Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА №2
З ДИСЦИПЛІНИ “ ОРГАНІЗАЦІЯ ОБЧИСЛЮВАЛЬНИХ ПРОЦЕСІВ”

Виконав:
Студент ІІІ курсу ФІОТ
групи ІО-82
Шендріков Євгеній

Перевірив:
Сімоненко А. В.

м. Київ – 2021 р.

Лістинг програми

slab_internal.c

```
#include <assert.h>
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

#include "slab.h"
#include "hash.h"

/**
 * These are caches we allocate to store 'kmem_bufctl' and 'kmem_slab'.
 * Also, sizeof(kmem_bufctl) and sizeof(kmem_slab) must be smaller
 * than 1/8th the system page size so that caches do not require bufctls
 * and then recurse infinitely.
 */
static struct kmem_cache *my_cache = NULL;        // Cache for kmem_cache
static struct kmem_cache *bufctl_cache = NULL;
static struct kmem_cache *slab_cache = NULL;
static struct kmem_cache *hash_cache = NULL;
static struct kmem_cache *hash_node_cache = NULL;

/* Size of a page on the system */
static size_t system_pagesize = 0;

/**
 * Add a new slab into the slab linkedlist and to the freelist.
 * Since the new slab is complete (refcount == 0), we want to add it to the end of the
list.
 */
static inline void
cache_add_slab(struct kmem_cache *cp, struct kmem_slab *slab)
{
        struct kmem_slab *head;
        struct kmem_slab *tail;

        if (!cp->slabs) {
                // There are no slabs in the cache currently,
                // add the new one at the beginning and update freelist
                DEBUG_PRINT("Adding new (first) slab to top of list\n");
                cp->slabs = slab;
                cp->freelist = slab;

                // Since the list is circular & doubly linked
                slab->next = slab;
                slab->last = slab;
        } else {
                // Add the new slab to the end of the list & pointer shit
                DEBUG_PRINT("Adding new slab \033[1;34m%p\033[0m to tail\n",
(void*)slab);
                head = cp->slabs;
                tail = head->last;
                tail->next = slab;
                slab->last = tail;
                slab->next = head;
                head->last = slab;
        }

        DEBUG_PRINT("Cache %s got new slab \033[1;34m%p\033[0m, next:
\033[1;34m%p\033[0m, last: %p\n", cp->name, (void*)slab, (void*)slab->next,
```

```
(void*)slab->last);

        // Update the freelist pointer
        head = cp->freelist;
        while (head->size == head->refcount) {
                head = head->last;
        }
        if (head != cp->freelist) {
                DEBUG_PRINT("Setting %s freelist to \033[1;34m%p\033[0m\n", cp->name,
(void*)head);
                cp->freelist = head;
        }
        DEBUG_PRINT("Slab freelist is now \033[1;34m%p\033[0m\n", (void*)cp->freelist);
        DEBUG_PRINT("Freelist size %lu, total %lu\n", cp->freelist->refcount, cp-
>freelist->size);

        cp->slab_count++;
        DEBUG_PRINT("Cache %s now has %u slabs\n", cp->name, cp->slab_count);
}

/**
 * Moves a slab to the tail of the list.
 * These slabs should be empty (e.g. refcount 0)
 */
static inline void
cache_empty_slab(struct kmem_cache *cp, struct kmem_slab *slab)
{
        DEBUG_PRINT("Moving slab \033[1;34m%p\033[0m to HEAD of freelist of cache
%s\n", (void*)(slab->last), cp->name);
        if (cp->freelist == slab) {
                // If this slab was first on the freelist...
                // However, don't update the freelist pointer if it would go to a slab
without space
                cp->freelist = slab->next->refcount < slab->next->size ? slab->next :
NULL;
                DEBUG_PRINT("Updating freelist pointer to \033[1;34m%p\033[0m\n",
(void*)cp->freelist);
        }
        if (cp->slabs == slab) {
                // If there's only one slab, it's already at the end
                return;
        }

        slab->last->next = slab->next;
        slab->next->last = slab->last;

        slab->last = cp->slabs->last;
        cp->slabs->last->next = slab;

        slab->next = cp->slabs;
        slab->last = cp->slabs->last;

        cp->slabs = slab;

        DEBUG_PRINT("Slab \033[1;34m%p\033[0m is now the HEAD of cache %s\n",
(void*)(cp->slabs->last), cp->name);
}

/** Remove this slab from the list */
static inline void
cache_remove_slab(struct kmem_cache *cp, struct kmem_slab *slab)
{
        DEBUG_PRINT("Removing slab \033[1;34m%p\033[0m from cache %s freelist\n",
(void*)slab, cp->name);
```

```
        cp->slab_count--;
        if (cp->slabs == slab->next && cp->slabs == slab->last) {
                cp->slabs = NULL;
                cp->freelist = NULL;
        }

        slab->last->next = slab->next;
        slab->next->last = slab->last;

        if (cp->slabs == slab) {
                cp->slabs = slab->next;
        }

        if (cp->freelist == slab) {
                cp->freelist = slab->next->refcount < slab->next->size
                        ? slab->next
                        : NULL;
        }
}

/**
 * Initialize a newly allocated slab.
 * This is used for slabs with object size < 1/8th of a page.
 * In these case, we don't use separate bufctls, but keep the data directly on the page
and put the slab data at the end.
 * Set offset param to skip initializing the first n items in the cache.
 */
static inline struct kmem_slab *
slab_init_small(struct kmem_cache *cp, void *page, size_t offset)
{
        struct kmem_slab *slab;
        size_t available;
        void *i;

        DEBUG_PRINT("Setting up new (small object) slab for cache %s...\n", cp->name);

        // We put the slab at the end of the page
        slab = (struct kmem_slab *)((uintptr_t)page + system_pagesize - sizeof(struct
kmem_slab));

        // Zero out the new slab metadata (this set all things to 0/NULL)
        memset(slab, 0, sizeof(struct kmem_slab));

        available = system_pagesize - sizeof(struct kmem_slab);
        slab->size = (available / cp->object_size) - offset - 1;

        slab->firstbuf.buf = (void**)((uintptr_t)page + (offset * cp->object_size));
        slab->lastbuf.buf = (void**)((uintptr_t)page + ((slab->size) * cp-
>object_size));
        DEBUG_PRINT("One page (%lu bytes) can hold %lu x %lu byte bufs, "
                "plus %lu bytes for slab metadata\n",
                system_pagesize, slab->size, cp->object_size,
                sizeof(struct kmem_slab));

        // Initialize the freelist pointers to be the next
        // To avoid overhead, the first byte of each buf is the pointer
        int count;
        count = 0;
        for (i = slab->firstbuf.buf;
             i <= slab->lastbuf.buf;
             i = (void*)((uintptr_t)i + cp->object_size)) {
                count++;
                *((void**)i) = (void*)((uintptr_t)i + cp->object_size);
        }
```

```
        return slab;
}

static inline struct kmem_slab *
slab_init_large(struct kmem_cache *cp, void *page, int flags)
{
        struct kmem_slab *slab;
        struct kmem_bufctl *bufctl;
        struct kmem_bufctl *last;
        unsigned int i;

        DEBUG_PRINT("Setting up new (large object) slab for cache %s...\n", cp->name);

        // Allocate and zero out a new slab
        slab = kmem_cache_alloc(slab_cache, flags);
        memset(slab, 0, sizeof(struct kmem_slab));

        slab->size = system_pagesize / cp->object_size;
        DEBUG_PRINT("One page (%lu bytes) can hold %lu x %lu byte bufs\n",
               system_pagesize, slab->size, cp->object_size);

        // Allocate bufctls that point to our new data
        // Do the first and last separately to minimize in-loop branching
        slab->firstbuf.bufctl = kmem_cache_alloc(bufctl_cache, flags);
        last = slab->firstbuf.bufctl;
        last->slab = slab;
        last->buf = page;
        last->next = NULL;
        kmem_hash_insert(cp->hash, last->buf, last);
        for (i = 1; i < slab->size-1; i++) {
                bufctl = kmem_cache_alloc(bufctl_cache, flags);
                bufctl->slab = slab;
                bufctl->buf = (void*)((uintptr_t)page + (i * cp->object_size));
                bufctl->next = NULL;
                last->next = bufctl;

                // Insert this bufctl -> buf into the hashtable
                kmem_hash_insert(cp->hash, bufctl->buf, bufctl);
                last = bufctl;
        }
        slab->lastbuf.bufctl = kmem_cache_alloc(bufctl_cache, flags);
        last->next = slab->lastbuf.bufctl;
        last = slab->lastbuf.bufctl;
        last->slab = slab;
        last->buf = (void*)((uintptr_t)page + (i * cp->object_size));
        last->next = NULL;
        kmem_hash_insert(cp->hash, last->buf, last);

        return slab;
}

/**
 * Add a new slab to the given cache.
 * Returns a pointer to the new slab, or 0 on error.
 */
static struct kmem_slab *
cache_grow(struct kmem_cache *cp, int flags)
{
        void *page;
        struct kmem_slab *slab;

        DEBUG_PRINT("Allocating new slab for cache %s...\n", cp->name);
```

```
        // Allocate page-aligned memory
        if (0 != posix_memalign(&page, system_pagesize, system_pagesize))
                return NULL;

        slab = cp->type == SMALL_CACHE
                ? slab_init_small(cp, page, 0 /* No offset */)
                : slab_init_large(cp, page, flags);
        slab->start = page;

        // Add the slab into the cache's freelist
        cache_add_slab(cp, slab);

        return slab;
}

/**
 * Return all bufctls from a slab to their cache
 * ASSUMES: cache type is REGULAR_CACHE
 */
static inline void
slab_reap_large(struct kmem_slab *slab)
{
        struct kmem_bufctl *bufctl;
        unsigned count;

        bufctl = slab->firstbuf.bufctl;
        for (count = 0; bufctl && count < slab->size; count++) {
                kmem_cache_free(bufctl_cache, bufctl);
                bufctl = bufctl->next;
        }
}

/** Reclaims all empty slabs in the cache */
static void
cache_reap(struct kmem_cache *cp, unsigned force)
{
        struct kmem_slab *slab;
        struct kmem_slab *next;
        void *buf;

        if (!cp->slabs) return;
        DEBUG_PRINT("Reaping slabs from cache %s (starts with %u, at
\033[1;34m%p\033[0m)\n", cp->name, cp->slab_count, (void*)cp->slabs);
        slab = cp->slabs;
        while (force || (slab->refcount == 0 && cp->slab_count > 1)) {
                cache_remove_slab(cp, slab);
                buf = (void*)((unsigned long)slab->start>> 12 << 12);
                if (cp->type == REGULAR_CACHE) {
                        slab_reap_large(slab);
                }

                next = slab->next;
                kmem_cache_free(slab_cache, slab);
                DEBUG_PRINT("Freeing \033[1;34m%p\033[0m, from slab\n", buf);
                free(buf);
                if (slab == next) break;
                slab = next;
        }
        DEBUG_PRINT("Cache %s now has %u slabs\n", cp->name, cp->slab_count);
}

/**
 * Called on a newly-full slab.
 * This moves the given slab to the HEAD position in the freelist.
```

```
 */
static inline void
slab_complete(struct kmem_cache *cp, struct kmem_slab *slab)
{
        struct kmem_slab *old_last;
        struct kmem_slab *old_next;

        old_last = slab->last;
        old_next = slab->next;

        slab->last = cp->slabs->last;
        slab->next = cp->slabs->next;
        cp->slabs = slab;

        old_last->next = old_next;
        old_next->last = old_last;

        if (cp->freelist == slab) {
                DEBUG_PRINT("Updating freelist pointer\n");
                cp->freelist = old_next;
        }
}

/**
 * Allocate a buf out of the given slab.
 * ASSUMED: that the cache type == SMALL_CACHE
 * ASSUMED: that the slab has free bufs available
 */
static inline void *
cache_alloc_small(struct kmem_cache *cp, struct kmem_slab *slab)
{
        void **buf;

        buf = slab->firstbuf.buf;
        if (!buf) {
                DEBUG_PRINT("Unable to obtain buf, slab is full...\n");
                DEBUG_PRINT("Slab size %lu, refcount %lu\n", slab->size, slab-
>refcount);
                slab = cache_grow(cp, KM_SLEEP);
                buf = slab->firstbuf.buf;
                if (!buf) return NULL;
        }
        DEBUG_PRINT("Allocating item from small cache at \033[1;34m%p\033[0m\n",
(void*)buf);
        slab->firstbuf.buf = *buf;
        slab->refcount++;

        DEBUG_PRINT("Slab refcount is now %lu\n", slab->refcount);

        return buf;
}

/**
 * Allocate a buf out of the given slab.
 * We need to take a bufctl from the slab's freelist.
 * ASSUMED: that the cache type == REGULAR_CACHE
 * ASSUMED: that the slab has free bufs available
 */
static inline void *
cache_alloc_large(struct kmem_cache *cp, struct kmem_slab *slab)
{
        struct kmem_bufctl *bufctl;

        bufctl = slab->firstbuf.bufctl;
```

```
        if (!bufctl) {
                DEBUG_PRINT("Unable to obtain bufctl, slab is full...\n");
                slab = cache_grow(cp, KM_SLEEP);
                bufctl = slab->firstbuf.bufctl;
                if (!bufctl) return NULL;
        }
        slab->refcount++;
        slab->firstbuf.bufctl = bufctl->next;

        DEBUG_PRINT("Slab refcount is now %lu\n", slab->refcount);

        return bufctl->buf;
}

/**
 * Free an item from the cache
 * ASSUMED: the cache type == SMALL_CACHE
 */
static inline void
cache_free_small(struct kmem_cache *cp, void *buf)
{
        void *page;
        struct kmem_slab *slab;

        // Find the start of the page
        DEBUG_PRINT("Freeing item \033[1;34m%p\033[0m from small cache %s\n", buf, cp-
>name);
        page = (void*)((unsigned long)buf >> 12 << 12);
        DEBUG_PRINT("Found start of page at\033[1;34m %p \033[0m\n", page);


        slab = (struct kmem_slab *)((uintptr_t)page + system_pagesize - sizeof(struct
kmem_slab));

        // Update the last buffer pointer to this one
        // And make this buf point to NULL (end of list)
        *((void**)slab->lastbuf.buf) = buf;

        if((--slab->refcount) == 0 && cp->slab_count > 1) {
                // Don't reap the last slab in the cache
                DEBUG_PRINT("Slab is no longer referenced. Reaping...\n");
                cache_empty_slab(cp, slab);
                cache_reap(cp, 0);
        } else {
                DEBUG_PRINT("Slab refcount is now %lu\n", slab->refcount);
        }
}


/**
 * Free an item from the cache.
 * Here, we need to obtain the bufctl from the cache's hash
 * That buf gets added back to the slab's freelist
 * ASSUMED: the cache type == REGULAR_CACHE
 */
static inline void
cache_free_large(struct kmem_cache *cp, void *buf)
{
        struct kmem_slab *slab;
        struct kmem_bufctl *bufctl;

        DEBUG_PRINT("Freeing item \033[1;34m%p\033[0m from large cache %s\n", buf, cp-
>name);
        bufctl = kmem_hash_get(cp->hash, buf);
```

```
        if (!bufctl) {
                DEBUG_PRINT("Unable to find bufctl for item \033[1;34m%p\033[0m\n",
buf);
                return;
        }
        slab = bufctl->slab;
        assert(slab);

        // Insert this bufctl back into the freelist
        slab->lastbuf.bufctl->next = bufctl;
        slab->lastbuf.bufctl = bufctl;

        if ((--slab->refcount) == 0 && cp->slab_count > 1) {
                // Don't reap the slab slab in the cache
                DEBUG_PRINT("Slab is no longer referenced. Reaping...\n");
                cache_empty_slab(cp, slab);

                // Reclaim the slab
                cache_reap(cp, 0);
        } else {
                DEBUG_PRINT("Slab refcount is now %lu\n", slab->refcount);
        }
}
```

slab.c

```
#include <assert.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>

#include "slab.h"
#include "hash.h"
#include "slab_internal.c"

/**
 * Bootstrapping function, create all the internal caches we'll need,
 * includes hacky variable to tell kmem_cache_create to not init the hash tables at
create time,
 * because that wouldn't create recursion, since those caches won't be initialized
until this function completes.
 */
static uint8_t create_hash_on_create = 1;

static void
init_global_caches()
{
        void *firstpage;

        firstpage = malloc(system_pagesize);
        create_hash_on_create = 0;

        my_cache = firstpage;
        my_cache->name = "my_cache";
        my_cache->object_size = sizeof(struct kmem_cache);
        my_cache->slabs = NULL;
        my_cache->freelist = NULL;
        my_cache->type = SMALL_CACHE;
        my_cache->hash = NULL;

        slab_init_small(my_cache, my_cache, 1);

        hash_node_cache = kmem_cache_create("hash_node_cache",
```

```
                                        sizeof(struct kmem_hash_node),
                                        0 /* No align */);

        hash_cache = kmem_cache_create("hash_cache",
                                       sizeof(struct kmem_hash),
                                       0 /* No align */);

        slab_cache = kmem_cache_create("kmem_slab cache",
                                       sizeof(struct kmem_slab),
                                       0 /* No align */);

        bufctl_cache = kmem_cache_create("kmem_bufctl cache",
                                         sizeof(struct kmem_bufctl),
                                         0 /* No align */);

        create_hash_on_create = 1;

        // Now, init the hash tables for these caches
        my_cache->hash = kmem_hash_init(hash_cache, hash_node_cache);
        hash_node_cache->hash = kmem_hash_init(hash_cache, hash_node_cache);
        hash_cache->hash = kmem_hash_init(hash_cache, hash_node_cache);
        slab_cache->hash = kmem_hash_init(hash_cache, hash_node_cache);
        bufctl_cache->hash =kmem_hash_init(hash_cache, hash_node_cache);
}

/**
 * Create a new cache for objects of a given size.
 * Returns a pointer to an initialized cache, or NULL on error.
 */
struct kmem_cache *
kmem_cache_create(char *name, size_t size, size_t align)
{
        struct kmem_cache *cp;
        size_t align_diff;

        DEBUG_PRINT("Creating new slab: %s. Object size %lu, aligned at %lu\n", name,
size, align);

        // Preconditions: size > 0 and align is 0 or a power of 2
        assert(size > 0);
        assert(align == 0 || !(align & (align - 1)));


        if (!system_pagesize) {
                system_pagesize = sysconf(_SC_PAGESIZE);
                DEBUG_PRINT("System page size is %lu bytes\n", system_pagesize);
        }

        if (!my_cache)
            init_global_caches();

        cp = kmem_cache_alloc(my_cache, KM_SLEEP);
        if (!cp) return NULL;

        // Initialize the new cache
        cp->name = name;
        cp->slabs = NULL;
        cp->freelist = NULL;

        align_diff = align ? size % align : 0;
        cp->object_size = size + align_diff;

        cp->type = cp->object_size < (system_pagesize / 8) ? SMALL_CACHE :
REGULAR_CACHE;
```

```
        DEBUG_PRINT("Cache type is: %d\n", cp->type);

        if (create_hash_on_create) {
                cp->hash = kmem_hash_init(hash_cache, hash_node_cache);
                DEBUG_PRINT("Adding hash %p to cache %s\n", (void*)cp->hash, name);
        }

        // Add the first slab, so we're ready to go at first allocation
        if (!cache_grow(cp, KM_SLEEP)) {
        DEBUG_PRINT("Failed adding initial slab to cache %s\n", name);
        }

        return cp;
}

/**
 * Allocate an item from the given cache flags is one of KM_SLEEP or KM_NOSLEEP,
 * depending if we should block until memory is available to allocate.
 * If unable to allocate (and KM_NOSLEEP),then return NULL.
 */
void *
kmem_cache_alloc(struct kmem_cache *cp, int flags)
{
        struct kmem_slab *slab;
        void *data;

        DEBUG_PRINT("Allocating new item from cache %s\n", cp->name);

        // Get the first slab with free bufs
        // This is the first item in the freelist, except for when that is the HEAD of
the list
        slab = cp->freelist;
        while (!slab || slab->refcount >= slab->size) {
                // No slabs are available, get a new one
                DEBUG_PRINT("Growing the cache...\n");
                slab = cache_grow(cp, flags);
                if (!slab && flags == KM_NOSLEEP) break;
        }

        if (!slab) {
                DEBUG_PRINT("Unable to allocate new slab for cache %s\n", cp->name);
                return NULL;
        }

        data = cp->type == REGULAR_CACHE ? cache_alloc_large(cp, slab) :
cache_alloc_small(cp, slab);

        if (slab->size == slab->refcount) {
                // Slab is full, move it off the cache's freelist
                DEBUG_PRINT("Slab is now complete, moving...\n");
                slab_complete(cp, slab);
        }

        return data;
}

/** Return an element to the cache */
void
kmem_cache_free(struct kmem_cache *cp, void *buf)
{
        if (cp->type == SMALL_CACHE)
            cache_free_small(cp, buf);
        else
```

```
            cache_free_large(cp, buf);
}

/** Destroy the given cache */
void
kmem_cache_destroy(struct kmem_cache *cp)
{
        kmem_hash_free(hash_cache, cp->hash);
        cp->freelist = NULL;
        cache_reap(cp, 1);
}
```

slab.h

```
#ifndef SLAB_H
#define SLAB_H

#include <stddef.h>

#if DEBUG
#define DEBUG_PRINT(...) fprintf(stderr, __VA_ARGS__)
#else
#define DEBUG_PRINT(...) do {} while (0)
#endif

/**
 * The basic idea is to keep caches of pre-initialized objects that the allocator can
quickly give out.
 */
#define KM_SLEEP 0
#define KM_NOSLEEP 1

#define REGULAR_CACHE 0
#define SMALL_CACHE 1

union buf_ish {
        struct kmem_bufctl *bufctl;
        void *buf;
};


struct kmem_slab {
        struct kmem_slab *next;
        struct kmem_slab *last;
        union buf_ish firstbuf;
        union buf_ish lastbuf;  /* For small objects (1/8 pagesize) we don't use
bufctls,
                                 * but keep the bufs directly on the page. In that
case,
                                 * these point directly to the next item in the slab's
freelist.
                                 * Otherwise, they'll be bufctls.
                                 */

        size_t size;            /* Number of bufs total on slab */
        size_t refcount;        /* How many bufs are in use */
        void *start;            /* Address of the allocated memory for this slab */
};


struct kmem_bufctl {
        struct kmem_bufctl *next;  /* Next free buffer in the slab */
        struct kmem_slab *slab;    /* A pointer back to the slab */
```

```
        void *buf;                  /* This is a pointer to the real data */
};

/** Container for an object cache */
struct kmem_cache {
        char *name;                     /* Used for debug purposes */
        unsigned slab_count;            /* Number of slabs in this cache */
        size_t object_size;             /* The size of one object in the cache, including
alignment */
        struct kmem_slab *slabs;        /* Circular, doubly linked list of slabs.
                                         * Sorted as empty (all allocated), then partial
slabs, (some allocated),
                                         * and complete (all free, refcount = 0)
                                         */

        struct kmem_slab *freelist;     /* Pointer to first non-empty slab */
        unsigned char type;             /* Either REGULAR_CACHE or SMALL_CACHE */
        struct kmem_hash *hash;         /* Hash table for mapping buf -> bufctl */
};


/** Create a new cache for objects of a given size */
struct kmem_cache *
kmem_cache_create(
        char *name,
        size_t size,
        size_t align
        //void (*constructor)(void *, size_t),
        //void (*destructor)(void *, size_t)
);

/**
 * Allocate an item from the given cache flags is one of KM_SLEEP or KM_NOSLEEP,
 * depending if we should block until memory is available to allocate
 */
void *
kmem_cache_alloc(
        struct kmem_cache *cp,
        int flags
);

/** Return an element to the cache */
void
kmem_cache_free(
        struct kmem_cache *cp,
        void *buf
);

/** Destroy the given cache */
void
kmem_cache_destroy(
        struct kmem_cache *cp
);

#endif
```

hash.c

```
#include <stdint.h>
#include <stdio.h>

#include "slab.h"
#include "hash.h"
```

```
struct kmem_hash *
kmem_hash_init(struct kmem_cache *hash_cache, struct kmem_cache *node_cache)
{
        DEBUG_PRINT("Allocating hash from cache %s\n", hash_cache->name);
        struct kmem_hash *hash = kmem_cache_alloc(hash_cache, KM_NOSLEEP);
        if (!hash)
            DEBUG_PRINT("Unable to init hash...\n");

        hash->node_cache = node_cache;
        memset(hash->buckets, 0, sizeof(struct kmem_hash_node*) * NUM_BUCKETS);

        return hash;
}

void
kmem_hash_free(struct kmem_cache *hash_cache, struct kmem_hash *hash)
{
        struct kmem_hash_node *node;
        struct kmem_hash_node *temp;
        int i;

        for (i = 0; i < NUM_BUCKETS; i++) {
                node = hash->buckets[i];
                while(node) {
                        temp = node->next;
                        kmem_cache_free(hash->node_cache, node);
                        node = temp;
                }
        }

        kmem_cache_free(hash_cache, hash);
}

void
kmem_hash_insert(struct kmem_hash *hash, void *key, void *data)
{
        unsigned bucket;
        struct kmem_hash_node *node;
        struct kmem_hash_node *old_head;

        bucket = (uintptr_t) key % NUM_BUCKETS;

        node = kmem_cache_alloc(hash->node_cache, KM_SLEEP);
        node->bufaddr = key;
        node->value = data;

        old_head = hash->buckets[bucket];
        hash->buckets[bucket] = node;
        node->next = old_head;
}

void *
kmem_hash_get(struct kmem_hash *hash, void *key)
{
        unsigned bucket;
        struct kmem_hash_node *node;

        bucket = (uintptr_t)key % NUM_BUCKETS;
        node = hash->buckets[bucket];
        while (node) {
                if (node->bufaddr == key) return node->value;
                node = node->next;
        }
```

```
        return NULL;
}

void
kmem_hash_remove(struct kmem_hash *hash, void *bufaddr)
{
        uintptr_t bucket;
        struct kmem_hash_node *last;
        struct kmem_hash_node *node;

        bucket = (uintptr_t) bufaddr % NUM_BUCKETS;
        last = NULL;
        node = hash->buckets[bucket];
        while (node) {
                if (node->bufaddr == bufaddr) {
                        if (last)
                            last->next = node->next;
                        else
                            hash->buckets[bucket] = node->next;
                        kmem_cache_free(hash->node_cache, node);
                        return;
                }
                last = node;
                node = node->next;
        }
}
```

hash.h

```
#ifndef SLAB_HASH_H
#define SLAB_HASH_H

#include <string.h>

#include "slab.h"

/**
 * Basic hash table implementation.
 * It provides the mapping between buf -> bufctl for larger caches.
 */
#define NUM_BUCKETS 32

struct kmem_hash_node {
        void *bufaddr;                /* Address of the membuf */
        void *value;                  /* Address of the bufctl or slab */
        struct kmem_hash_node *next;  /* Next item in the list */
};

struct kmem_hash {
        struct kmem_hash_node *buckets[NUM_BUCKETS];
        struct kmem_cache *node_cache;
};

struct kmem_hash *
kmem_hash_init(struct kmem_cache *hash_cache, struct kmem_cache *node_cache);

void
kmem_hash_free(struct kmem_cache *hash_cache, struct kmem_hash *hash);

/**
 * Insert a bufctl into the hash table
 * ASSUMES it is not already present
```

```
 */
void
kmem_hash_insert(struct kmem_hash *hash, void *bufaddr, void *data);

/**
 * Get a bufctl from a given membuf address
 * Returns NULL if not found
 */
void *
kmem_hash_get(struct kmem_hash *hash, void *bufaddr);

/**
 * Remove the bufctl for the given address from the table
 */
void
kmem_hash_remove(struct kmem_hash *hash, void *bufaddr);

#endif
```

test.c

```
##include <stdio.h>
#include <stdint.h>
#include "slab.h"
#include "hash.h"

struct big_foo {
        int nums[128];
};

struct foo {
        int a;
        int b;
        int c;
};

int
main()
{
        struct foo *datas[340];
        struct big_foo *big_datas[10];
        struct kmem_cache *cache = kmem_cache_create("test_cache", sizeof(struct foo),
0);
        printf("Cache address: %p\n\n", (void*)cache);
        struct foo *test_struct1 = kmem_cache_alloc(cache, KM_SLEEP);
        printf("Allocated item at %p\n\n", (void*)test_struct1);

        struct foo *test_struct2 = kmem_cache_alloc(cache, KM_SLEEP);
        printf("Allocated item at %p\n\n", (void*)test_struct2);

        test_struct1->a = 2;
        test_struct1->b = 4;
        test_struct1->c = 10;

        test_struct2->a = 1;
        test_struct2->b = 5;
        test_struct2->c = 11;

        printf("a + b + c = %d, expected = 16\n", (test_struct1->a + test_struct1->b +
test_struct1->c));
        printf("a + b + c = %d, expected = 17\n", (test_struct2->a + test_struct2->b +
test_struct2->c));
        printf("a + b + c = %d, expected = 16\n\n", (test_struct1->a + test_struct1->b
```

```
+ test_struct1->c));

        kmem_cache_free(cache, test_struct1);
        kmem_cache_free(cache, test_struct2);
        for (int i = 0; i < 340; i++) {
                datas[i] = kmem_cache_alloc(cache, KM_SLEEP);
                datas[i]->a = i;
                datas[i]->b = i*i;
                datas[i]->c = 3*i;
        }
        printf("Lots of small objects: %d, expected 19", (datas[3]->a + datas[4]->b));

        printf("\nfreeing the first slab\n");
        for (int i = 0; i < 338; i++) {
                kmem_cache_free(cache, datas[i]);
        }

        printf("Num slabs: %d\n", cache->slab_count);
        kmem_cache_destroy(cache);

        printf("\n----------\nTesting Hash Table\n----------\n\n");
        int test = 7;
        int test2 = 8;
        kmem_hash_insert(cache->hash, &test, &test2);
        int *res = kmem_hash_get(cache->hash, &test);
        printf("Result: %d", *res);

        printf("\n----------\nTesting Big Cache\n----------\n\n");
        struct kmem_cache *big_cache = kmem_cache_create("test_struct2", sizeof(struct
big_foo), 0);
        for (int i = 0; i < 10; i++) {
                big_datas[i] = kmem_cache_alloc(big_cache, KM_SLEEP);
                big_datas[i]->nums[0] = i;
        }

        printf("Test value %d, expected 9\n", big_datas[2]->nums[0] + big_datas[7]-
>nums[0]);

        for (int i = 0; i < 10; i++) {
                kmem_cache_free(big_cache, big_datas[i]);
        }
        kmem_cache_destroy(big_cache);
}
```

Makefile

```
CFLAGS=-Wall -Wextra -pedantic

all: slab

debug: CFLAGS += -DDEBUG -g
debug: slab

slab:
   gcc $(CFLAGS) -c slab.c hash.c

test: slab
test: CFLAGS += -DDEBUG -g
test:
   gcc $(CFLAGS) test.c -o slab_test slab.o hash.o
   ./slab_test
```

Приклад виконання програми

```
jackshen@DESKTOP-613PBCF:/mnt/c/Users/johnb/Desktop/OOP-Lab2-Shendrikov$ make
gcc -Wall -Wextra -pedantic -c slab.c hash.c
jackshen@DESKTOP-613PBCF:/mnt/c/Users/johnb/Desktop/OOP-Lab2-Shendrikov$ make test
gcc -Wall -Wextra -pedantic -DDEBUG -g -c slab.c hash.c
gcc -Wall -Wextra -pedantic -DDEBUG -g test.c -o slab_test slab.o hash.o
./slab_test
Creating new slab: test_cache. Object size 12, aligned at 0
System page size is 4096 bytes
Setting up new (small object) slab for cache my_cache...
One page (4096 bytes) can hold 70 x 56 byte bufs, plus 56 bytes for slab metadata
Creating new slab: hash_node_cache. Object size 24, aligned at 0
Allocating new item from cache my_cache
Growing the cache...
Allocating new slab for cache my_cache...
Setting up new (small object) slab for cache my_cache...
One page (4096 bytes) can hold 71 x 56 byte bufs, plus 56 bytes for slab metadata
Adding new (first) slab to top of list
Cache my_cache got new slab 0x556bb3fd4fc8, next: 0x556bb3fd4fc8, last: 0x556bb3fd4fc8
Slab freelist is now 0x556bb3fd4fc8
Freelist size 0, total 71
Cache my_cache now has 1 slabs
Allocating item from small cache at 0x556bb3fd4000
Slab refcount is now 1
Cache type is: 1
Allocating new slab for cache hash_node_cache...
Setting up new (small object) slab for cache hash_node_cache...
One page (4096 bytes) can hold 167 x 24 byte bufs, plus 56 bytes for slab metadata
Adding new (first) slab to top of list
Cache hash_node_cache got new slab 0x556bb3fd6fc8, next: 0x556bb3fd6fc8, last: 0x556bb3fd6fc8
Slab freelist is now 0x556bb3fd6fc8
Freelist size 0, total 167
Cache hash_node_cache now has 1 slabs
Creating new slab: hash_cache. Object size 264, aligned at 0
Allocating new item from cache my_cache
Allocating item from small cache at 0x556bb3fd4038
Slab refcount is now 2
Cache type is: 1
Allocating new slab for cache hash_cache...
Setting up new (small object) slab for cache hash_cache...
One page (4096 bytes) can hold 14 x 264 byte bufs, plus 56 bytes for slab metadata
Adding new (first) slab to top of list
Cache hash_cache got new slab 0x556bb3fd8fc8, next: 0x556bb3fd8fc8, last: 0x556bb3fd8fc8
Slab freelist is now 0x556bb3fd8fc8
Freelist size 0, total 14
Cache hash_cache now has 1 slabs
Creating new slab: kmem_slab cache. Object size 56, aligned at 0
Allocating new item from cache my_cache
Allocating item from small cache at 0x556bb3fd4070
Slab refcount is now 3
Cache type is: 1
Allocating new slab for cache kmem_slab cache...
Setting up new (small object) slab for cache kmem_slab cache...
One page (4096 bytes) can hold 71 x 56 byte bufs, plus 56 bytes for slab metadata
Adding new (first) slab to top of list
Cache kmem_slab cache got new slab 0x556bb3fdafc8, next: 0x556bb3fdafc8, last: 0x556bb3fdafc8
Slab freelist is now 0x556bb3fdafc8
Freelist size 0, total 71
Cache kmem_slab cache now has 1 slabs
Creating new slab: kmem_bufctl cache. Object size 24, aligned at 0
Allocating new item from cache my_cache
Allocating item from small cache at 0x556bb3fd40a8
Slab refcount is now 4
Cache type is: 1
Allocating new slab for cache kmem_bufctl cache...
Setting up new (small object) slab for cache kmem_bufctl cache...
One page (4096 bytes) can hold 167 x 24 byte bufs, plus 56 bytes for slab metadata
Adding new (first) slab to top of list
Cache kmem_bufctl cache got new slab 0x556bb3fdcfc8, next: 0x556bb3fdcfc8, last: 0x556bb3fdcfc8
Slab freelist is now 0x556bb3fdcfc8
Freelist size 0, total 167
Cache kmem_bufctl cache now has 1 slabs
Allocating hash from cache hash_cache
Allocating new item from cache hash_cache
Allocating item from small cache at 0x556bb3fd8000
Slab refcount is now 1
Allocating hash from cache hash_cache
Allocating new item from cache hash_cache
Allocating item from small cache at 0x556bb3fd8108
Slab refcount is now 2
Allocating hash from cache hash_cache
```

```
Allocating new item from cache hash_cache
Allocating item from small cache at 0x556bb3fd8210
Slab refcount is now 3
Allocating hash from cache hash_cache
Allocating new item from cache hash_cache
Allocating item from small cache at 0x556bb3fd8318
Slab refcount is now 4
Allocating hash from cache hash_cache
Allocating new item from cache hash_cache
Allocating item from small cache at 0x556bb3fd8420
Slab refcount is now 5
Allocating new item from cache my_cache
Allocating item from small cache at 0x556bb3fd40e0
Slab refcount is now 5
Cache type is: 1
Allocating hash from cache hash_cache
Allocating new item from cache hash_cache
Allocating item from small cache at 0x556bb3fd8528
Slab refcount is now 6
Adding hash 0x556bb3fd8528 to cache test_cache
Allocating new slab for cache test_cache...
Setting up new (small object) slab for cache test_cache...
One page (4096 bytes) can hold 335 x 12 byte bufs, plus 56 bytes for slab metadata
Adding new (first) slab to top of list
Cache test_cache got new slab 0x556bb3fdefc8, next: 0x556bb3fdefc8, last: 0x556bb3fdefc8
Slab freelist is now 0x556bb3fdefc8
Freelist size 0, total 335
Cache test_cache now has 1 slabs
Cache address: 0x556bb3fd40e0

Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde000
Slab refcount is now 1
Allocated item at 0x556bb3fde000

Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde00c
Slab refcount is now 2
Allocated item at 0x556bb3fde00c

a + b + c = 16, expected = 16
a + b + c = 17, expected = 17
a + b + c = 16, expected = 16

Freeing item 0x556bb3fde000 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 1
Freeing item 0x556bb3fde00c from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 0
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde018
Slab refcount is now 1
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde024
Slab refcount is now 2
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde030
Slab refcount is now 3
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde03c
Slab refcount is now 4
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde048
Slab refcount is now 5
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde054
Slab refcount is now 6
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde060
Slab refcount is now 7
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde06c
Slab refcount is now 8
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde078
Slab refcount is now 9
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde084
Slab refcount is now 10
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde090
Slab refcount is now 11
```

```
Slab refcount is now 315
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdeedc
Slab refcount is now 316
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdeee8
Slab refcount is now 317
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdeef4
Slab refcount is now 318
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef00
Slab refcount is now 319
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef0c
Slab refcount is now 320
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef18
Slab refcount is now 321
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef24
Slab refcount is now 322
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef30
Slab refcount is now 323
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef3c
Slab refcount is now 324
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef48
Slab refcount is now 325
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef54
Slab refcount is now 326
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef60
Slab refcount is now 327
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef6c
Slab refcount is now 328
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef78
Slab refcount is now 329
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef84
Slab refcount is now 330
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef90
Slab refcount is now 331
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdef9c
Slab refcount is now 332
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdefa8
Slab refcount is now 333
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fdefb4
Slab refcount is now 334
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fde00c
Slab refcount is now 335
Slab is now complete, moving...
Updating freelist pointer
Allocating new item from cache test_cache
Growing the cache...
Allocating new slab for cache test_cache...
Setting up new (small object) slab for cache test_cache...
One page (4096 bytes) can hold 335 x 12 byte bufs, plus 56 bytes for slab metadata
Adding new slab 0x556bb3fe0fc8 to tail
Cache test_cache got new slab 0x556bb3fe0fc8, next: 0x556bb3fdefc8, last: 0x556bb3fdefc8
Setting test_cache freelist to 0x556bb3fe0fc8
Slab freelist is now 0x556bb3fe0fc8
Freelist size 0, total 335
Cache test_cache now has 2 slabs
Allocating item from small cache at 0x556bb3fe0000
Slab refcount is now 1
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fe000c
Slab refcount is now 2
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fe0018
Slab refcount is now 3
```

```
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fe0024
Slab refcount is now 4
Allocating new item from cache test_cache
Allocating item from small cache at 0x556bb3fe0030
Slab refcount is now 5
Lots of small objects: 19, expected 19
freeing the first slab
Freeing item 0x556bb3fde018 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 334
Freeing item 0x556bb3fde024 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 333
Freeing item 0x556bb3fde030 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 332
Freeing item 0x556bb3fde03c from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 331
Freeing item 0x556bb3fde048 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 330
Freeing item 0x556bb3fde054 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 329
Freeing item 0x556bb3fde060 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 328
Freeing item 0x556bb3fde06c from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 327
Freeing item 0x556bb3fde078 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 326
Freeing item 0x556bb3fde084 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 325
Freeing item 0x556bb3fde090 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 324
Freeing item 0x556bb3fde09c from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 323
Freeing item 0x556bb3fde0a8 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 322
Freeing item 0x556bb3fde0b4 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 321
Freeing item 0x556bb3fde0c0 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 320
Freeing item 0x556bb3fde0cc from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 319
Freeing item 0x556bb3fde0d8 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 318
Freeing item 0x556bb3fde0e4 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 317
Freeing item 0x556bb3fde0f0 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 316
Freeing item 0x556bb3fde0fc from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 315
Freeing item 0x556bb3fde108 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 314
Freeing item 0x556bb3fde114 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 313
Freeing item 0x556bb3fde120 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 312
Freeing item 0x556bb3fde12c from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 311
Freeing item 0x556bb3fde138 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 310
```

```
Slab refcount is now 6
Freeing item 0x556bb3fdef84 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 5
Freeing item 0x556bb3fdef90 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 4
Freeing item 0x556bb3fdef9c from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 3
Freeing item 0x556bb3fdefa8 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 2
Freeing item 0x556bb3fdefb4 from small cache test_cache
Found start of page at 0x556bb3fde000
Slab refcount is now 1
Freeing item 0x556bb3fde00c from small cache test_cache
Found start of page at 0x556bb3fde000
Slab is no longer referenced. Reaping...
Moving slab 0x556bb3fe0fc8 to HEAD of freelist of cache test_cache
Reaping slabs from cache test_cache (starts with 2, at 0x556bb3fdefc8)
Removing slab 0x556bb3fdefc8 from cache test_cache freelist
Freeing item 0x556bb3fdefc8 from small cache kmem_slab cache
Found start of page at 0x556bb3fde000
Slab refcount is now 18446744073709551615
Freeing 0x556bb3fde000, from slab
Cache test_cache now has 1 slabs
Freeing item 0x556bb3fe0000 from small cache test_cache
Found start of page at 0x556bb3fe0000
Slab refcount is now 4
Freeing item 0x556bb3fe000c from small cache test_cache
Found start of page at 0x556bb3fe0000
Slab refcount is now 3
Freeing item 0x556bb3fe0018 from small cache test_cache
Found start of page at 0x556bb3fe0000
Slab refcount is now 2
Num slabs: 1
Freeing item 0x556bb3fd8528 from small cache hash_cache
Found start of page at 0x556bb3fd8000
Slab refcount is now 5
Reaping slabs from cache test_cache (starts with 1, at 0x556bb3fe0fc8)
Removing slab 0x556bb3fe0fc8 from cache test_cache freelist
Freeing item 0x556bb3fe0fc8 from small cache kmem_slab cache
Found start of page at 0x556bb3fe0000
Slab refcount is now 1
Freeing 0x556bb3fe0000, from slab
Cache test_cache now has 0 slabs

----------
Testing Hash Table
----------

Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd6000
Slab refcount is now 1
Result: 8
----------
Testing Big Cache
----------

Creating new slab: test_struct2. Object size 512, aligned at 0
Allocating new item from cache my_cache
Allocating item from small cache at 0x556bb3fd4118
Slab refcount is now 6
Cache type is: 0
Allocating hash from cache hash_cache
Allocating new item from cache hash_cache
Allocating item from small cache at 0x556bb3fd8630
Slab refcount is now 6
Adding hash 0x556bb3fd8630 to cache test_struct2
Allocating new slab for cache test_struct2...
Setting up new (large object) slab for cache test_struct2...
Allocating new item from cache kmem_slab cache
Allocating item from small cache at 0x556bb3fda000
Slab refcount is now 1
One page (4096 bytes) can hold 8 x 512 byte bufs
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc000
Slab refcount is now 1
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd6018
Slab refcount is now 2
Allocating new item from cache kmem_bufctl cache
```

```
Allocating item from small cache at 0x556bb3fdc018
Slab refcount is now 2
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd6030
Slab refcount is now 3
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc030
Slab refcount is now 3
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd6048
Slab refcount is now 4
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc048
Slab refcount is now 4
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd6060
Slab refcount is now 5
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc060
Slab refcount is now 5
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd6078
Slab refcount is now 6
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc078
Slab refcount is now 6
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd6090
Slab refcount is now 7
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc090
Slab refcount is now 7
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd60a8
Slab refcount is now 8
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc0a8
Slab refcount is now 8
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd60c0
Slab refcount is now 9
Adding new (first) slab to top of list
Cache test_struct2 got new slab 0x556bb3fda000, next: 0x556bb3fda000, last: 0x556bb3fda000
Slab freelist is now 0x556bb3fda000
Freelist size 0, total 8
Cache test_struct2 now has 1 slabs
Allocating new item from cache test_struct2
Slab refcount is now 1
Allocating new item from cache test_struct2
Slab refcount is now 2
Allocating new item from cache test_struct2
Slab refcount is now 3
Allocating new item from cache test_struct2
Slab refcount is now 4
Allocating new item from cache test_struct2
Slab refcount is now 5
Allocating new item from cache test_struct2
Slab refcount is now 6
Allocating new item from cache test_struct2
Slab refcount is now 7
Allocating new item from cache test_struct2
Slab refcount is now 8
Slab is now complete, moving...
Updating freelist pointer
Allocating new item from cache test_struct2
Growing the cache...
Allocating new slab for cache test_struct2...
Setting up new (large object) slab for cache test_struct2...
Allocating new item from cache kmem_slab cache
Allocating item from small cache at 0x556bb3fda038
Slab refcount is now 2
One page (4096 bytes) can hold 8 x 512 byte bufs
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc0c0
Slab refcount is now 9
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd60d8
Slab refcount is now 10
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc0d8
Slab refcount is now 10
Allocating new item from cache hash_node_cache
```

```
Allocating item from small cache at 0x556bb3fd60f0
Slab refcount is now 11
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc0f0
Slab refcount is now 11
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd6108
Slab refcount is now 12
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc108
Slab refcount is now 12
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd6120
Slab refcount is now 13
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc120
Slab refcount is now 13
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd6138
Slab refcount is now 14
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc138
Slab refcount is now 14
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd6150
Slab refcount is now 15
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc150
Slab refcount is now 15
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd6168
Slab refcount is now 16
Allocating new item from cache kmem_bufctl cache
Allocating item from small cache at 0x556bb3fdc168
Slab refcount is now 16
Allocating new item from cache hash_node_cache
Allocating item from small cache at 0x556bb3fd6180
Slab refcount is now 17
Adding new slab 0x556bb3fda038 to tail
Cache test_struct2 got new slab 0x556bb3fda038, next: 0x556bb3fda000, last: 0x556bb3fda000
Setting test_struct2 freelist to 0x556bb3fda038
Slab freelist is now 0x556bb3fda038
Freelist size 0, total 8
Cache test_struct2 now has 2 slabs
Slab refcount is now 1
Allocating new item from cache test_struct2
Slab refcount is now 2
Test value 9, expected 9
Freeing item 0x556bb3fde000 from large cache test_struct2
Slab refcount is now 7
Freeing item 0x556bb3fde200 from large cache test_struct2
Slab refcount is now 6
Freeing item 0x556bb3fde400 from large cache test_struct2
Slab refcount is now 5
Freeing item 0x556bb3fde600 from large cache test_struct2
Slab refcount is now 4
Freeing item 0x556bb3fde800 from large cache test_struct2
Slab refcount is now 3
Freeing item 0x556bb3fdea00 from large cache test_struct2
Slab refcount is now 2
Freeing item 0x556bb3fdec00 from large cache test_struct2
Slab refcount is now 1
Freeing item 0x556bb3fdee00 from large cache test_struct2
Slab is no longer referenced. Reaping...
Moving slab 0x556bb3fda038 to HEAD of freelist of cache test_struct2
Reaping slabs from cache test_struct2 (starts with 2, at 0x556bb3fda000)
Removing slab 0x556bb3fda000 from cache test_struct2 freelist
Freeing item 0x556bb3fda000 from small cache kmem_slab cache
Found start of page at 0x556bb3fda000
Slab refcount is now 1
Freeing 0x556bb3fde000, from slab
Cache test_struct2 now has 1 slabs
Freeing item 0x556bb3fe2000 from large cache test_struct2
Slab refcount is now 1
Freeing item 0x556bb3fe2200 from large cache test_struct2
Slab refcount is now 0
Freeing item 0x556bb3fd6180 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 16
Freeing item 0x556bb3fd6168 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 15
Freeing item 0x556bb3fd6150 from small cache hash_node_cache
```

```
Found start of page at 0x556bb3fd6000
Slab refcount is now 14
Freeing item 0x556bb3fd6138 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 13
Freeing item 0x556bb3fd6120 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 12
Freeing item 0x556bb3fd6108 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 11
Freeing item 0x556bb3fd60f0 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 10
Freeing item 0x556bb3fd60d8 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 9
Freeing item 0x556bb3fd60c0 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 8
Freeing item 0x556bb3fd60a8 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 7
Freeing item 0x556bb3fd6090 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 6
Freeing item 0x556bb3fd6078 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 5
Freeing item 0x556bb3fd6060 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 4
Freeing item 0x556bb3fd6048 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 3
Freeing item 0x556bb3fd6030 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 2
Freeing item 0x556bb3fd6018 from small cache hash_node_cache
Found start of page at 0x556bb3fd6000
Slab refcount is now 1
Freeing item 0x556bb3fd8630 from small cache hash_cache
Found start of page at 0x556bb3fd8000
Slab refcount is now 5
Reaping slabs from cache test_struct2 (starts with 1, at 0x556bb3fda038)
Removing slab 0x556bb3fda038 from cache test_struct2 freelist
Freeing item 0x556bb3fdc0f0 from small cache kmem_bufctl cache
Found start of page at 0x556bb3fdc000
Slab refcount is now 15
Freeing item 0x556bb3fdc108 from small cache kmem_bufctl cache
Found start of page at 0x556bb3fdc000
Slab refcount is now 14
Freeing item 0x556bb3fdc120 from small cache kmem_bufctl cache
Found start of page at 0x556bb3fdc000
Slab refcount is now 13
Freeing item 0x556bb3fdc138 from small cache kmem_bufctl cache
Found start of page at 0x556bb3fdc000
Slab refcount is now 12
Freeing item 0x556bb3fdc150 from small cache kmem_bufctl cache
Found start of page at 0x556bb3fdc000
Slab refcount is now 11
Freeing item 0x556bb3fdc168 from small cache kmem_bufctl cache
Found start of page at 0x556bb3fdc000
Slab refcount is now 10
Freeing item 0x556bb3fdc0c0 from small cache kmem_bufctl cache
Found start of page at 0x556bb3fdc000
Slab refcount is now 9
Freeing item 0x556bb3fdc0d8 from small cache kmem_bufctl cache
Found start of page at 0x556bb3fdc000
Slab refcount is now 8
Freeing item 0x556bb3fda038 from small cache kmem_slab cache
Found start of page at 0x556bb3fda000
Slab refcount is now 0
Freeing 0x556bb3fe2000, from slab
Cache test_struct2 now has 0 slabs
jackshen@DESKTOP-613PBCF:/mnt/c/Users/johnb/Desktop/OOP-Lab2-Shendrikov$
```