Slab Allocator Project

5th February 2007

1 The Project

In this project you must implement the slab allocator as described in Jeff Bonwick's paper [1]. Your implementation must conform to the interface specification given in Appendix A (which is self explanatory). Your result should be designed as a library that implements the above interface. Your implementation must keep the total internal fragmentation below 12.5%. The differences with respect to Bonwick's paper are:

1. The assignment is to implement an **application level** slab allocator (*not* a kernel memory allocator). This allocator should obtain its storage using the `mmap()` system call, which allocates a specified number of *pages*.
2. Slab deletion is done whenever the `kmem_cache_reap()` call is made, not in response to memory pressure. (use `munmap()` to return the memory to the operating system.
3. You do **not** have to implement the self-scaling hash table alluded to in section 3.2.3 of the paper [1] – this is surprisingly hard to do well. You can use a simple, unbalanced binary tree implementation or borrow an existing AVL tree implementation (note that this data structure *must* be able to handle delete operations).

At `/home/slab` on the machine `cs418.cs.jhu.edu`, you will find the following files:

1. `slab.h` – The interface specification.
2. `slab-tester` – A sample testing program. There are certain compile time switches within the program. You can use them to control the degree of testing while dev elopement.
3. `objects.def` – Object definitions for the test program. Feel free to add other object definitions in the same pattern for extended testing.
4. `Makefile` – the Makefile.
5. `slab.c` – a stub of implementation file.

You must submit implementation files along with the updated `Makefile` as a tarball.

2 Notes

1. You must not make *any* changes to the header file.
2. You can obtain the correct value for PAGE_SIZE by including `/usr/include/sys/user.h`.
3. Your implementation *must not use* `malloc()` *ever*.
4. Your implementation must not rely on the test program for anything.
5. You are strongly encouraged to keep the `-Wall -Werror` options to the C compiler. Most warnings are genuine errors.
6. You are encouraged to study the test program as it serves as extended behavior specification. You are also encouraged to start using it as early as possible.
7. Based on previous years' experience, you are **strongly** encouraged to use a configuration management system (CVS and Subversion are installed on `cs418`).

3 Grading

1. Grading will be based on
   - Whether your allocator works correctly (without faults).
   - Whether particular things that the allocator needs to do work correctly:
     - Object Caching.
     - Slab size selection – handling small/large/huge objects.
     - Slab allocation and deallocation.
     - Allocation and deallocation of backing store.
     - Coloring.
   - Code Quality.
   - Correct implementation of the debugging interface.
2. A testing program is provided for your convenience. We reserve the right to run other test cases.
3. Your homework will be tested on the `cs418` machine, and must therefore compile and run on it.
4. Code that does not compile, `#ifdef`ed /commented out code, *etc.* will receive no credit.
5. We will make a reasonable attempt to grade as much of your submission as possible, but features that cannot be tested will receive no credit. For example: if your allocator segfaults during cache creation, no further testing is possible, and you will receive no credit for all other components as well.

4 Administrivia

1. In this project, you may work in teams of two.
2. The project is due on Tuesady February 27th at 4pm. You are strongly encouraged *not* to delay on starting this project!
3. You may *not* make reference to any existing slab allocator implementation in the course of this assignment.

Appendix A: Slab Allocator Interface

```
#ifndef SLAB_H
#define SLAB_H

/*****************************************************************
    EN 600.318/418 Operating Systems
    Slab Allocator Project.

    This header is a modified version of the interface written by
    Prof. Jonathan S Shapiro.

 *****************************************************************/

/*****************************************************************
   General Notes:

   1) This is a modified version of the Jeff Bonwik's slab allocator.
   2) You must provide an implementation for all of the following
      functions.
   3) All invalid arguments must result in an assert() failure
      (not a segmentation fault).
   4) If any PRECONDITIONS fail, it must result in an assert() failure
      and thus terminate the program.
 *****************************************************************/

#include <unistd.h>

/*****************************************************************
   Main allocator interface
 *****************************************************************/

/* Create a cache that produces objects of the specified size and
   alignment, that will be constructed and destructed using the
   specified constructor and destructor. */
struct kmem_cache *
kmem_cache_create(char *name,
  size_t size,
  int align,
  void (*constructor)(void *, size_t),
  void (*destructor)(void *, size_t));

/* Destroy an entire cache, including all of its slabs.
   PRECONDITION: The cache referred to by ‘cp’ has no slabs containing
                 active objects. */
void kmem_cache_destroy(struct kmem_cache *cp);

/* Allocate an object from a previously allocated cache. Allocates a
   new underlying slab if needed. Note that this does NOT take a flags
   argument, which is different from the paper! */
void *kmem_cache_alloc(struct kmem_cache *cp);

/* Deallocate an object, returning its storage to the cache.
   PRECONDITION: Object ‘buf’ must belong to cache ‘cp’. */
void kmem_cache_free(struct kmem_cache *cp, void *buf);

/* Free all empty slabs in the cache ‘cp’, and return the allocated
   pages to the main backend store. This function returns the number
   of pages freed.

   This is a backend ‘‘interface’’ to trigger garbage collection. */
size_t kmem_cache_reap(struct kmem_cache *cp);


/**********************************************************************
 * Debugging interface
 **********************************************************************/

/* Return the number of slabs in a cache */
size_t
debug_get_nslabs(struct kmem_cache *cp);

/* Return a pointer to the i’th slab associated with the given cache,
   where i=0 gives the first such slab.

   If there is no i’th slab, you _must_ return a NULL pointer
   (not an assert() failure).

   The pointer to the slab_header should remain
   valid until the next call that frees a slab or a cache. */
struct slab_header *
debug_get_slab(struct kmem_cache *cp, size_t i);

struct slab_query
```

```
  /* color: the offset IN BYTES from the start of the slab at which
     the first object appears */
  size_t color;
  size_t size; /* size of objects in this slab */
  size_t align; /* alignment of objects in this slab */
  unsigned nFree; /* number of free objects in this slab*/
  unsigned nAlloc; /* number of allocated objects in this
   slab*/
;

/* Given a slab pointer (previously returned by debug_get_slab()) and
   a pointer to its containing cache, return the information in the
   slab_query structure defined above. */
struct slab_query
debug_get_slab_info(struct kmem_cache *cp, struct slab_header *slab);

/* Return a pointer to the cache header for the cache of cache headers */
struct kmem_cache *debug_get_cacheheader_cache();

/* Return a pointer to the cache header for the bufctl cache, if such
   a cache currently exists, or NULL if it does not: */
struct kmem_cache *debug_get_bufctl_cache();

#endif /* SLAB_H */
```

Bibliography

[1] Jeff Bonwick, *The slab allocator: An object-caching kernel memory allocator.* In USENIX Summer 1996 conference, pages 87–98, 1994.
[2] Jeff Bonwick and Jonathan Adams, *Magazines and vmem: Extending the slab allocator to many cpu's and arbitrary resources.* In Proc. 2001 USENIX Technical Conference, 2001.