CS33 Homework Assignment 10 Solutions

Fall 2019

1. In lecture 24 we discussed how linking takes place and, in particular, how it works using ELF. Slide XXIV-42 gives the relocation section and the object code for *subr*. Show the modified object code after it has been relocated using the information given in slide XXIV-46. (Keep in mind that the architecture is little-endian; be sure to read the notes that go along with Slide XXIV-46.)

   Answer:

```
 0:   48 83 ec 08
 4:   89 fe
 6:   bf 00 25 41 00          address of .rodata
 b:   b8 00 00 00 00
10:   e8 1b 00 00 00          PC-relative address of printf
15:   8b 35 25 3b 01 00       PC-relative address of X
1b:   bf 01 25 41 00          address of .rodata+1
20:   b8 00 00 00 00
25:   e8 06 00 00 00          PC-relative address of printf
2a:   48 83 c4 08
2e:   c3
```

2. *Malloc* and *free* were designed to work well in situations in which not much, if anything, is known about the pattern of allocation and free requests that a program might make. Because the sizes of requests are not known in advance, there can be significant fragmentation issues as well as time wasted due to searching for sufficiently large free blocks. But suppose a program dynamically allocates memory to accommodate variables of just, say, six datatypes — u_t, v_t, w_t, x_t, y_t, and z_t, each of a different size (assume all sizes are relatively small). Though it's still the case that the order of allocation and free requests is essentially random, explain how one might be able to exploit the fact that all requests are from the six sizes so that allocation and freeing is much more efficient than it would be if they were done strictly with malloc and free. Your approach should be built on top of *malloc* and *free*, i.e., it calls them as subroutines. You don't need to provide code, but simply explain the approach.

   Answer: One possibility is to have a separate pool of memory for each datatype. Each pool might be composed of blocks whose size is the least common multiple of the sizes of each of the datatypes. These blocks are allocated and freed via calls to *malloc* and *free*. When a particular datatype is allocated, storage is taken from such a block from the datatype's pool. If there are no available blocks, then one is malloc'd. When a datatype is freed, its storage goes back to the block in its pool. If the block is completely free, its storage is released by calling free. Thus all blocks handled by *malloc* and *free* are of the same size, minimizing fragmentation and searching issues.

This procedure is the basis of *slab allocation*, a technique used in many operating systems for allocation of OS memory. A paper describing it may be found at https://www.usenix.org/legacy/publications/library/proceedings/bos94/full_papers/bonwick.a.