Homework 5: bvMalloc

Due: 11:59 PM, October 22, 2019

In this assignment you will create your own light-weight `malloc` simulation. You will be responsible for maintaining your own virtual address space and allocating portions of it as a user calls your implementations of `malloc` and `free` which will be named `bvMalloc` and `bvFree` respectively. You will implement two allocation strategies: Best Fit and First Fit.

Set Up

First off, this is a simulation that will utilize the heap but with an extra level of indirection on top. All of the memory that will be managed here will live in the dynamic region but you will manage all allocations to it yourself. Your virtual heap will be fixed in size and you will not be required to increase or decrease this total amount. This means that there will be no virtual calls to `sbrk` to increase your working space. On the first invocation of `bvMalloc` you are to allocate (via `malloc`) 1000 bytes to work with. This will represent your virtual heap that you will do all of your allocations and de-allocations to. Once you have created your 1000-byte virtual heap, assume that the full amount is free and available for use.

This assignment will be like others in that the entirety of your code will live in a file called `bvMemory.h`. Again, I will interact with your code through various function calls (to `bvMalloc` and `bvFree`). Since your code will need to maintain state in between my calls to your functions, you must utilize global variables within your header file. This is where you will maintain any of your data structures that you will use to keep track of the memory allocations. Do not allocate these structures within your virtual heap! The virtual heap should be for user data only.

`void* bvMalloc(int size, int bestFit);`

The prototype for `bvMalloc` is a bit different than that of actual `malloc` because, again, we're simulating things here.

First off, `bvMalloc` actually has one extra argument. The first argument is similar to the argument passed to `malloc` – `size` denotes the number of bytes to be allocated for the data. This second argument is an integer that denotes if the memory allocation strategy should be a "best fit" strategy. If the value is 1, then the best fit strategy should be used. If the value is 0, then the first fit strategy should be used.

The return type of `bvMalloc` must be a pointer to the data that has been allocated. Expect that the user will write to this data! If you find that you cannot allocate a contiguous region of data in your virtual address space of `size` bytes, then you should return 0 (which is NULL). A value of 0 will be taken to mean that there was a failure and that the memory could not be allocated within your virtual address space. All other returned values should correspond to the location of data allocated for the user.

It is important to note that the argument `size` will always be strictly greater than 0 and that the argument `bestFit` will always be either 0 or 1.

The best-fit policy will choose to allocate the data into the smallest free slot of memory that can fit the data.

The first-fit policy will choose to allocate the data into the first slot of memory that it can fit in. Lower addresses must always be utilized before larger addresses.

HINT: The first call to `bvMalloc` (no matter first-fit or best-fit) should **always** return the address of the beginning of your virtual address space as this location is appropriate for either strategy when faced with an entirely empty working space.

`void bvFree(void* address);`

Your implementation of `bvFree` should be similar to that of `free`. The provided argument will be an address that you have previously allocated via a call to `bvMalloc`. Your goal is to free up that location in your virtual address space.

Under no circumstance will I ask you to free an address that is not currently allocated.

Submission: Turning in the assignment

The result of your work should be saved in the following location:

```
\\CS-Data\Students\your_user_name\cmsc432\hw_5\bvMemory.h
```

If you fail to save the file exactly as it is written here, you will be marked down by 4 points for failure to follow instructions and hindering grading scripts. Similarly, make sure that your prototypes for `bvMalloc` and `bvFree` **exactly** match the descriptions in the document.