Memory Manager

CS453: Operating systems

Overview

In this project we will implement our own memory management. More specifically, we will replace malloc/free with your own memory management scheme based on the Buddy system discussed in class.

This project is challenging and you may not be able to get it to work fully. However, you will learn a lot about memory management along the way!

Startup

Run the **git pull --rebase** command in your backpack folder (on the master branch). You should see a p6 project folder in your backpack with all the starter code. It should look like the listing below:

```
[user@localhost p6(master)]$ ls -R
.:
backpack.sh  buddy-non-preload  buddy-preload  buddy-system-movie.mpg
Buddy-System-notes.pdf  Makefile

./buddy-non-preload:
buddy.c  buddy.h  buddy-test.c  buddy-unit-test.c  Makefile  malloc-test.c

./buddy-preload:
buddy.c  buddy.h  Makefile  malloc-test.c
[amit@kohinoor backpack(master)]$
```

First, you will be completing the `buddy.c` and `buddy-unit-test.c` files in the `buddy-non-preload` subfolder. Make sure to study the header file `buddy.h` for the javadocs of functions that you have to implement. Then you will port the code over to the `buddy-preload` folder. The porting only involves changing some function prototypes.

Specification

Buddy System Memory Management

Implement your own memory manager using the Buddy Algorithm. You should use the **sbrk()** to initially allocate a large block of memory. A good initial amount is 512MB. From there on manage the chunk of memory returned by **sbrk** using your own memory management functions.

Note that you will have to store the data structures used to represent the buddy system somewhere in memory. You may statically declare these pointers in you code (in the data segment). For this purpose you may assume that the maximum amount of memory that you will ever manage is limited to 32GB. The pointers associated with each free block in the buddy system should be stored in the block as explained in the Buddy system algorithm.

Have all the initialization be in a separate function. If the user doesn't call this function, then it is transparently called whenever the user calls buddy malloc/calloc for the first time. You can test for that using a static variable. We will use the following prototypes for the buddy functions:

```
void buddy_init(size_t);
void *buddy_calloc(size_t, size_t);
void *buddy_malloc(size_t);
void *buddy_realloc(void *ptr, size_t size);
void buddy_free(void *);
```

Note that if a 0 is passed as an argument to **buddy_init**, then it initializes the memory pool to be of the default size (512MB, as specified above). If the caller specifies an unreasonably small size, then the buddy system may not be able to satisfy any requests.

If the memory cannot be allocated, then your **buddy_calloc** or **buddy_malloc** functions should set the errno to **ENOMEM** (which is defined in **errno.h** header file).

Note that we have provided you with a `buddy.h` header file that contains all the declarations and prototypes. We have also provided a skeleton **buddy.c** that has the declaration for the pool and the table of lists for the buddy system.

Testing

Build a test suite for your buddy system. Sample test code that you can use as a starting point is in the `buddy-non-preload` folder. It contains two performance test files:

`buddy-test.c` and `malloc-test.c` that run identical tests using the two different allocators. **You should not modify these two files!** In addition, there is a unit test file `buddy-unit-test.c`, to which you are **required** to add more tests.

Interposing Malloc

Interposing allows us to add our library in as a shim. Thus when malloc/free/realloc/etc. are called our versions are called instead of the C standard library versions. To use interposing, your buddy system allocator will need to implement the same interface as malloc (and with the same function names and signature). We will make our buddy system into another shared library, which will be named **libbuddy.so**.

Use the following command to interpose for malloc using libbuddy.so:

```
LD_LIBRARY_PATH=.:../list/lib LD_PRELOAD=./libbuddy.so ./mydash
```

The command says to search for the preloaded library in the current folder (we also included `../list/lib` so the program can find the libmylib.so library). However if the mydash program calls `chdir` system call, its current directory changes to something else and the system will no longer be able to find the buddy library. The solution to that is to give the full path of the current folder:

```
LD_LIBRARY_PATH=/home/faculty/amit/cs453/buddy/:../list/lib \
LD_PRELOAD=libbuddy.so ./mydash
```

but that would be different for everyone so a neat solution is to use the pwd command to get the full path of whatever the current directory is where the buddy library is located.

```
LD_LIBRARY_PATH=$(pwd):../list/lib LD_PRELOAD=./libbuddy.so ./mydash
```

The command `$(pwd)` runs the command pwd and returns the output as the right hand side of the assignment.

Note that to time the interposed version, use the following command:

```
time LD_LIBRARY_PATH=$(pwd) LD_PRELOAD=./libbuddy.so ./malloc-test
<appropriate arguments>
```

Integrating with your shell

Use the interposing version of your buddy system to integrate with the shell. Run all the base tests on your dash to check that it works well with the buddy memory allocator.

Integrating with other programs

Now you can use preloading to test your buddy system with any program on your system! Be warned though that this may not work as most programs are multi-threaded and your buddy system isn't (unless you do the extra credit). You would also need to initialize the buddy system to have more memory (8G or more).

Extra Credit

Buddy System Performance

Test your buddy system implementation against malloc and make sure that it outperforms it. For the purposes of measuring performance, use the buddy-test.c and malloc-test.c code provided in the sample code for this project. Here is performance comparison for the reference solution (tested on onyx, compiled with -O2 optimizer flag)

```
[user@onyx buddy-non-preload]$ time buddy-test 20000000 1234 s
real    0m1.348s
user    0m1.345s
sys     0m0.002s

[user@onyx buddy-non-preload]$ time malloc-test 20000000 1234 s
real    0m2.090s
user    0m2.088s
sys     0m0.001s
```

Thread Safety

Make your buddy system library be thread-safe. Compare its run time performance against malloc and report in your README.md file.

References

- Read Section 5.4 (Address Arithmetic) and Section 8.7 (Example--A Storage Allocator) from the C book by Kernighan and Ritchie to help you get started.
- Donald Knuth. Fundamental Algorithms. The Art of Computer Programming 1 (Second ed.) pp. 435-455. Addison-Wesley.
- Look up Buddy Memory System on Wikipedia.

Submission

Project Layout

- The top level of your submission directory for this project must have a Makefile that builds the both the preloaded and non-preloaded versions and other related text programs. The structure of the folder should be the same as what was given to you via backpack.

- Make sure to not modify the two test programs buddy-test.c and malloc-test.c.

- To test the buddy system library with the mydash project, you can simply copy your buddy system library to your p3 project and preload it to test it with your mydash project. This is how we will test your project. Make sure that your p3 project is up to date so that when you make a branch for this project, we will have the latest version of p3 project on it.

- The README.md file should on the top-level of your p6 folder. Other than the usual stuff, it should clearly document what parts of this assignment you have attempted.

- Prepare your directory for submission by removing all executables and object files and other clutter. You should only have the source code, README.md file, Makefiles, test scripts and test files before submitting.

Files committed to git (backpack)

Run the following commands

1. make clean
2. git add <file ...> (on each file!)
3. git commit -am “Finished project”
4. git branch buddy
5. git checkout buddy
6. git push origin buddy
7. git checkout master

Check to make sure you have pushed correctly

- Use the command **git branch -r** and you should see your branch listed (see the example below)

```
$ git branch -r
  origin/HEAD -> origin/master
```

origin/master
origin/buddy

Submit to Blackboard

You must submit the sha1 hash of your final commit on the correct branch. Your instructor needs this in order to troubleshoot any problems with submission that you may have.

- git rev-parse HEAD

Grading Rubric

Provided via backpack.