CS360 -- Lab 2 -- Buffering

- Jian Huang
- CS360
- Url: http://www.cs.utk.edu/~huangj/cs360/360/labs/lab2/lab.html

---

This lab gives you some practice with **read()**, **fread()**, and buffering.

First, in whatever directory you'll be writing code, do the following:

```
UNIX> ln -s ~huangj/cs360/labs/lab2/hosts
UNIX> ln -s ~huangj/cs360/labs/lab2/converted
```

This will make links to the files **hosts** and **converted** which are in my lab directory. We want you to make a link instead of a copy to save on disk space. When you're done with this assignment, delete these links and any copies of these files. This is because these files are 335K and 488K respectively. It won't take many students to copy these files and kill the disk space allotted to students.

Now, look at the **hosts** file:

```
UNIX> head hosts
128.169.202.200 cocker.vet.utk.edu      cocker
128.169.201.208 berkelium.chem.utk.edu  berkelium
128.169.201.152 cherokee.gg.utk.edu     cherokee
192.31.99.193   slipgate1.utk.edu       slipgate1
192.249.11.79   csmacw14.cs.utk.edu     csmacw14
128.169.94.12   cygnusx1.cs.utk.edu     cygnusx1
128.169.93.24   pepsi.cs.utk.edu        pepsi
128.169.92.48   wham-o.cs.utk.edu       wham-o
127.0.0.1       localhost       loghost
128.169.202.248 mccall.oac.utk.edu      mccall
UNIX>
```

Each line of the **hosts** file refers to a machine somewhere at UT, ORNL, or Princeton. Each machine has an ``IP'' address, which is the first word on the line. This address consists of four numbers between 0 and 255, and is how machines are accessed on the internet. Next are all the names by which this machine is known. You'll note that most machines have just two names -- a local name (without dots, e.g. ``cocker''), and an absolute name (with dots, e.g. ``cocker.vet.utk.edu''). Some machines have only local names (like ``localhost''), and some have more than one local name (like 128.112.128.1, which goes by ``princeton.edu'', ``princeton'', and ``newsserver''). Whenever a machine has an absolute name, it also is known by a local name which is the prefix of the absolute name (e.g. ``cocker'' is the prefix of ``cocker.vet.utk.edu'').

Now, the file **converted** contains the same information as **hosts**, only it's in a terser form. **Converted** is just a stream of bytes adhering to a specific format. Like **hosts**, it contains successive entries for each machine; however, you can't read **converted** with **head** or **vi**. Instead you must write a program to read it. Each machine entry has the following form:

- The first four bytes are the four bytes of the IP address. In other words, the first four bytes **converted** are 128, 169, 202, 200 --- this is the IP address of ``cocker''.
- The next four bytes are interpreted as an integer. This integer contains the number of names by which the machine is known.
- The next bytes contain the names of the machine, null terminated.
- The next entry starts immediately after the last null character.

To save on space, if an entry contains an absolute machine name, the local name corresponding to the prefix of that absolute name is ommitted.

Thus, the first 58 bytes of **converted** contain the information for the machines ``cocker.vet.utk.edu'' and ``berkelium.chem.utk.edu'':

x	Byte *x*	Byte *x+1*	Byte *x+2*	Byte *x+3*	Byte *x+4*	Byte *x+5*	Byte *x+6*	Byte *x+7*
1	128	169	202	200	0	0	0	1
9	99 (**'c'**)	111 (**'o'**)	99 (**'c'**)	107 (**'k'**)	101 (**'e'**)	114 (**'r'**)	46 (**'.'**)	118 (**'v'**)
17	101 (**'e'**)	116 (**'t'**)	46 (**'.'**)	117 (**'u'**)	116 (**'t'**)	107 (**'k'**)	46 (**'.'**)	101 (**'e'**)
25	100 (**'d'**)	117 (**'u'**)	0 (**'\0'**)	128	169	201	208	0
33	0	0	1	98 (**'b'**)	101 (**'e'**)	114 (**'r'**)	107 (**'k'**)	101 (**'e'**)
41	108 (**'l'**)	105 (**'i'**)	117 (**'u'**)	109 (**'m'**)	46 (**'.'**)	99 (**'c'**)	104 (**'h'**)	101 (**'e'**)
49	109 (**'m'**)	46 (**'.'**)	117 (**'u'**)	116 (**'t'**)	107 (**'k'**)	46 (**'.'**)	101 (**'e'**)	100 (**'d'**)
57	117 (**'u'**)	0 (**'\0'**)						

Part 1

Write the program **l2p1**. This program reads host information from the file **converted** into a red-black tree keyed on machine names. After reading in the file, it prompts the user to enter a machine name, and then prints out the IP address, and all names for that machine. The red-black tree should include both local and absolute names. It should also contain local names that are prefixes to absolute names, even though those names are not in the **converted** file (i.e. ``cocker'' and ``berkelium'' should be in the red-black tree, even though they are not explicitly in the **converted** file). You may assume that there are no duplicate local names in the **converted** file. Use buffered (standard) I/O routines to read in the **converted** file. In other words, use **fread()**, **fgetc()**, **fscanf()**, etc. --- whichever works best for you. **~cs360/lab2/l2p1** is an executable file that works as yours should. When in doubt, see what this program does. The following is example input and output from **l2p1**:

```
UNIX> ~cs360/lab2/l2p1
Hosts all read in

Enter host name: hydra3e
128.169.94.137: hydra3e.cs.utk.edu hydra3e

Enter host name: sun4server
128.112.130.225: commonserver decmipsserver decmserver diskfarm.princeton.edu diskfarm homeserver irisserver mailserver oedfileserver rtpcserver sun3server

Enter host name: duncan
```

```
128.169.94.83: duncan.cs.utk.edu duncan

Enter host name: xxx
no key xxx

Enter host name: < CNTL-D >
UNIX>
```

If you try to run **l2p1** and it says:

```
l2p1: converted: No such file or directory
```

then you have not made a link to **~huangj/cs360/labs/lab2/converted**. Do:

```
UNIX> ln -s ~huangj/cs360/labs/lab2/converted
```

and try again. You'll notice that it takes a few seconds for **l2p1** to read in the **converted** file. You can time this using **time**, and using the null file **/dev/null** as input:

```
UNIX> time ~cs360/lab2/l2p1 < /dev/null
Hosts all read in

Enter host name: 3.4u 0.6s 0:04 99% 0+1824k 1+0io 0pf+0w
```

This says that the program took 3.4 seconds of user time, 0.6 seconds of system time, and 0:04 (4 seconds) wall clock time. Your program should have a comparable time (between 3-5 seconds -- mine will be faster because I compiled it with the **-O** flag of **gcc**. This flag tries to make the code as fast as possible. You can try this too if you want).

Part 2

Write the program **l2p2**. This program is the exact same as **l2p1** except you must use only the system calls **open()**, **close()** and **read()** to read in the file **converted**. (You can use buffered I/O to read/write from standard input and standard output, but *not* from the file **converted**). Don't worry about efficiency -- just make the program work. If you have structured things right, this should take just a few modifications to **l2p1.c**. **~cs360/lab2/l2p2** is an executable file that works as yours should. Note that it is *considerably* slower than **l2p1**:

```
UNIX> time ~cs360/lab2/l2p2 < /dev/null
Hosts all read in

Enter host name: 4.7u 21.9s 0:26 99% 0+1796k 5+0io 5pf+0w
UNIX>
```

You may well get running times in the 30--40 second range.

Part 3

Write the program **l2p3**. Your job is to take **l2p2** and make it run as fast as **l2p1**. You may still use only **open()**, **close()** and **read()** to read in the file **converted**. You will have to add your own buffering. By this, I mean you should only call **read()** once. Make your buffer **350000** bytes so that you can read in all of **converted** at once. There is a working executable in **~cs360/lab2/l2p3**:

```
UNIX> time ~cs360/lab2/l2p3 < /dev/null
Hosts all read in

Enter host name: 3.0u 0.7s 0:04 76% 0+2064k 41+0io 41pf+0w
UNIX>
```

Advice

As usual, when doing this lab *start small and build your way up*. In other words, don't even think about inserting anything into a rb-tree until you can successfully read the **converted** file and interpret its contents. Then start to worry about the rb-tree. Your rb-tree should be keyed on machine name, and have a **val.v** field which points to a struct like the following:

```
struct ip {
  unsigned char address[4];
  Dllist names;
};
```

If you want to have the names printed out alphabetically, use a rb-tree instead of a dlist for **names**. However, when you first start to write rb-tree code, don't bother with the struct. Instead, just have the ip address be the **val.v** field. Add the struct once you get the rb-tree lookup working. Your rb-tree code should be identical for all three parts.

If you get a bus error in l2p3

There's a good chance that when you first try **l2p3**, you'll get a bus error. This is because you will try to access the number of names for an entry in your buffer as an integer, but it won't be aligned. For example, suppose you have a buffer **s**, that contains all of **converted**, and you know that the number of names for the second entry is at **(s+13)**. If you try something like:

```
  char *s;
  int *i, count;

  i = (int *) (s+13);
  count = *i
```

then you will get a bus error. Instead, use **memcpy(&count, s+13, sizeof(int))**. Read the man page on **memcpy()** if this is sounding a little confusing to you.