Homework 4

Virtual Memory and I/O

1. Suppose we have a machine with a 2^48 (48 bit) virtual address space, a maximum of 2^36 bytes of physical memory, and a page size of 2^16 bytes. Assume a single-level page table to begin.
   a. Draw a virtual address, splitting it into the virtual page number and the virtual page offset. Note how many bits are used for each.
   b. Draw a plausible page table entry, showing the length in bits of each of the fields. How many of your entries fit on a page?
   c. Assume virtual addresses from 0x100000000000 to 0x1000ffffffff and from 0x700010000000 to 0x9fffffffffff are in use. How much space does your page table occupy?
   d. Describe a plausible 2-level page table approach for this system and repeat a-c assuming that approach.

2. We talked in class about mapping the same shared library into each application's address space, letting us get by with having only one copy of the library's code in memory. However, real libraries also have data. For example, the gethostbyname() function returns a pointer to a hostent structure statically allocated in the library:

```
struct hostent thehostentry;

struct hostent *gethostbyname(const char *name) {
   ...write thehostentry...
   return &thehostentry;
}
```

   We might expect that all applications using this shared library would share the same copy of thehostentry with disastrous results, but, in fact, each application seems to have its own thehostentry, just as if the library were statically linked to the application. Describe how such semantics could be implemented using virtual memory mechanisms. You may want to re-read the section on copy-on-write in your textbook.

3. Your book talks about mark-and-sweep garbage collection. A completely different approach is known as stop-and-copy garbage collection. In this approach, the heap memory is divided in half into the "working memory" and the "free memory". The program always uses just the working memory. When the program runs out of memory, it stops and garbage collection is invoked. As the garbage collector traverses the graph of nodes reachable from the root nodes, it copies each reachable node to the free memory, placing the nodes sequentially in the free memory in the order in which they are traversed. By means of mechanisms that are not important here, each node is copied only once, and all the pointers between the reachable nodes

are updated appropriately. Then, the garbage collector simply starts treating the free memory as the working memory and vice versa, essentially throwing away all the unreachable nodes en masse. One often finds that stop-and-copy greatly increases cache hit rates, especially for pointer-based data structures such as lists, trees, and graphs. Why?

4. Traceroute (/usr/sbin/traceroute) lets you trace the route which your packets take from source to destination. Read about traceroute and find the routes to several of your favorite sites. Ping lets you measure the round-trip latency to a host. Use ping to determine the latencies to several sites. Dig (/usr/bin/dig) and whois (/usr/bin/whois) let you find detailed information about DNS names. Try them out. There is nothing to hand in here.

5. The read and write system calls are designed for byte streams. You can also use them with packet-oriented communication, for example with UDP (the Unreliable Datagram Protocol). In UDP, each write call emits a packet. However, packets have a minimum and a maximum size. What does (or should) the system do if a write is for fewer or more bytes than these limits? This is not an error condition.