**COMPSCI 453/552: Operating Systems**

**Programming Assignment 6**

# Shell Project, part 3 (with Buddy System Memory Manager) (110–140 points)

Due Date -On Class Home Page–

## 1 Introduction

In the last release of the mini-shell we will work on providing our own memory management.

We will replace `malloc` with your own memory management scheme based on the Buddy system discussed in class.

## 2 Requirements

### 2.1 Memory Management (100 points)

Implement your own memory manager using the Buddy Algorithm. You should use the `sbrk()` to initially allocate a large block of memory. A good amount may be 64MB. See the example `ch11/sbrk-test.c` for the usage of the `sbrk` function. From there on manage the chunk of memory returned by `sbrk` using your own memory management functions.

Note that you will have to store the data structures used to represent the buddy system some where in memory. There are two parts to this issue. First, you need to store the array of pointers to the blocks somewhere. You may reserve some amount of memory in the chunk you obtained from `sbrk` for this purpose. Or you may statically declare these pointers in you code. The second option is simpler. For this purpose you may assume that the maximum amount of memory that you will ever manage is limited to 4GB. Secondly, the pointers associated with each free block in the buddy system should be stored in the block as explained in the Buddy system algorithm. Note that `readline` will internally continue to use `malloc` and we will let that be (until we use interposing as explained later).

Have all the initialization be in a separate function. If the user doesn't call this function, then it is transparently called whenever the user calls buddy malloc/calloc for the first time. You can test for that using a static variable.

You should use the following prototypes for your buddy functions:

```
void  buddy_init(size_t);
void *buddy_calloc(size_t, size_t);
void *buddy_malloc(size_t);
void *buddy_realloc(void *ptr, size_t size);
void  buddy_free(void *);
```

If the memory cannot be allocated, then your `buddy_calloc` or `buddy_malloc` functions should set the errno to `ENOMEM` (which is defined in `errno.h` header file).

Build a test suite for your buddy system. Sample test code that you can use as a starting point is at `lab/buddy-system/` folder in the lab examples for the class. It contains two performance test files: `buddy-test.c` and `malloc-test.c` that run identical tests using the two different allocators. **Please do not modify the files**. In addition, there is a unit test file `buddy-unit-test.c`, to which you can add more tests.

*References.*

- Read Section 5.4 (Address Arithmetic) and Section 8.7 (Example–A Storage Allocator) from the C book by Kernighan and Ritchie to help you get started.
- Donald Knuth. *Fundamental Algorithms. The Art of Computer Programming 1* (Second ed.) pp. 435-455. Addison-Wesley.
- Look up on Wikipedia.

### 2.2 Interposing Malloc

Look at the example in `ch11/library-interposer` to see how you can interpose all your malloc/free/calloc calls. This allows us to even get readline to use our malloc implementation. To use interposing, your buddy system allocator will implement the same interface as malloc (and with the same function names).

You would need to make your buddy system into another shared library. Name this library to be `libbuddy.so`. Use the following command to interpose for malloc using `libbuddy.so`:

```
LD_LIBRARY_PATH=. LD_PRELOAD=libbuddy.so ./mydash
```

### 2.3 Integrating with your shell

Now use the interposing version of your buddy system to integrate with the shell. Run all the base tests on your dash to check that it works well with the buddy new memory allocator.

### 2.4 Buddy System Performance

**Required for graduate students. Extra credit for undergraduates (10 points)**

Test your buddy system implementation against malloc and make sure that it out-performs it. For the purposes of measuring performance, use the `buddy-test.c` and `malloc-test.c` code provided in the sample code for this project.

Here is performance comparison for my solution (tested on onyx, compiled with -O3 optimizer flag)

```
[amit@onyx buddy-system]$ time buddy-test 20000000 1234 s
```

```
real    0m1.817s
user    0m1.812s
sys     0m0.001s
[amit@onyx buddy-system]$ time malloc-test 20000000 1234 s

real    0m2.838s
user    0m2.830s
sys     0m0.001s
[amit@onyx buddy-system]$
```

## 3 Extra Credit (10 points)

Modify your buddy system implementation such that a program can allocate multiple co-existing buddy systems! Note that this may require you to modify the buddy system interface.

## 4 Extra Credit (10 points)

Make your buddy system library be thread-safe. Compare its runtime performance against malloc. Is it still faster?

## 5 Hints on Testing and Development.

- Implement the Buddy system as a separate class and test it using small programs. A good simple test would be to use your linked lists and the TestList program with your memory manager. Once you are confident in your Buddy memory manager, then integrate it into your shell and then test the shell again to make sure that it still works as well as before. Your test cases from the previous project will come in handy here.

## 6 Source Code Control

Include a section in your README file with your observation on your usage of subversion. Also include the log file for your main source file and name this file to `svn.log` and leave it at the top level of your submission.

## 7 Documentation (10 points)

Use your `README` file to communicate the structure of the files, the significant highlights of your project, test cases, the limitations and features, things that can still be improved etc. Continue to use `doxygen` tool from the previous project.

## 8 Submitting the Assignment

- **Required submission directory layout!** The top level of your submission directory must have a Makefile that builds and leaves the dash executable in the top level along with both libraries (that are also built by the Makefile): `libmylib.so` and `libbuddy.so`. The buddy system library should be suitable for pre-loading (which requires that it has the same interface as malloc does).
- The name of the executable must be `mydash`. However, you are free to leave the dash source in a separate sub-folder (in fact, that is preferable). Failure to follow this instruction will cost you at least 10% of the grade.
- Also make sure that there is a `README` file on the top-level that contains your name, date, assignment number, design and observations about the assignment, test cases. It should clearly document what parts of this assignment you have attempted.
- Make sure to include the unmodified `buddy-test.c` as well as `malloc-test.c` along with your buddy system code so I can run those tests independently of dash.
- Prepare your directory for submission by removing all executables and object files and other clutter. You should only have the source code, README files, Makefiles, test scripts and test files before submitting.

Change to the top level of your project directory and type the following command (on `onyx`) to submit the assignment.

```
submit amit cs453 p6
```

or

```
submit amit cs552 p6
```