CS411 Operating Systems II Fall 2011

Project #2

Due 23:59:59 2 November 2011
Plan of attack due 23:59:59 26 October 2011
Demo required! Watch for scheduling information.

Introduction:

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

The SLOB (Simple List Of Blocks) allocator, 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.

In this project, you will investigate this issue by modifying the SLOB allocator to use the **best-fit** allocation algorithm, and by writing system calls to provide a measure of the degree of internal fragmentation within the SLOB allocator at a specific point in time.

Objectives:

1. Attempt to improve the fragmentation rate of the SLOB allocator by modifying it to use a different memory allocation algorithm.
2. Understand and modify an existing component of the Linux kernel tree.

Tasks:

1. Read through Chapter 12 of the Love book and read the file header at the top of `mm/slob.c` to familiarize yourself with the memory allocation schemes presently used in the kernel. Browse through the code in `mm/slob.c`. to understand how the SLOB allocator works. Specifically, make sure that you understand that the function `slob_alloc()` implements the **first-fit** allocation algorithm.

2. Write up your *plan of attack* describing the changes you will need to make to `mm/slob.c` so that it runs the **best-fit** allocation algorithm, and submit via TEACH. It will not be evaluated based on the correctness of your proposed plan, but rather on the degree to which it demonstrates that you have examined the existing Linux code and understand roughly how it works.

3. To get the kernel to use the SLOB allocator, copy the config file into your source directory, and use `'make oldconfig'`. You can find it under *General setup* → Choose SLAB allocator →SLOB (Simple Allocator). Once this is done, you should be able to compile a kernel that uses SLOB instead of SLAB.

4. Make a branch in your *git* repository based on the vanilla kernel and named
   `linux-3.0.4-slob_syscalls`.

Within this new branch, devise a method to keep track of all of the memory claimed by the SLOB allocator for small allocations (i.e., every time the `if(!b)` beginning on line 366 of `slob.c` is entered) along with the amount of memory that was not served in an allocation request (i.e. the amount of memory contained in all blocks on the free list). Write separate system calls at the bottom of `mm/slob.c` to return each of these amounts in bytes and as an unsigned int, respectively called

   `sys_get_slob_amt_claimed()`

and one called

   `sys_get_slob_amt_free()`.

Comparing the values returned by each of these functions will give you some idea of the degree of fragmentation suffered by SLOB. Write a simple program that uses these two syscalls to make this comparison.

5. Make another branch in your *git* repository based on the `linux-3.0.4-slob_syscalls` branch and name it

   `linux-3.0.4-best_fit_slob`

Within this new branch, modify `mm/slob.c` so that it uses the **best-fit** algorithm. Begin with the function `slob_alloc()`, and make changes elsewhere as appropriate. The changes necessary to do this should not be too significant, so if you find yourself writing many lines of code you should rethink your implementation.

Note, however, that the SLOB allocator stores free memory in a hierarchical fashion, i.e., in a list of lists of free blocks. Your implementation must account for this by examining each block in each list.

Make sure you add a section to the file header including your team number, the names of your team members, and a description of the changes you made.

6. Use the program you wrote in task **4** to compare the amount of fragmentation suffered by SLOB using the **first-fit** allocation algorithm and the amount suffered by SLOB using the **best-fit** allocation algorithm. Record your observations and conclusions in a README file.

7. Use *git* `diff` to make a patch file representing the changes between your `linux-3.0.4-best_fit_slob` branch and the master branch. Name your patch file

   `linux-3.0.4-best_fit_slob.patch`

Make sure this patch contains only the changes you want to submit, no experimental changes.

Continued on next page ...

Useful resources:

- Chapter 12 in the Love textbok
- Chapter 6 in the Love textbook
- The file header for `mm/slob.c`
- The Linux Cross Referencer: http://lxr.linux.no/#linux+v2.6.34.7
- The course mailing list: cs411-f11@engr.orst.edu

What to turn in:

1. Your team *Plan of Attack*
2. Create a bzipped `tar` archive containing the items below and submit it via teach by the due date.
   - The file `mm/slob.c` from the `linux-3.0.4-best_fit_slob` branch in your *svn* repo.
   - The patch file created in **7**
   - A compiled kernel image named `vmlinuz.proj2.teamXX`, including the changes from your most recent `linux-3.0.4-best_fit_slob` branch.
   - A brief README file describing the results of your comparisons in task **6**
   - The *changelog* from your *git* repo
3. Your group must also submit, by 4:00 pm on Friday, 4 November, a signed hardcopy of your completed *Credit Distribution Agreement*.
4. Each member of your team must individually write and submit a separate *Review* document (see the course wiki for the requirements). This document must be submitted to http://engr.oregonstate.edu/teach before midnight on Sunday, May 1.

Note: The evaluation criteria will be posted 5-6 days before the due date. Be sure to check this posting before making your final submission.

Credits:

Rob Hess:

- Project design and testing