COM 341, 1418, Operating Systems: Task #4 Kernel Memory Management

Toksaitov Dmitrii Alexandrovich
`toksaitov.d@gmail.com`

May 3, 2011

1 Introduction

This time you need to implement a kernel heap manager based on the Doug Lea's Malloc, or dlmalloc allocator.

Students have two weeks to finish this task. The kernel source tree should be packed and sent to `toksaitov.d@gmail.com` before the deadline.

As usual you need to comply with additional requirements:

Letters

- Compose you letter properly in a formal way.
- Do not send blank letters. Briefly describe you source tree structure and the approach used for the solution in this task.
- Do not use *translit* in your letters. Write in English instead.

Archives

Pack all kernel source files into a *tar* archive (*tarball*) and apply compression with ether *gzip* or *bunzip2* (man tar) utilities.

Makefiles

Provide a proper *Makefile* to compile you kernel binary image. Changes in certain source files should signal the *make* utility to recompile this and only this files and relink all object files (old and new) into a proper kernel executable.

Students' work

- Team work is *NOT* encouraged in this task.
- Equal blocks of code or similar structural pieces in separate works will be considered as academic dishonesty and all "authors" will get zero points for this task.

It is recommended to use a Version Control System (*VCS*) such as *Git* or *Mercurial* to record you changes and to be able to snap back to a certain source tree state in case of problems. You will get 0.2 extra points if you will provide your kernel source tree via the local private repository or via a public web-based one (*GitHub*, *Bitbucket*, etc.).

2 Objectives

In order to implement the heap manager in your kernel, you need a working paging and page mapping system.

3 Functions

Here is the list of functions that to implement. Note that different function names can be used.

k_malloc

Returns a pointer to a newly allocated memory block of *size* bytes from the *heap* or a *NULL* pointer if no space is available. The memory block can be forced to start at a page boundary if the *aligned* flag was set.

```
void* k_malloc(heap_t *heap, size_t size, bool aligned);
...
```

k_free

Releases the block of memory pointed to by *block* from the *heap*, that had been previously allocated using *k_malloc*.

```
void* k_free(heap_t *heap, void *block);
...
```

4 Links

4.1 Version Control Systems

Git — The Official Website

`http://git-scm.com`
`http://git-scm.com/documentation`

Pro Git — The Creative Commons Licensed Book on Git

`http://progit.org/book`

Mercurial — The Official Website

`http://mercurial.selenic.com`
`http://mercurial.selenic.com/guide`

Hg Init: a Mercurial Tutorial Written by Joel Spolsky

`http://hginit.com`

4.2 Web-based Source Code Repositories

GitHub

`http://github.com`

Bitbucket

`http://bitbucket.org`

4.3 Doug Lea's Malloc

Description

`http://g.oswego.edu/dl/html/malloc.html`

Implementation

`ftp://g.oswego.edu/pub/misc/malloc.c`

4.4 Intel® Developer's Manuals

Volume 1: Basic Architecture

`http://www.intel.com/Assets/PDF/manual/253665.pdf`

Volume 3A: System Programming Guide, Part 1

`http://www.intel.com/Assets/PDF/manual/253668.pdf`

4.5 The GNU Compiler Collection

GCC Online Documentation
`http://gcc.gnu.org/onlinedocs`

4.6 The Netwide Assembler

Online Documentation for NASM 2.09.06
`http://www.nasm.us/xdoc/2.09.06/html/nasmdoc0.html`