CS 421 – Fall 2017
Project #4

100 Points **Due: Friday, 12/15/17 at 11:59PM in D2L**

In this project, you will be implementing a memory allocator for the heap of a user-level process, with functions similar to those provided by `malloc()` and `free()`.

A memory allocator asks the operating system to expand the heap portion of the process's address space by calling `mmap`. The memory allocator then gives out this memory to the calling process. This involves managing a free list of memory and finding a contiguous chunk of memory that is large enough for the user's request; when the user later frees memory, it is added back to this list.

Here is the complete, *exact* API you must support.

- `int initMemory(int sizeOfRegion)`: called one time by a process using your routines; `sizeOfRegion` is the number of bytes that you should request from the OS using `mmap`. Note that you may need to round up this amount so that you request memory in units of the page size (see the `man` pages for `getpagesize()`). The function returns 0 on success and -1 if it is unable to obtain a memory region of that size.

- `void * allocateMemory(int size)`: similar to the library function `malloc()`. It takes as input the size in bytes of the memory block to be allocated and returns a pointer to the start of that memory block. The function returns NULL if there is not enough free space within `sizeOfRegion` allocated by `initMemory` to satisfy this request.

- `int freeMemory(void *ptr)`: frees the memory block that `ptr` falls *within*. Just like with the standard `free()`, if `ptr` is `NULL`, then no operation is performed. The function returns 0 on success and -1 if `ptr` does not fall *within* a currently allocated memory block (note that this includes the case where the object was already freed with `freeMemory`).

- `int isMemoryValid(void *ptr)`: This function returns 1 if `ptr` falls *within* a currently allocated memory block and 0 if it does not.

- `int getMemorySize(void *ptr)`: If `ptr` falls *within* the range of a currently allocated memory block, then this function returns the size in bytes of that memory block; otherwise, the function returns -1.

Your implementation of this memory allocator should observe the following:

- When requesting memory from the OS, you must use `mmap`.
- Your memory allocator must call `mmap` only one time (when it is first initialized).

- As you may have realized from the API, your memory allocator will be slightly more "sophisticated" than the traditional `malloc` and `free` in that it will be flexible in how the user can specify what memory should be freed.
- Your implementations of `allocateMemory(int size)` and `freeMemory(void *ptr)` are identical to `malloc` and `free`, except the `ptr` passed to `freeMemory` does not have to have been previously returned by `allocateMemory`; instead, `ptr` can point to any valid range of memory returned by `allocateMemory`.

  For example, the following code sequence is valid with your allocator, but not with the traditional `malloc` and `free`:

```
int *ptr;

// The returned memory object is between ptr and ptr+49
if ((ptr = (int *)allocateMemory(50 * sizeof(int))) == NULL)
      exit(1);

// Could replace 30 with any value from 0 to 49
freeMemory(ptr+30);
```

- You are free to use any data structures you want to manage the free list but you should use the standard **best-fit algorithm** to manage the free space. You will probably need a more sophisticated data structure than the traditional `malloc` to track the regions of memory allocated by `allocateMemory`. Specifically, this data structure will allow you to efficiently map any address to the corresponding memory block or to determine that there is no corresponding memory block.
- You may include external implementations of basic data structures, such as heaps, priority queues, etc. in your submission as long as you clearly state the source of these libraries.
- To test your program, I will use/provide my own test C code that attempts to allocate/free various blocks of memory using this API and then use that allocated memory for specific purposes.

Submission

To complete this assignment, simply submit your C code files, along with a make file, zipped up in a single folder, to the D2L submission dropbox folder **Project 4** by the deadline specified.