Design
At the start this program

```
// The main thread will create the dictionary data structure, listen for connections, and
// hand those off to worker threads
int main(int argc, char** argv)
{
}

// This function will read the dictionary file into some other structure for the thread to use
// return type will probably change to return hoWever I store the file
int readInDict ()
{
}

// This will be a thread that is handed a client
// process the spelling request
// might be broken into more pieces
int processRequest ()
{
}

// This will be the thread for logging
int processLog ()
{
}
```

First thing I tried to implement was a switch statement for the command line arguments
I'm getting weird segfaults when I'm running it in the background with & but I can't always replace it with an if else tree if they occur nearer to the end of development

```
    int connectionPort;
    char *dictionary_path;
    FILE *dictp;
    switch (argc)
    {
    case 3: dictionary_path = argv[2];
        printf("case 3");
        connectionPort = atoi(argv[1]);
        break;
    case 2: dictionary_path = DEFAULT_DICTIONARY;
        connectionPort = atoi(argv[1]);
        printf("case 2");
        break;
    case 1: dictionary_path = DEFAULT_DICTIONARY;
        connectionPort = DEFAULT_PORT;
        printf("case 1");
        break;
    default: perror("Too many arguments\n");
        printf("default");
        return -1;
        break;
    }
```

Next I decided to just build the dictionary data structure because it seems like the easiest place to start. An array of strings seems the most reasonable, but since I don't know the dictionary size I'm unsure how the best way to allocate size. I could run through the entire dictionary file twice. I chose the max word length 25 (which is a constant defined in the header by running this bash command on the dictionary:

```
 cat word.txt | sed 's/ /\n/g' | sort | uniq | awk '{print length, $0}' | sort -nr | head
```

which gave

23 electroencephalograph's

as the longest word

during testing of the dictionary I forgot to use a primitive with a large enough value

```
32759 courthouses
32760 courtier
32761 courtier's
32762 courtiers
32763 courting
32764 courtlier
32765 courtliest
32766 courtliness
zach@DESKTOP-R2DH3EL:/mnt/c/Users/Zach/Desktop/3207/Lab1/Project 3$ gcc -o Test dictionaryTest.c
zach@DESKTOP-R2DH3EL:/mnt/c/Users/Zach/Desktop/3207/Lab1/Project 3$ ./Test
@@~

 The arrazysize is 32767

zach@DESKTOP-R2DH3EL:/mnt/c/Users/Zach/Desktop/3207/Lab1/Project 3$
```

```
3 99054 99055 99056 99057 99058 99059 99060 99061 99062 99063 99064 99065 99066 99067 99068 99069 99070 99071 99072 99073 99074 99075 99076 99077 99078 99079 9
9080 99081 99082 99083 99084 99085 99086 99087 99088 99089 99090 99091 99092 99093 99094 99095 99096 99097 99098 99099 99100 99101 99102 99103 99104 99105 9910
6 99107 99108 99109 99110 99111 99112 99113 99114 99115 99116 99117 99118 99119 99120 99121 99122 99123 99124 99125 99126 99127 99128 99129 99130 99131 99132 9
9133 99134 99135 99136 99137 99138 99139 99140 99141 99142 99143 99144 99145 99146 99147 99148 99149 99150 99151 99152 99153 99154 99155 99156 99157 99158 9915
9 99160 99161 99162 99163 99164 99165 99166 99167 99168 99169 99170 zach@DESKTOP-R2DH3EL:/mnt/c/Users/Zach/Desktop/3207/Lab1/Project 3$
```

I forgot to close and reopen or at least seek the start of the file before trying to read in words. Ended up just fseek to the start of the file

After I tested the dictionary (printed entire dictionary, number of entries, along with first and last) I moved on to some skeleton methods in main

the main thread will have to add client sockets to a queue and workers will need to remove them so for now the code looks something like

```
void addToClientQueue(int clientsocket)
{
    getlock;
    {
        clientqueue[nextempty] = clientsocket;
        clients++
        nextempty++;
    }
    releaselock;
}

int retrieveFromClientQueue()
{
    int clientsocket;
    getlock;
    {
        clientsocket = clientqueue[nextclient];
        clients--;
        nextclient++;
    }
    releaselock;
    return clientsocket;
}
```

along with entries for the log queue which will look similar

```
void prepareForLogging(char* word, int found)
{
    char *entry;
    entry = logConcat(word, found);
    getlock;
    {
        logqueue[nextentry] = entry;
        tolog++;
```

```
        nextentry++;
    }
    releaselock;
}

char* takeForLogging(char* word, int found)
{
    char *entry;
    getlock;
    {
        entry = logqueue[nextforlog];
        tolog--;
        nextforlog++;
    }
    releaselock;
}

/*  Slightly wasteful but I just made the entry big enough to hold
    OK or MISSPELLED
*/
char* logConcat(char *word, int found)
{
    char *entry = malloc(strlen(word) + 12);
    strcpy(entry, word);
    if (found == 1)
    {
        strcat(entry, " OK");
    }
    else
    {
        strcat(entry, " MISSPELLED");
    }
    return entry;
}
```

also started on the worker thread loop

```
int processRequest ()
{
    int clientsocket;
    while(1)
    {
        clientsocket = retrieveFromClientQueue();

        establishconnection

        checkword()
        sendresult
        prepareForLogging(result);
        close(clientsocket);
    }
}
```

at this point I'll start filling in code

```
cis-lclient04:~/2107/3207/a>gcc -o test spellcheckTest.c
cis-lclient04:~/2107/3207/a>test
cis-lclient04:~/2107/3207/a>
cis-lclient04:~/2107/3207/a>./test


 The array size is 99171

~~~~~~~~A MISSPELLED@@@@@@@@message MISSPELLEDcis-lclient04:~/2107/3207/a>
```

spell check is not working, I think the dictionary entries all contain returns

```
>>>dog
@@@@@@@@dog 4 | dog OK~512
I actually don't have anything interesting t
>>>cat
d.o.g.@@@@@@@@cat 4 | cat OK~512
I actually don't have anything interesting t
>>>flbe
c.a.t.@@@@@@@@flbe 5 | flbe MISSPELLED~512
I actually don't have anything interesting t
>>>
```

working over network

making sure the arguments are being read properly

```
cis-lclient15:~/2107/3207/a>argTest
case 1
the port 3207 and dict words.txt

cis-lclient15:~/2107/3207/a>argTest 1234
case 2
the port 1234 and dict words.txt

cis-lclient15:~/2107/3207/a>argTest test.txt
case 2
the port 3207 and dict test.txt

cis-lclient15:~/2107/3207/a>argTest 1234 test.txt
case 3
the port 1234 and dict test.txt
```

while testing alternate dictionaries I realized my windows machines created dictionaries with CRLF at the end of line opposed to just LF, so this is now enforced in the code

client testing client works on a single transaction

```
cis-lclient15:~/2107/3207/a/b>client
cis-lclient15:~/2107/3207/a/b>

Server loaded with 99171 dictionary entries

Servicing socket 5
Servicing socket 6
```

working with multiple transactions

```
cis-lclient15:~/2107/3207/a/b>Project3_Server


Server loaded with 99171 dictionary entries

Servicing socket 5
Servicing socket 6
Servicing socket 7
Servicing socket 6
Servicing socket 8
```

```
cis-lclient15:~/2107/3207/a/b>client
WORKERMADE
WORKERMADE
WORKERMADE
WORKERMADE
WORKERMADE
sending
sending

Please enter a word for spellchecking or 'Esc' to exit
>>>
Please enter a word for spellchecking or 'Esc' to exit
>>> sending

Please enter a word for spellchecking or 'Esc' to exit
>>> sending
sending

Please enter a word for spellchecking or 'Esc' to exit
>>>
Please enter a word for spellchecking or 'Esc' to exit
>>> cis-lclient15:~/2107/3207/a/b>
```

successfully testing with client.c