Computer Systems
Project 4
Fancier allocators

1 Preparing for mark-sweep garbage collection

For this project and the next one, we will be working with a *best fit* allocator.[1] It keeps a doubly linked-list of free blocks; when an allocation is requested, it searches that list for the closest fit. In the absence of an acceptable free block, it pointer-bumps to expand the heap.

Our goal is to prepare this allocator for use in a *mark-sweep garbage collector*. The collector will use this allocator to create new blocks. It will traverse the heap to *mark* live blocks. It will then need to *sweep* allocated blocks that are *dead*, freeing each one.

The key question here is: How will the GC **find** the allocated-and-dead blocks? We will need to modify this allocator to support that operation.

2 Getting started

2.1 Creating the repository

1. **Login to the server:** Connect to the course server.
2. **Login to GitLab:** From your browser, login to `https://gitlab.amherst.edu`
3. **Start a new project:** On the top toolbar of the GitLab window, click the little drop-down menu marked by a plus-sign. Select `New project`. Set the *Project name* to be `sysproj-4`, and leave the other default values. Click on the `Create project` button at the bottom.
4. **Clone the repository onto the course server:**

```
$ git clone git@gitlab.amherst.edu:yourusername/sysproj-4.git
$ cd sysproj-4
```

5. **Download the source code:** After you download the files, use `ls -l` to list the directory and see what you have.

```
$ wget -nv -i https://bit.ly/cosc-171-20F-p4
$ ls -l
```

[1] This choice of allocator is a change of plans. We had discussed using a *segregated fits* allocator, but I wanted to keep the complexity of the project reasonable. I will provide the segregated fits allocator for the adventurous, but we will focus our efforts on the best fit implementation.

6. **Add/commit/push the source code to the repository:**

```
$ git add *
$ git commit -m "Starting code."
$ git push
```

2.2 Compiling and running

This collection of code works identically to the code from Project-3. To compile the pieces:

```
$ make clean
$ make libbf memtest
```

Then, to run `memtest` (or anything else) with your allocator code, and then turn off your code:

```
$ export LD_PRELOAD=${PWD}/libbf.so
$ ./memtest
$ unset LD_PRELOAD
```

3 Your assignment

3.1 Part I: Commenting the code

Open `bf-alloc.c`. Its basic structure matches that of `pb-alloc.c` from the previous project. There are some key differences, including the definition of the `header_s`, which now contains additional fields for organizing each block.

Notice that `malloc()` and `free()` are devoid of comments. **Comment these functions**, providing a guide to any reader of the code as to what is happening in each group of lines. As before, you may collaborate freely with others in figuring out the code and writing these comments.

3.2 Part II: Make it align

This allocator does **not** provide double-word aligned blocks in the way that it should. **Port your code from Project-3**, making the blocks that are created by pointer-bumping be properly aligned.

3.3 Part II: Create an *allocated list*

As written, objects are only even linked into the *free list*; that is, when an object is allocated, it isn't on a list at all.

So that a garbage collector may search the allocated blocks, looking for dead ones during its *sweep* phase, you must **create a linked list of allocated blocks**. When a block is allocated, add it to this list; when freed, remove it from this list before adding it to the free list.

3.4 Part IV: Test your code

Modify the testing program, `memtest`, to do some allocations, deallocations, and reallocations. Make sure that it still works properly with all of your code changes.

3.5 Totally optional challenge

Notice that your repository includes `sf-alloc.c`, a *segregated fits* allocator. Preparing this allocator for use in a mark-sweep GC is different. It would require enhancing the headers, at the top of each page, to provide *per-block bitmaps* to keep track of which blocks are *allocated*, and, to support a heap traversal, which blocks are *visited/live*.

If you are interested in trying to add these things to this more complex allocator, let me know, and we can discuss how that would be done.

4 How to submit your work

First, be sure that the most recent versions of your work are up-to-date on the GitLab server by performing an *add/commit/push* with `git`. Then, go to GitLab with your browser, and add me (*sfkaplan*) as a *Developer* to your repository.

This assignment is due on Sunday, Oct-04, 11:59 pm.