CS342 Fall 2015 – Project 4
Memory Management: Buddy Algorithm

Assigned: November 27, 2015, Friday
Due date: December 12, 2015, Saturday, 23:55

In this project you will implement a simple memory management library that uses Buddy allocator as the allocation algorithm. Your library will be used by applications that would like to allocate memory for its objects. You will develop your project in Linux/C. The project can be done in groups of 2. This time you will implement a static library (ending with .a).

Develop a memory management library (libbuddy.a) that will manage a given chunk of free memory. That means the library will allow an application to allocate and free memory from the given chunk. The chunk of free memory (a contiguous chunk of free space) will be created by the application using the library and a pointer to the chunk and the size of the chunk will be passed to the library (see sample code in github). The library will manage the chunk space. Then the application will make requests to the library to allocate and deallocate memory and the library will do the allocations from the chunk.

Your library will implement the following functions (you can not change this interface):

- **int binit (void *chunkptr, int size).** This initializes the library to manage the provided chunk space. The chunkptr points to the chunk. The chunk is created outside of the library. The size parameter is the size of the chunk to manage (to allocate memory from). Returns -1 if error, 0 if success.
- **void *balloc (int size).** Allocate memory for an object (i.e., for a request). The memory should be allocated from the chunk. The size of allocation is given as a parameter (request size). The allocation should be at least that big (in fact the allocated amount will be a power of 2, since we use are using buddy algorithm to manage the chunk). Returns a pointer to the beginning of the allocation so that the application can use the pointer to put something (an object/structure/value) into the allocated space.
- **void bfree (void ptr).** Free the allocated space. The space to be freed is pointed by ptr.
- **void bprint ()**. Print current state information about the chunk: which portions are allocated, which portions are free (in sorted order with respect to address). The format for printing out is up to you.

An application app.c will be compiled and linked with your library as follow:

```
gcc –Wall –o app  app.c -L.  -lbuddy
```

We can run the application as follows, for example:

```
./app 1024
```

The chunk size will be 1024 KB (kilobytes) in this example.

We will develop test applications to test and stress your library implementation. You should also develop test applications to test your library.

We provide you a sample code to start with at github.

- https://github.com/korpeoglu/p4
- Note that the code there can be updated at any time without any notice. It is just a sample code to guide you and to get you started.

Report. Do some timing experiments and write a report about the results. Put tables and/or figures into your report showing the results. Try to interpret the results.

Clarifications:

- Your library will be a static library (.a). See Makefile provided about how a static library can be created.
- The library will be used by single threaded programs (thread-safety is not a concern).
- Do not use malloc or a similar existing allocation routines in the implementation of your library. You will manage the chunk space with your own functions and implementation.
- You will keep the management structures also in the chunk space itself (i.e., a tree-like structure will be on the chunk itself). Do not use a separate structure sitting outside of the chunk space to keep track of the allocated and free portions of the chunk. That booking information should be part of the chunk itself and you will not use malloc to create any dynamic structure. You will use just pointer operations in your implementation.
- You will use Makefile to compile. You will submit Makefile as well. We will just type "make" and compile your programs.
- The minimum chunk size is 32 KB and maximum chunk size is 32 MB.
- The minimum request size is 256 bytes and maximum request size is 64 KB.
- Develop you program in a 64-bit machine (most machines are 64-bit) and 64-bit OS. Addresses (pointers) are 64 bits (8 bytes long). Hence we use *long int* to refer to a memory address an integer.
- More clarifications may be posted later.