CS 221 – Lab #4

Today, we have two objectives. First, to make sure all 8 of your machines work. And second, to practice some more C.

Finishing the cluster

By now, all 8 of your machines should be plugged in to electricity and network connections.

1. If you have not done so already, you need to copy the completed "image" onto each SD card. As you initialize each machine's SD card, you also need to give each machine a unique name. (See my handout for Lab 1 on how to do this.) Our convention is to consecutively number our machines. Depending on which corner of the room you are working in, your machines should be numbered pi09 to pi 16, pi17 to pi24, pi25 to pi32, or pi33 to pi40.

2. Test your connections. One of your 8 machines is connected to the keyboard and monitor. We will call it the master node. Be sure you are able to ping the other machines and get a response. In other words, if you are at the cluster 25-32 and your master machine is p25, then ping machines 26 to 32. To ping a machine, just type in the command `ping pi26`, for example. After about 10 seconds hit control-C to terminate the connection test, before going on to pinging the next computer.

   Also, see if you can remotely log in from one machine to another. Use the ssh command to log in remotely. For example, if you are on pi25, say `ssh pi26`. And once you are on pi26, see if you can continue logging in to other machines.

3. Starting tomorrow we will be writing actual parallel programs. To get ready for this task, we need to make one more change. It will be convenient for each machine in your cluster to know the IP address of each of the other machines in the cluster. Record this information here.

2-digit machine number PIxx	IP address

On each machine, edit the /etc/hosts file. At the end of the file, append lines of the form
<ip address><tab><hostname>

For example,
101.102.103.104 pi95

Each line you are appending pertains to one of the machines in the cluster, so you will need 8 lines of this format. Don't delete or modify the existing information in /etc/hosts.

C Practice

First, if necessary, please show me your C programs from Lab 3. Today, I would like to practice some simple I/O skills that should become almost second nature to us.

1. Write a program called longest_word.c . It will read from standard input. The idea is that I have a text file, and I want to know what its longest word is. Rather than doing formal file I/O, we can simply redirect the standard input. For example, to run your program on the input file general.txt, you could type: `longest_word < general.txt` .

   To simplify this task for you, let me suggest an algorithm for solving this problem. You can transcribe this idea into C:

```
A loop to read each line of input.
  get a line of text
  if we have reached the end-of-file,
    break

  Call strtok for the first time on this line.
  Delimiters include ".,?!\t\n "
  if this token is null,
    continue to the next line because this line is essentially blank
  At this point, we know the token exists and represents a word
  Update longest_word, if necessary

  A loop to consider all of the remaining tokens on the line.
    Call strtok to get the next token.
    If the token is null,
      break from the inner loop:  we're done with the line.
    At this point, we know the token exists and represents a word
    Update longest_word, if necessary

Print out the longest_word.
```

After you get the program to work, we can make 2 very simple enhancements. Have your program also count the total number of lines and total number of words that it finds. You can then corroborate your program's output with what `wc` says.

2. Let's generate some random numbers. A lot of them. Write a program called generate.c. Its purpose is to print a list of 10,000 random numbers, one number per line. The random numbers should be positive integers in the range 1..100.

   There is one thing I need to tell you about random numbers. When we run our programs repeatedly, we would like to get different random results. Here is a simple way to accomplish that in C. At the beginning of your `main()` function, just after you declare your variables, include this statement, which will initialize the random number seed:

   ```
   srand(time(NULL));
   ```

   When you are testing your program, make sure that it in fact can generate the numbers 1 and 100. Make sure that numbers outside that range, such as 0, do not appear in the output.

   To make the output more manageable, we should write it to a file. Use I/O redirection like this:

   ```
   generate > output.txt
   ```

   And then you can look at output.txt inside a text editor.

   At some point in the future, the numbers 100 and 10,000 could change, so declare these to be constants in your program.

3. This third and final program is designed to work closely with the program you just finished. Let's analyze the output of your random number generator, and count how many times a certain target value appears. As an example, we are interested in knowing how often the number 9 is generated by generate.c.

   Call your new program count.c. At the top of your program, you should declare a constant TARGET set to 9. The program count.c should read from standard input, and assume that the input consists solely of integers, as in the output from the generate program. The program should count how many of these numbers are equal to our target value 9. At the end of the program, your should print out this number of occurrences.

   How does your program know if it's done reading the input? Here is one way to solve this problem. You can use the scanf() function to read an integer. Note that scanf() itself returns an int. If this int is zero, this means there was nothing in the input that it could convert for you. That means either it ran out of input entirely, or that it encountered some input of the wrong type that it wasn't expecting, such as a string. Since we know that our input consists only of integers, we can assume

that scanf() returning 0 means we are done reading input. If you are interested in the details, type in the command `man scanf`.

Here is my favorite part: testing your program. We would like to run the generate program to give us some input for count. And we would like to test our program repeatedly. A slick way to run both programs in succession is to use the | redirection symbol, like this:

```
generate | count
```

This means that we run the generate program, and its standard output becomes the standard input to count. No output file is actually produced.

Run your program several times to observe that the output is slightly different each time. How many nines should you be seeing?

Finally, I would like you to make one more change. Maybe we aren't so hot about the number 9 after all. Allow the user to specify the target value as a command line argument. You should use the built-in function atoi() to convert a string to an integer. Give the user an error message if the command line argument is missing.

If you feel especially ambitious, you could even add command line arguments to generate.c so that it can accept parameters from the user regarding the range of possible random numbers and how many random numbers to produce.