra issues.

Code

```
/*
 * slob_alloc: entry point into the slob allocator.
 */
static void *slob_alloc(size_t size, gfp_t gfp, int align, int node)
{
        struct slob_page *sp = NULL;
        struct slob_page *sp_t;
        struct list_head *slob_list;
        slob_t *b = NULL;
        unsigned long flags;

        if (size < SLOB_BREAK1)
                slob_list = &free_slob_small;
        else if (size < SLOB_BREAK2)
                slob_list = &free_slob_medium;
        else
                slob_list = &free_slob_large;

        spin_lock_irqsave(&slob_lock, flags);
        /* Iterate through each partially free page, try to find room */
        list_for_each_entry(sp_t, slob_list, list) {
```

```
#ifdef CONFIG_NUMA
                /*
                 * If there's a node specification, search for a partial
                 * page with a matching node id in the freelist.
                 */
                if (node != -1 && page_to_nid(&sp_t->page) != node)
                        continue;
#endif
                /* Enough room on this page? */
                if (sp_t->units < SLOB_UNITS(size))
                        /* Not enough room */
                        continue;

                if (sp == NULL)
                        sp = sp_t;

                if (sp_t->units < sp->units)
                        /* Get the smallest slob_page that
                          * is large enough for our needs */
                        sp = sp_t;
        }

        /* Attempt to alloc */
        if(sp != NULL) {
                b = slob_page_alloc(sp, size, align);
        }
        spin_unlock_irqrestore(&slob_lock, flags);

        /* Not enough space: must allocate a new page */
        if (!b) {
                b = slob_new_pages(gfp & ~__GFP_ZERO, 0, node);
                if (!b)
                        return NULL;
                sp = slob_page(b);
                set_slob_page(sp);

                /* We allocatted a new page, increment the count */
                slob_page_count++;


                spin_lock_irqsave(&slob_lock, flags);
                sp->units = SLOB_UNITS(PAGE_SIZE);
                sp->free = b;
                INIT_LIST_HEAD(&sp->list);
                set_slob(b, SLOB_UNITS(PAGE_SIZE), b + SLOB_UNITS(PAGE_SIZE));
                set_slob_page_free(sp, slob_list);
                b = slob_page_alloc(sp, size, align);
                BUG_ON(!b);
                spin_unlock_irqrestore(&slob_lock, flags);
        }
        if (unlikely((gfp & __GFP_ZERO) && b))
```

```
                memset(b, 0, size);
        return b;
}

asmlinkage long sys_slob_claimed(void)
{
        long slob_total = SLOB_UNITS(PAGE_SIZE) * slob_page_count;
        return slob_total;
}

/*
 * Return a count of all free units across used pages */
asmlinkage long sys_slob_free(void)
{
        long free_units = 0;
        struct slob_page *sp;
        struct list_head *slob_list;

        /* Pages with small blocks (page units < 256) */
        slob_list = &free_slob_small;
        list_for_each_entry(sp, slob_list, list) {
                free_units += sp->units;
        }

        /* Pages with medium blocks (page units < 1024) */
        slob_list = &free_slob_medium;
        list_for_each_entry(sp, slob_list, list) {
                free_units += sp->units;
        }

        /* Pages with large blocks (1024 < page units < PAGE_SIZE) */
        slob_list = &free_slob_large;
        list_for_each_entry(sp, slob_list, list) {
                free_units += sp->units;
        }
        return free_units;
}
```