Course: Operating Systems - Semester 1 of 5785
Assignment 4

Directions

A. Due Date: 2 January 2025 at 11:55pm

B. Groups of up to two (2) students may submit this assignment.

C. Code for this assignment (Ass4) must be submitted via Github using the per-assignment private repository opened for you in the OSCourse organization. More details on the repository are found below.

D. There are 100 points total on this assignment.

E. GitHub has autograding tests included that run after each push. The tests and points are not complete, but do cover the basic input/output behavior of the programs you must write. Some test files are found in the `tests/` subdirectory in the repository. Do not remove or modify those files.

   (a) You can find a test script to test the input-output behavior of your tool in the file called `test-script.bat` in the starter code.

   (b) The last two tests with Ctrl+C interruptions are in separate scripts - `test13-two-files-signals.sh` and `test13-two-files-signals.sh`. They work only on Linux.

   (c) Expected outputs for each input-output test are found in the starter code.

F. What to turn in:

   (a) Makefile, all source code (*.c, *.h) and library files necessary to compile and run your programs

   (b) README.md file with the contents described in section 11.

G. **Do not** turn in:

   - Compiled executable files
   - Object files (.o)
   - Output files

H. If your code creates a directory called outfiles as part of your work, ensure that it is not uploaded to your repository. Ensure your GitHub repository does not contain a directory called outfiles.

General Requirements

1. All of the code below must be written in C and compilable and executable in a standard Linux Ubuntu (14+) or Linux Mint (20+) environment.

2. All code must have comments - each function must have an introductory comment detailing its purpose, input parameters, and return value.

3. Use `malloc` or `calloc` for (at least) your buffer and array allocations. Don't forget to use `free` when you're done with them.

Signals, Threads, and Mutual Exclusion: Long Words

In this assignment you will use the Pthreads library to write some user level thread code that uses monitors. The main functionality of the program imitates a map-reduce longest words algorithm (parallel read and synchronization barriers). The main goal of the assignment is to gain familiarity with the Pthreads library for user threads, monitors, and synchronization barriers.

A secondary goal of this assignment is the use of signal handlers in C. We'll use one in the tool to experiment with how its done.

You'll find some help and additional references for Pthreads, mutual exclusion, and monitors at the following links:

- `https://hpc-tutorials.llnl.gov/posix/`
- `https://man7.org/linux/man-pages/man7/pthreads.7.html` (or just `man 7 pthreads`)
- `https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html`

1 Makefile (7 points)

For this assignment, you will need to make a simple Makefile with the following targets:

1. all - the default rule that compiles the executable for the assignment. It should call the longestwords target.
2. longestwords - compiles the object files for the longest words program to an executable called `longestwords`
3. main.o - compiles the main.c file into a file called main.o
4. file-processor.o - compiles the file-processor.c file into a file called file-processor.o
5. clean - a rule that erases all of the .o and executable file generated during compilation. It must leave just the code files and the Makefile.

The name of the file must be `Makefile`

Note: The name of the file must be exactly as above. Do not add any suffixes or endings to the file. For example, do NOT call the file Makefile.txt.

Points

- all target. **2 points**
- longestwords target. **2 points**
- main.o target. **1 point**
- file-processor.o target. **1 point**
- clean target. **1 point**

2 Header Files

C uses header files to expose functionality between code files. The following header files are given as part of the starter code. **Do not** modify them.

1. monitor.h - contains the definition for the monitor, including shared state, lock, and condition variables.
2. file-processor.h - contains the function prototype for the `processFile` function that the threads run.
3. checkin.h - contains the function prototype for the `checkin` function that helps with implementing the synchronization barrier.

3 Code Documentation (8 points)

Add documentation to all functions via a comment just above the function body in all C files. This includes all functions, including ones provided in the starter code.

The function header comment must be in the following format:

```
/* [name of function] [description of function's job]
*  Input: [parameters and what they mean]
*  Output: [meaning of value returned]
*/
```

For example, a header comment on a simple adding function might be:

```
/* Add: adds two integers
*  Input: num1 - first number, num2 - second number
*  Output: Sum of the two numbers
*/
```

4 Program Input

The user will provide the following parameters to the longestwords program:

- The number of longest words to print. Must be a positive integer, any value from 1 to the maximum integer value C supports.
- The files to process. Must be 1 or more files.

Some sample program inputs:

1. ./longestwords 10 simple-tests.txt
2. ./longestwords 100 file1.txt file2 file3 file4
3. ./longestwords 50 file1.txt tests/file2 file1.txt

Note that the number of words must come before the names of the files, that files might be in the current directory or in another directory, and that the same file might be submitted multiples times.

If a non-existent file is given, it should be ignored.

If there is no number given or if there are no valid files given, the program should quit with a usage message.

5 Word Lengths and Uniqueness

We will reuse some code from the previous assignment to create a multi-threaded tool that reads files and prints out the longest words in them. As in the previous assignment, the length of a word is determined by its alphabetic characters only, without respect to any non-alphabetic characters included in it. For example, the following words are all considered length 5:

1. hello
2. hello,
3. ;hello1
4. hello?"
5. h_e-12_ll_o';

Consider only **unique words**. That is, if a word appears multiple times, it will only be output once and only count once toward the total longest words. If a word has non-alphabetic symbols or numbers, each form is a unique word. For example, each of the above words is unique even though the alphabetic characters (hello) are the same.

6 Mutual Exclusion: Monitor

The file processing will be performed using multiple threads that execute concurrently. The words must be stored in a data structure that is shared between the different threads. Therefore, to prevent race conditions and loss of data, you must use a monitor to control access to the shared data structure.

I have provided you with a monitor structure to use. The monitor has the following fields:

1. State fields - the array of words and the array of word counts per length.
2. Synchronization fields - three integers and two condition variables you'll use for the synchronization barrier.
3. Lock - a lock that you'll use to protect the contents of the monitor and state fields.

The monitor's definition is found in `monitor.h`. You should not modify it.

7 Longest Words Processing Steps

The longestwords tool must read through all of the files provided, assigning a separate thread for each file. The threads must run in parallel. The words must be cataloged based on length in the state fields of the monitor so they can be printed out at the end. Use `malloc` or `calloc` to allocate your state fields. While there is no guarantee how many words there will be in the test, you may assume that each individual word will be shorter than 2000 characters.

As an exercise in synchronization, we will force the threads to synchronize after each chapter in the file. All files contain some lines that begin with the word "Chapter" followed by a number. An example input might be:

```
1  Chapter 1
2  Loomings.
3  Call me Ishmael.
4  Some years ago-never mind how long precisely-having little or no money in my purse, and nothing
5  particular to interest me on shore, I thought I would sail about a little and see the watery part
6  of the world.
7  It is a way I have of driving off the spleen and regulating the circulation.
8  Chapter 2
9  Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November
10 in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing
11 up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand
12 of me, that it requires a strong moral principle to prevent me from deliberately stepping into the
13 street, and methodically knocking people's hats off-then, I account it high time to get to sea as
14 soon as I can.
15 This is my substitute for pistol and ball.
16 Chapter 3
17 Chapter 4
18 With a philosophical flourish Cato throws himself upon his sword; I quietly take to the ship.
19 Chapter there is nothing surprising in this.
20 A full Chapter 5 attitude is necessary.
```

Lines 1, 8, 16, and 17 mark the beginnings of chapters. The other inclusions of the word chapter on other lines do not meet the criteria. They are not a the beginning of a line with a number after it.

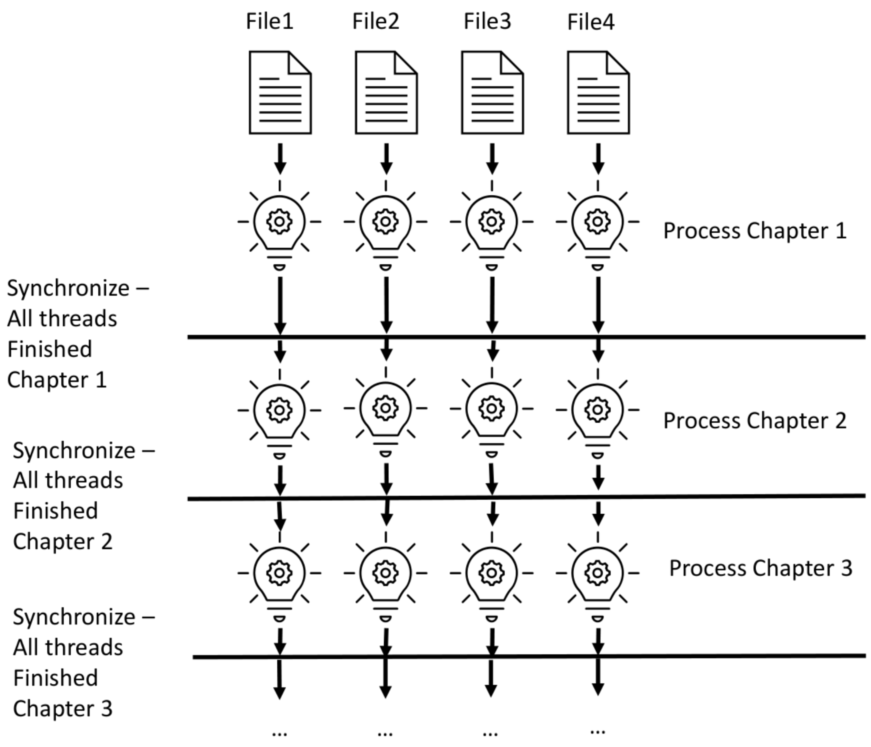
File1
File2
File3
File4
Process Chapter 1
Synchronize – All threads Finished Chapter 1
Process Chapter 2
Synchronize – All threads Finished Chapter 2
Process Chapter 3
Synchronize – All threads Finished Chapter 3
...

Figure 1: File processing steps.

When processing the files, each thread must pause processing until all other threads have reached the end of their current chapter. This process is shown in Figure 1. In the figure, there are four files being processed. Each one starts processing chapter 1. When done, they all wait until all are ready to begin chapter 2. When done, they all wait until all are ready to begin chapter 3. The synchronization continues until all chapters are completed. For convenience, all tests will be with files with an identical number of chapters.

When all threads have finished processing their chapters, the main thread should resume and output the longest words in all of the files. The total number of words output must be same number as was input at the command line.

The output words must be alphabetized within their word lengths. Print each word along with its overall index number. Print a summary line per word length showing how many words were in the word length category. Figure 2 shows the output final steps.

7.1 Processing Example

Consider the following three input files:

file1:

```
Chapter 1
Hello there consequently I am
Chapter 2
ready willing and able able able
Chapter 3
consequently arithmetic
```

file2:

```
Chapter 1
```

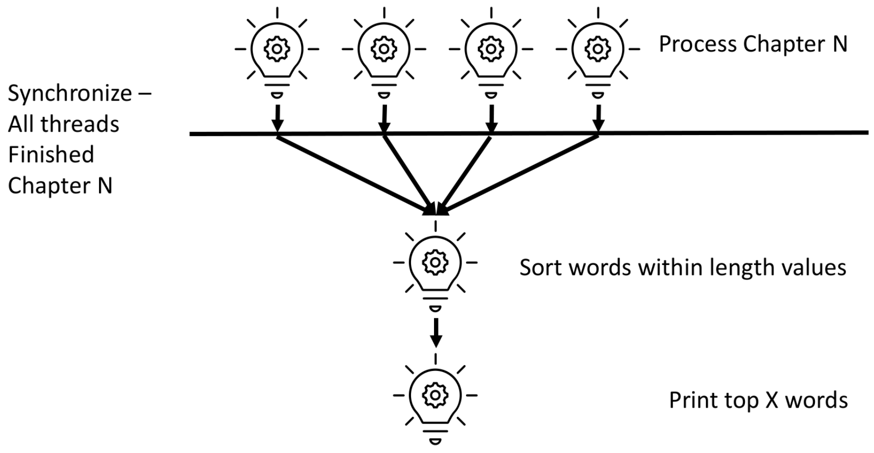
Process Chapter N
Synchronize – All threads Finished Chapter N
Sort words within length values
Print top X words

Figure 2: Output final steps

ready willing and able able able
Hello there consequently I am horsewatcher
Chapter 2
altruistic I I am am words
Chapter 3
consequently arithmetic,

file3:

Chapter 1
ready willing and able
Jumping jumping jacks
Chapter 2
altruistic-like
consequently; arithmetic,
Chapter 3

The tool was instructed to output the 5 longest words. The following will be the output:

```
1 altruistic-like

   1 words length 14

2 consequently
3 consequently;
4 horsewatcher

   3 words length 12

5 altruistic

   1 words length 10
```

Note that “consequently” appears multiple times in the input files, but only once in the output. Also note the alphabetization of the words within each word length - words of length 14 are internally alphabetized, words of length 12 are internally alphabetized, and words of length 10 are internally alphabetized.

There are 3 running total lines printed for the three lengths (14, 12, 10) shown in the output. The numbers at the beginning of each word line are the index of the word in the output.

Since only 5 words were requested, 5 are printed. For length 10 there are actually 3 words, but only the first one alphabetically is printed to reach 5 words total. The other two words of the same length (arithmetic and arithmetic,) are not output. Requesting the 7 longest words would show the others of the same length:

```
1 altruistic-like

   1 words length 14

2 consequently
3 consequently;
4 horsewatcher

   3 words length 12

5 altruistic
6 arithmetic
7 arithmetic,

   3 words length 10
```

7.2 Race Conditions

We will avoid race conditions in the code by using a lock for mutual exclusion. This is essential when adding words to the shared state or searching for existing words.

If you choose to use the C libraries to break a line into words, keep in mind that `strtok` is **not thread safe**. Use `strtok_r`, which is thread safe, instead.

7.3 Sorting Words

Sort the words alphabetically using the `qsort` function in C or some other method. Your sorting must **not** be case sensitive.

8 Barrier Synchronization

The threads will wait for each other at the end of each chapter using a reusable synchronization barrier. A synchronization barrier is an algorithm in which a fixed number of threads must arrive at a condition variable before any one of them can exit. The barrier is configured with the expected number of threads and forces them to wait until that number of threads has reached the barrier. Once reached, all threads can proceed.

Implementing a simple synchronization barrier is not hard, but making a reusable one is tricky. Since we want to enable threads to enter (for instance) for Chapter 2, go on to process Chapter 2, and then wait again for Chapter 3, we must prevent race conditions that might lead a thread getting ahead of others. For example, if a thread finished Chapter 4, we would not want to let it move on to Chapter 5 if the other threads are just getting started on 4.

To do that, we will use code from the textbook that implements a reusable synchronization barrier. The barrier has two stages - one for threads entering the barrier and one for threads exiting the barrier. The division into stages ensures that threads will not be confused about whether they can continue on or not. The code for the barrier is in a function called `checkin` and is found in the given starter code (`checkin.c`). You **should not** need to modify it.

To make the synchronization clear during the output, each thread must output a line when it begins a chapter and when it completes its entire run. The line must be prefixed with the date and time (use `strftime` with the `%c` flag). For instance, consider the following output from processing 3 files, each with 3 chapters:

```
Thu Dec 12 13:41:57 2024 tests/trivial2.txt started chapter 1.
Thu Dec 12 13:41:57 2024 tests/trivial3.txt started chapter 1.
Thu Dec 12 13:41:57 2024 tests/trivial1.txt started chapter 1.
Thu Dec 12 13:41:57 2024 tests/trivial3.txt started chapter 2.
Thu Dec 12 13:41:57 2024 tests/trivial2.txt started chapter 2.
Thu Dec 12 13:41:57 2024 tests/trivial1.txt started chapter 2.
Thu Dec 12 13:41:57 2024 tests/trivial2.txt started chapter 3.
Thu Dec 12 13:41:57 2024 tests/trivial3.txt started chapter 3.
Thu Dec 12 13:41:57 2024 tests/trivial1.txt started chapter 3.
Thu Dec 12 13:41:57 2024 tests/trivial3.txt completed.
Thu Dec 12 13:41:57 2024 tests/trivial2.txt completed.
Thu Dec 12 13:41:57 2024 tests/trivial1.txt completed.
```

9 Signal Handling

To make things more interesting, we'll add a signal handler to the tool as well. Processing longer books may take a while; my tool took over 20 seconds to process four long files on my laptop. The user might want to know how many words have been processed so far without stopping the processing. To support that, add a signal handler that lets the user to press Ctrl+C to receive an update on the tool's progress. The update will show the total number of unique words seen so far in all files.

The update message (*i.e.* signal handler) must output its text to stderr. Be sure to use the correct output parameters when writing so that the regular output goes to stdout but the signal handler's output goes to stderr.

9.1 Output Sample

Let's say we ran the tool on four long files. The tool's run time is 20 seconds. After about 2 seconds, the user presses Ctrl+C (SIGINT) and the tool outputs its current state. The user does this 4 more times, for a total of 5 times. The output to STDERR is as follows:

```
Total unique words so far: 6788
Total unique words so far: 10719
Total unique words so far: 14031
Total unique words so far: 16841
Total unique words so far: 19581
```

At each point, the signal handler outputs the total number of unique words so far. Note that the number of words processed is strictly increasing and represents the total number of words counted by all threads put together.

Despite the repeated Ctrl+C interruptions, the tool completes correctly, finally outputting the word list ordered as above.

10 Input Ouptput Tests (80 points)

The autograder will perform a series of tests on the tool and check its output. The expected outputs for the short tests are given here. All expected outputs (short and long) are included in the template GitHub repository.

In all outputs shown with dates and times, the time and date should be adjusted as appropriate to the actual time.

10.1 No length (3 points)

Tests that the tool outputs a usage message if no length is given.

Input:

```
./longestwords tests/simple-tests
```

Expected output:

```
Usage:  longest-words topCount file1 [...]
```

10.2 No file (3 points)

Tests that the tool outputs a usage message if no file name is given.

Input:

```
./longestwords 10
```

Expected output:

```
Usage:  longest-words topCount file1 [...]
```

10.3 Simple test longest 5 words (5 points)

Tests the tool on a single file asking for the longest 5 words.

```
./longestwords 5 tests/simple-tests
```

Expected output:

```
Thu Dec 12 14:00:54 2024 tests/simple-tests started chapter 1.
Thu Dec 12 14:00:54 2024 tests/simple-tests completed.
1 fifteens
2 fourteen
3 thirteen

   3 words length 8
4 beetles
5 Chapter

   2 words length 7
```

10.4 Simple test longest 10 words (5 points)

Tests the tool on a single file asking for the longest 10 words.

```
./longestwords 10 tests/simple-tests
```

Expected output:

```
Thu Dec 12 14:00:54 2024 tests/simple-tests started chapter 1.
Thu Dec 12 14:00:54 2024 tests/simple-tests completed.
1 fifteens
2 fourteen
3 thirteen

   3 words length 8
4 beetles
5 Chapter
6 cricket
7 dinners
8 invited
9 Serious
10 thither

   7 words length 7
```

10.5 Simple test with second non-existent file longest 10 words (4 points)

Tests that the tool outputs the correct output with a file that does not exist (tests/filenotexist doesn't exist). The output must be identical to the previous test.

```
./longestwords 10 tests/filenotexist tests/simple-tests
```

Expected output:

```
Thu Dec 12 14:00:54 2024 tests/simple-tests started chapter 1.
Thu Dec 12 14:00:54 2024 tests/simple-tests completed.
1 fifteens
2 fourteen
3 thirteen

   3 words length 8
4 beetles
5 Chapter
6 cricket
7 dinners
8 invited
9 Serious
10 thither

   7 words length 7
```

10.6 Simple test longest 15 words (5 points)

Tests the tool on a single file asking for the longest 15 words.

```
./longestwords 15 tests/simple-tests
```

Expected output:

```
Thu Dec 12 14:00:54 2024 tests/simple-tests started chapter 1.
Thu Dec 12 14:00:54 2024 tests/simple-tests completed.
1 fifteens
2 fourteen
3 thirteen

   3 words length 8
4 beetles
5 Chapter
6 cricket
7 dinners
8 invited
9 Serious
10 thither

   7 words length 7
11 "fourth
12 horses
13 second

   3 words length 6
14 "lines
15 "thing"

   2 words length 5
```

10.7 Three short files longest 5 words (6 points)

Tests with 3 short files, retrieving 5 longest words.

```
./longestwords 5 tests/trivial1.txt tests/trivial2.txt tests/trivial3.txt
```

Expected output:

```
Thu Dec 12 14:00:54 2024 tests/trivial2.txt started chapter 1.
Thu Dec 12 14:00:54 2024 tests/trivial3.txt started chapter 1.
Thu Dec 12 14:00:54 2024 tests/trivial1.txt started chapter 1.
Thu Dec 12 14:00:54 2024 tests/trivial3.txt started chapter 2.
Thu Dec 12 14:00:54 2024 tests/trivial2.txt started chapter 2.
Thu Dec 12 14:00:54 2024 tests/trivial1.txt started chapter 2.
Thu Dec 12 14:00:54 2024 tests/trivial2.txt started chapter 3.
Thu Dec 12 14:00:54 2024 tests/trivial3.txt started chapter 3.
Thu Dec 12 14:00:54 2024 tests/trivial1.txt started chapter 3.
Thu Dec 12 14:00:54 2024 tests/trivial3.txt completed.
Thu Dec 12 14:00:54 2024 tests/trivial2.txt completed.
Thu Dec 12 14:00:54 2024 tests/trivial1.txt completed.
1 altruistic-like

   1 words length 14
2 consequently
3 consequently;
4 horsewatcher

   3 words length 12
5 altruistic

   1 words length 10
```

The output of the chapter start messages might be in a different order.

10.8 Three short files longest 10 words (6 points)

Tests with 3 short files, retrieving 10 longest words.

```
./longestwords 10 tests/trivial1.txt tests/trivial2.txt tests/trivial3.txt
```

Expected output:

```
Thu Dec 12 14:00:54 2024 tests/trivial2.txt started chapter 1.
Thu Dec 12 14:00:54 2024 tests/trivial3.txt started chapter 1.
Thu Dec 12 14:00:54 2024 tests/trivial1.txt started chapter 1.
Thu Dec 12 14:00:54 2024 tests/trivial3.txt started chapter 2.
Thu Dec 12 14:00:54 2024 tests/trivial2.txt started chapter 2.
Thu Dec 12 14:00:54 2024 tests/trivial1.txt started chapter 2.
Thu Dec 12 14:00:54 2024 tests/trivial2.txt started chapter 3.
Thu Dec 12 14:00:54 2024 tests/trivial3.txt started chapter 3.
Thu Dec 12 14:00:54 2024 tests/trivial1.txt started chapter 3.
Thu Dec 12 14:00:54 2024 tests/trivial3.txt completed.
Thu Dec 12 14:00:54 2024 tests/trivial2.txt completed.
Thu Dec 12 14:00:54 2024 tests/trivial1.txt completed.
1 altruistic-like

   1 words length 14
2 consequently
3 consequently;
4 horsewatcher
```

```
   3 words length 12
5 altruistic
6 arithmetic
7 arithmetic,

   3 words length 10
8 Chapter
9 Jumping
10 jumping

   3 words length 7
```

The output of the chapter start messages might be in a different order.

10.9 Two long files longest 10 words (7 points)

Test with 2 long files, retrieving 10 longest words.

```
./longestwords 10 tests/moby-dick-sentences.txt tests/pride-sentences.txt
```

10.10 Two long files longest 50 words (7 points)

Test with 2 long files, retrieving 50 longest words.

```
./longestwords 50 tests/moby-dick-sentences.txt tests/pride-sentences.txt
```

10.11 Four long files longest 50 words (8 points)

Test with 4 long files, retrieving 50 longest words.

```
./longestwords 50 tests/moby-dick-sentences.txt tests/pride-sentences.txt
tests/northanger-abbey-sentences.txt tests/frankenstein-sentences.txt
```

10.12 Four/Five long files longest 100 words (7 points)

Test with 4 long files with one file repeated twice (moby-dick-sentences.txt). Retrieving the 100 longest words.

```
./longestwords 100 tests/moby-dick-sentences.txt tests/moby-dick-sentences.txt
tests/pride-sentences.txt tests/northanger-abbey-sentences.txt
tests/frankenstein-sentences.txt
```

10.13 Two long files longest 100 words with 3 Ctrl+C Signals (7 points)

Test with 3 long files, retrieving the 100 longest words. While running, the test sends 3 Ctrl+C messages.

```
./longestwords 100 tests/moby-dick-sentences.txt tests/pride-sentences.txt
```

10.14 Four long files longest 150 words with 5 Ctrl+C Signals (7 points)

Test with 4 long files, retrieving 150 longest words. While running, the test sends 5 Ctrl+C messages.

```
./longestwords 150 tests/moby-dick-sentences.txt tests/pride-sentences.txt
tests/northanger-abbey-sentences.txt tests/frankenstein-sentences.txt
```

11 Using Git and GitHub (5 points)

Submit all of your code for the homework using the private repository for your work on GitHub. You must perform the following actions on the repository:

1. Each team member must make at least 1 substantive commit of code or text to the repository.
2. The team must make **at least** two (2) commits to the repository and add a comment to each.
3. The last commit must include the comment "Submitted for grading"
4. The repository must include a README.md file with the following details:
   - Student names and IDs
   - Assignment name (Assignment 4)
   - Course name (Operating Systems)
   - Semester and Year
   - Number of hours spent on the assignment total.

Repositories missing any of the above actions will be penalized.