Assignment 4: Multi-threaded Producer Consumer Pipeline

Start Assignment

Due Monday by 11:59pm **Points** 105 **Submitting** a file upload **Available** Feb 7 at 8am - Feb 24 at 11:59pm 18 days

Introduction

In this assignment, you'll write a program that will get you familiar with the use of threads, mutual exclusion and synchronization variables.

Learning Outcomes

- Describe what is mutual exclusion and why is it an important property to maintain when developing programs that may concurrently access shared resources (MLO 2)
- Describe the API you can use to create threads and wait for the termination of a thread (MLO 4)
- Describe what are synchronization variables and the API related to their use (MLO 4)

Instructions

Write a program that creates 4 threads to process input from standard input as follows

1. Thread 1, called the Input Thread, reads in lines of characters from the standard input.
2. Thread 2, called the Line Separator Thread, replaces every line separator in the input by a space.
3. Thread, 3 called the Plus Sign thread, replaces every pair of plus signs, i.e., "++", by a "^".
4. Thread 4, called the Output Thread, write this processed data to standard output as lines of exactly 80 characters.

Furthermore, in your program these 4 threads must communicate with each other using the Producer-Consumer approach.

Input

- A "line of input" is
  - A sequence of the allowed characters (see below) that does not include a line separator
  - Followed by a line separator.
- Line separator
  - Since the program is going to be tested on os1, and hence Unix, for the purposes of this assignment line separator is the newline character, i.e., '\n'
  - Note that line separator may not be the newline character on Windows or Mac, but that's irrelevant for this assignment.
- Allowed characters
  - Other than the line separator, the input will only consist of ASCII characters from space (decimal 32) to tilde (decimal 126). These are sometimes termed printable characters.
- Stop-processing line
  - Your program must process input lines until it receives an input line that contains only the characters STOP, i.e., STOP followed immediately by the line separator.
  - Examples: The following input lines must not cause the program to stop processing the input
    - STOP!
    - ISTOP
    - stop
    - Stop
- If there are any more lines in the input after the stop-processing line, the program must not process them.
- The input will not contain any empty lines, i.e., lines that only have space characters or no characters except the line separator.
- An input line will never be longer than 1000 characters (including the line separator).
- The input for the program will never have more than 49 lines before the stop-processing line.
- Your program doesn't need to check the input for correctness.

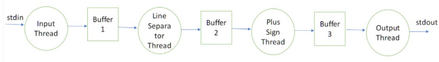
Output

- The "80 character line" to be written to standard output is defined as 80 non-line separator characters plus a line separator.
- **Your program must not wait to produce the output only when the stop-processing line is received**.
  - Whenever your program has sufficient data for an output line, the output line must be produced.
- After the program receives the stop-processing line and before it terminates, the program must produce all 80 character lines it can still produce based on the input lines which were received before the stop-processing line and which have not yet been processed to produce output.
- No part of the stop-processing line must be written to the output.
- In addition, your program must not output any user prompts, debugging information, status messages, etc.
- Your program must output only lines with 80 characters (with a line separator after each line).
- For the second replacement requirement, pairs of plus signs must be replaced as they are seen.
  - Examples:
    - The string "abc++def" contains only one pair of plus signs and must be converted to the string "abc^def".
    - The string "abc+++def" contains two pairs of plus signs and must be converted to the string "abc^^def".
  - Thus, whenever the Plus Sign thread replaces a pair of plus signs by a caret, the number of characters produced by the Plus Sign thread decreases by one compared to the number characters consumed by this thread.

Example

1. We start the program, type 10 characters (not containing any ++) and then press enter. These characters should not be printed to standard output right away because we have 11 characters available to write (10 characters we typed and a space that replaced the line separator) and don't yet have the 80 characters needed to write one complete line.
2. Next we type 170 characters (not containing any ++) and then press enter. Now there are 182 characters available to write, 11 characters from the Step 1, 170 characters that we typed in step 2 and the space that replaced the line separator in step 2.
3. The program must write 2 lines with 80 characters each. There are still 22 characters available for output.
4. Next we type the stop-processing line. Since we only write complete lines with 80 characters, although there are 22 characters still available to write, the program terminates without writing these characters to the standard output.

Multi-Threading Requirements



Pipeline of threads that gets data from stdin, processes it and displays it to stdout

- Each pair of communicating threads must be constructed as a producer/consumer system.
- If a thread T1 gets its input data from another thread T0, and T1 outputs data for use by another thread T2, then
  - T1 acts as a consumer with respect to T0 and T0 plays the role of T1's producer
  - T1 acts as a producer with respect to T2 and T2 plays the role of T1's consumer
- Thus each thread in the interior of the pipeline (i.e., the Line Separator and Plus Sign threads) will contain both producer code and consumer code.
- Each producer/consumer pair of threads will have its own shared buffer. Thus, there will be 3 of these buffers in your program, each one shared only by its producer and consumer.
- You must use synchronization variables for coordination.
- Your program must never sleep.
- If you size your buffers to hold 50 lines of 1000 characters each, you can model the problem as Producer-Consumer with unbounded buffers which will make your program simpler.
  - Recall that unbounded buffers are never full, though they can be empty.

Hints & Resources

- Consider creating a single threaded implementation of the required processing to get the processing logic ironed out.
  - In the single-threaded version, put the processing that will eventually be done by different threads in separate functions. Then you can use those functions in your multi-threaded program.
  - When you create a multi-threaded implementation of your program, for the same input, the output of both implementations of the programs must be the same.
- For the multi-threaded implementation:
  - We have provided an example program that implements a pipeline of producers and consumers consisting of 3 threads.
  - Feel free to adapt this program for your implementation. As always, you are also allowed to use any code presented in the course explorations. Cite the use of any code from the course.

What to turn in?

- You can only use C for coding this assignment and you must use the gcc compiler.
- You can use C99 or GNU99 standard or the default standard used by the gcc installation on os1.
- Your assignment will be graded on os1.
- Submit a single zip file with all your code, which can be in as many different files as you want.
- This zip file must be named `youronid_program4.zip` where youronid must be replaced by your own ONID.
  - E.g., if chaudhro was submitting the assignment, the file must be named `chaudhro_program4.zip`.
- In the zip file, you must include a text file called `README.txt` that contains instructions on how to compile your code using gcc to create an executable file that must be named `line_processor`.
- When you resubmit a file in Canvas, Canvas can attach a suffix to the file, e.g., the file name may become `chaudhro_program4-1.zip`. Don't worry about this name change as no points will be deducted because of this.

Grading Criteria

- This assignment is worth 12% of your final grade.
- The points for the assignment and the break-up for items is described in the grading rubric.

How Grading Will Be Done?

The graders will use both files and the keyboard to test your program.

Input

The program must read from standard input, which means the input can come from the keyboard or from a file.

- When the program is started without redirecting stdin, the input will come from the keyboard. E.g., when the program is started as follows

```
./line_processor
```

- When the program is started using input redirection, the input will come from a file. E.g., when the program is started as in the following command

```
./line_processor < input1.txt
```

Whether the input comes from the keyboard or from a file is transparent to the program which just needs to always read from stdin.

Output

The program must write to stdout, which means the output can go to the terminal or to a file.

- When the program is started without redirecting stdout, the output will go to the terminal. E.g., when using the following command

```
./line_processor
```

- When the program is started with stdout redirected, the output will go to a file. E.g., the output will go to the file output1.txt when using the following command

```
./line_processor > output1.txt
```

Whether the output goes to the terminal or to a file is transparent to the program which just writes to stdout.

Note the program can also be started with both input and output redirection. E.g., in the following example the input comes from input1.txt and the output goes to output1.txt

```
./line_processor < input1.txt > output1.txt
```

Again this is transparent to the program.

Grading Items in the Rubric

Input with file redirection can test the following items in the rubric:

- Line separators are correctly replaced by spaces.
- Pair of plus signs are correctly replaced by the caret symbol.
- Output lines are always 80 character long.
- The program does not write anything to the standard output other than the complete lines it can write before receiving the stop-processing line.
- The program terminates after writing all the 80 character lines it can write.

Input from the keyboard will be used to test the following item in the rubric

- A line of output is written as soon as 80 characters are available to output.

The remaining items in the rubric require reviewing the code.

Sample Input and Output Files

Here are some sample input files and corresponding output files that should be produced by these input files.

- input1.txt
- input2.txt
- input3.txt
- output1.txt
- output2.txt
- output3.txt

Note: If you do see ^M characters all over your files, which come from copying a Windows-saved file onto a Unix file system, try this command:

```
$ dos2unix bustedFile
```

Assignment 4

Criteria	Ratings		Pts
Line separators are correctly replaced by spaces.	4 pts Full Marks	0 pts No Marks	4 pts
Pair of plus signs are correctly replaced by the caret symbol.	4 pts Full Marks	0 pts No Marks	4 pts
Output lines are always 80 character long.	4 pts Full Marks	0 pts No Marks	4 pts
The program does not write anything to the standard output other than the complete lines it can write before receiving the stop-processing line.	4 pts Full Marks	0 pts No Marks	4 pts
The program terminates after writing all the 80 character lines it can write.	4 pts Full Marks	0 pts No Marks	4 pts
A line of output is written as soon as 80 characters are available to output.	15 pts Full Marks	0 pts No Marks	15 pts
Input and Line Separator threads are created and interact correctly as Producer-Consumer.	30 pts Full Marks	0 pts No Marks	30 pts
Plus Sign thread is created and interacts correctly as a consumer with Line Separator thread as its producer in the pipeline.	20 pts Full Marks	0 pts No Marks	20 pts
Output thread is created and interacts correctly as a consumer with the Plus Sign thread as its producer in the pipeline.	15 pts Full Marks	0 pts No Marks	15 pts
The code is well commented.	5 to >0.0 pts Full Marks	0 pts No Marks	5 pts
		Total Points: 105	