My malloc using mmap

Johan Montelius

HT2016

1 Introduction

This is a version of the Mylloc experiment but here we will use the system call `mmap()` instead of the system call `sbrk()` when we're asking for memory from the operating system. If you're running on an OSX machine this is probably what you want to do since `sbrk()` is deprecated and if it is used,it is implemented in terms of `mmap()`. It can also be interesting to look at even if you have completed the first assignment using `sbrk()` since you will learn how to use `mmap()` to do what we want.

You need to read this tutorial together with the original tutorial. In this short tutorial we will implement our own version of `sbrk()` so that the rest of the Mylloc system can run without any modifications. If one would do a `mmap()` solution from scratch this is probably not the most straight forward solution.

2 Implement sbrk()

We will implement the first simple version of malloc as described in the Mylloc paper. What we need is therefore an implementation of `sbrk()`, if we can fake this we're home. Create a file called `sbrk.c` and start coding.

In the first part of `sbrk.c` we use a GCC extension to the C language. We're declaring a function, `init()`, to be a *constructor*. This means that the function will be called when the program is loaded. The feature is normally used by shared libraries but we will make use of it in our experiment (does this work using Clang?).

```
#include <stdlib.h>
#include <sys/mman.h>

#define MAX_HEAP 64*1024*4096

char *heap;

char *brkp = NULL;
char *endp = NULL;

static void init() __attribute__((constructor));
```

```
void init() {
  heap = (char *)mmap(NULL, MAX_HEAP,
                      (PROT_READ | PROT_WRITE),
                      (MAP_PRIVATE | MAP_ANONYMOUS), -1,
                          0);
  brkp = heap;
  endp = brkp + MAX_HEAP;
}
```

What we will do in the initialization is to allocate a heap of our own. We do this with a call to the `mmap()` system call. The procedure will allocate a huge area (`MAX_HEAP` big) that is read and writable. It is a private area and is not the mapping of a file i.e. simply new virtual memory.

Once we know this we can implement a procedure `sbrk()` that when requested will increment the `brkp` pointer and return a pointer to the start of the allocated block.

```
void *sbrk(size_t size) {
  if(size == 0) {
    return (void *)brkp;
  }
  void *free = (void *)brkp;

  brkp += size;
  if(brkp >= endp) {
    return NULL;
  }
  return free;
}
```

We know that `sbrk()` should return the *top of the heap* when called with a zero argument so this is what we do. If we have reached the end of the heap we return `NULL` which is also fine according to the documentation. We could of course do another call to `mmap()` to get more memory but this is fine for our purposes.

If you now compile this file we will have an object file that implements `sbrk()`.

```
> gcc -c sbrk.c
```

We can then link everything together and doing so the call to `sbrk()` in `mylloc.c` will use our version.

```
> gcc -o bench sbrk.o rand.o mylloc.o bench.c -lm
```

If you look carefully you will notice that our mmapped heap is not located at the same place as the regular heap.