COMP 530 Introduction to Operating Systems

Fall 2017  
Kevin Jeffay

Worksheet 12, October 11

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Your Name: |  | You worked with: |  | +1/blank/-1: |  |
|  | Aaron Zhang |  | John Espenhahn |  | +1 |  |
|  |  |  | Brennan Proudfoot |  | +1 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Consider a pipeline of ST threads operating as a sequence of producers and consumers communicating with a bounded buffer between each producer/consumer pair as illustrated below and as you likely implemented in HW3.



In HW3 you hopefully observed that the behavior of the processing pipeline may be complex and may depend on a number of factors other than the logic in the processing code. For this worksheet you will consider exactly how these threads interact and how various output scenarios are possible as a function of various design choices for how the pipeline is implemented. Note that these design choices considered here go beyond what was required (and what you likely implemented) for HW3.

In answering the following questions you are to assume that:

1. The code for each producer portion of a thread is written so that the thread only produces one character at a time, and the code for each consumer portion of a thread is written so that the thread only consumes one character at a time, and

ST threads are user threads. If one of the threads needs to wait on a system queue(not a queue controlled by ST, aka when waiting on a condition variable), all processes will get blocked. This is because the system doesn’t have an understanding of the threads within a process.

When a thread waits on a ST queue

1. The output thread (the last thread in the processing pipeline) is designed to produce exactly one line of output for each character it receives (consumes). That is, unlikeHW3, the output behavior of this pipeline is to output a series of output lines each consisting of a single character and newline.

The following lists three different scenarios for programming and operating the processing pipeline. In each scenario, a particular set of outputs are generated and the program ends up in a particular state. For each scenario, explain why the behavior described results. To do this you will need to make reasonable assumptions about how ST threads are scheduled and how semaphores are implemented. Stated another way, use the descriptions of the behavior of the thread pipeline that follow to infer how threads are likely scheduled and how semaphores are likely implemented.

1) A line of input (a sequence of characters plus a newline) is provided as input to the program by the user. The user then pauses before entering more input. While the user pauses, she observes that fewer output lines are produced by the output thread than the number of characters that were entered on the input line. In addition, the number of characters on the missing lines (the expected lines of output that have not yet been output) is less than the total character capacity of all the bounded buffers used in the processing pipeline.

Explain how the pipeline threads have executed such that the observed behavior occurred.

2) In *ST*, the function **st\_sleep(x)** suspends the calling thread for *x* seconds. Assume the statement **st\_sleep(1)** is added to the code for the input thread so that it is executed at the end of each iteration of the main loop in the input thread. When this code is added and the user enters a single line of input and pauses before entering additional text, exactly one line of output is produced for each character entered on the input line.

Explain what has happened in this scenario. In particular, explain why has requiring the input thread to “sleep” so changed the execution behavior of the pipeline?

3) The **st\_sleep(1)** statement is removed from the input thread and added to the output thread so that it is executed at the end of each iteration of the processing loop in the output thread.

Assuming that the user enters a very long line of input (relative to the numbers buffers in the pipeline), explain what will happen in this scenario.

all of the buffers will have to fill up before any characters can be output. The output thread then outputs a single character (and newline) and puts itself to sleep. This sleep time is enough for all of the buffers to fill up again, putting the input and processing threads in the waiting state waiting for a hole to appear that they can push characters into. Because the input is much larger than the size of the buffers, this repeats several times, with the output thread outputting 1 character per second and the rest of the threads catching up while output sleeps.

Eventually, the input runs out. As input goes to fill one of the holes created by output, its input runs out and the whole process blocks again on getline().

When the process blocks on getline(), all of the buffers are full, and (total number of characters - total size of buffers) characters have been output, 1 per second.