University of Hamburg
Department of Informatics
Group of Scientific Computing
K. Chasapis, Dr. J. Kunkel, Dr. M. Dolz, M. Wiedemann

Praktikum „Kernel Programming“
in WiSe 2014/2015
Assignment 0
Deadline: 29.11.2014

---

1 Exercise: Link lists

Link lists is one of the fundamental data structures in computer science. In this exercise we require you to implement a generic link list of your preference (singly, doubly, circular). Your implementation must include at least the following functions:

create(), destroy(), insert(), delete(), next(), previous().

The functionality offered by the link list should be exported as an abstract data type. First, you should design a clean interface that will define the methods that you will use to manipulate the link list. Formulate this interface in a C header file. You should not export the definition of the link list structure at the header file. For this reason you can use a *type definition* as the one below:

typedef struct link_list link_list_t;

Function prototype example:

int insert(link_list_t head, **void** *value);

Implement the interface in a C file (.c). In a separate C file implement the **main()** function that you will use to test the link list. Create a Makefile that you will use to compile your code. For more information on link lists you can visit:
http://en.wikipedia.org/wiki/Linked_list.

2 Exercise: Free lists

In this exercise you will implement your own memory allocator. Allocators uses *free lists* to keep track of the available space. Your memory allocation should implement the following functions:

init(), destroy(), allocate(), free(), print().

Whereas for the first four functions the semantics are obvious for the print function we require to print the free list of the memory allocator.
There are several memory allocation strategies. In this exercise you are required to implement the first fit and the best fit. In the first strategy the allocator returns the first free block that will meet the size requirements. In the best fit strategy the allocator searches in the available blocks and returns the smaller block which fits the requested size.
Allocation function prototypes examples:

void * allocate_first_fit(**size_t** size);

void * allocate_best_fit(**size_t** size);

As in the first exercise define the allocator's interface in a header file and implement it in a C file. You should also deliver the Makefile and a **main()** function to test the allocator.
More information on the allocation strategies visit:
http://courses.cs.vt.edu/ csonline/OS/Lessons/MemoryAllocation/index.html.

Submission

You should submit the solution of the two exercises in a single tar file named after your last name and the assignment number (e.g chasapis_0.tar). You should include main functions and also test programs.
Submit the solution of your exercise by email to your corresponding supervisor.
More details will be given in the lecture time.