# CS3113fa17 Project 2 Memory Allocation (75 points)

# Due 12/7/17 at 11:45 PM



In this project, you will simulate four memory allocation algorithms. You have flexibility in how you implement the actual memory allocation. You must use C. Before starting, design your solution into small testable parts. It is not necessary to perform any actual memory allocation but your code should keep a record of the free and allocated spaces in a memory structure.

## Algorithms

You should implement the four memory allocation algorithms algorithms from Chapter 7 of the Stallings Book: (1) **First Fit**; (2) **Best Fit**; (3) **Next Fit**; and (4) **Buddy System**. Each of these algorithms were discussed at length in class. Use Chapter 7, pages 319 - 323, of Stalling's Operations Systems book (8th edition) and lecture slides as a reference.

To implement memory allocation, you have the freedom to choose a method you believe is best. The naive method of implementing methods (1) - (3) is to actually allocate large arrays for the code and label the contents of each array. For the Buddy System, you may choose to create binary trees that represent "buddies", etc. In any case, the description of your implementations should be detailed in your `readme` document.

Each algorithm will be given a starting memory allocation size. The allocation size will be $2^k$ bytes, where $k$ is an integer between 4 and 20.

Your code should be implemented in C. If you decide to use any external packages, you must clear it with the professor. You should also supply a readme file that will serve as your manual. The command `make all` should create an executable called `project2`. The first parameter is a string of the algorithm that should be used. The options are BESTFIT, FIRSTFIT, NEXTFIT, and BUDDY. The second parameter is the total memory allocation `N`. All algorithms have a script file name as the last parameter.

Below are four example invocations of the project 2 executable where N is the total memory allocation and the strings "ex75.txt" and "ex76.txt" are script files derived from examples in the book. Note, only one allocation and algorithm can be chosen for each execution of code. Be sure to free all memory prior to exiting the program.

```
./project2 BESTFIT N ex75.txt
./project2 FIRSTFIT N ex75.txt
./project2 NEXTFIT N ex75.txt
./project2 BUDDY N ex76.txt
```

# Commands

| Command | Description |
| --- | --- |
| `REQUEST A n` | Allocates `n` bytes for a process named `A`. On successful allocation, print `ALLOCATED A x`, where `A` is the name of the allocated process and `x` is the relative location of the allocated memory. On error, print `FAIL REQUEST A n`. |
| `RELEASE A` | Releases the memory held by a process `A`. On success, print `FREE A n x` where `n` is the amount of reclaimed memory and `x` is the start relative address of the released memory. On error, print `FAIL RELEASE A`. |
| `LIST AVAILABLE` | Prints location information for all available memory. Prints a list of pairs of "$(n_1, x_1)$ $(n_2, x_2)$ . . .", where $n_i$ is the amount available and $x_i$ is the starting location. If there are no available blocks you should return `FULL`. |
| `LIST ASSIGNED` | Returns a list of space separates=d triples including of the form "$(A_1, n_1, x_1)$ $(A_2, n_2, x_2)$ . . .", where $A_i$ represents process labels, $n_i$ represents the number of allocated bytes for that process, and $x_i$ is the relative starting address of the process. If there are no allocated blocks, this should return `NONE`. |
| `FIND A` | Locates the starting address and size allocated by the process labeled `A`. If successful, this command returns an tuple $(A, n, x)$, where $n$ is the amount allocated by `A` and `x` is the relative starting address of the process labeled `A`. |

Process labels may be strings with a max length of 16 characters, containing only upper and/or lowercase characters and no spaces. All memory locations should be considered relative to the start of the free memory block. The addresses and memory sizes should be represented as long, unsigned integers. And memory sizes should always be represented as bytes. All print commands should go to stdout.

# Script File

The script file is a text file with a series of commands as defined above. Your code should read each line of the text file and execute the command. Lines prefixed with `#` represent comments. Below is an example script file. The output of the script file will change depending on the algorithm and allocation used. Use the examples in the book to help you create scripts files that will work. The example below is an script file from Example 7.6 with extra commands added in between to view the progress of the script.

```
# Start from nothing
LIST AVAILABLE
REQUEST A 131072
# Check to see if A was added
LIST ASSIGNED
REQUEST B 240000
REQUEST C 65536
REQUEST D 262144
LIST AVAILABLE
RELEASE B
RELEASE A
LIST AVAILABLE
REQUEST E 75000
FIND E
RELEASE C
RELEASE E
RELEASE D
# Everything should be gone
LIST ASSIGNED
```

# Grading

Submit all your source files, your makefile, and your readme file. You do not need to develop your code in your Ubuntu Google cloud instance but you should ensure that your code runs in this environment. Be sure to develop develop your own code. If you use any external code, you should state this in your readme file and make it available to the TA. To evaluate your code, we generate random test scripts that produce known results. We will verify that the results from your code match expected results.

### Class-wide testing

To ensure everyone is getting consistent results, we will create a message board for students to post script files and the resulting outputs for comparison. **All students who submit new, example script/outfile comparisons before *Tuesday 12/5/2017* will receive 10% bonus points.** This will allow other students to test their algorithms before the due date.

| Task | Percent |
| --- | --- |
| Each command works correctly | 45% |
| Each algorithm produces the correct results | 40% |
| `readme` file is through and descriptive | 15% |
| Submitted script/outfile pairs | +10% |
| **Max Total** | **110%** |