HW1: Standalone I/O package for the Tutor VM

1. Introduction

For each homework in this class, students should make a subdirectory of the same name, e.g. hw1_part1, hw1_part2, and put everything there you want the grader to look at. Copy files from the course homework directory to your own. Supply a README file for the grader to read first, and include in it your authorship statement, names of team members and any special notes. The style of the program is important. See C coding guidelines.

2. Running the provided testio.lnx using remote gdb (Part 1)

You can find a good summary of gdb commands in http://www.cs.umb.edu/~cheungr/cs341/gdb-refcard.pdf. To set up remote gdb for the VM environment, please use instructions provided in Section 6 of VMWare-for-Tutor_PC_2021.pdf (or VMWare-for-Tutor_MAC_2021.pdf).

All source files needed for this homework assignment can be found in /courses/cs444/f21/cheungr/hw1. Copy these files over to your hw1_part1 directory. Build the executable by running make. Study testio_mod-orig.script and reproduce it, noting the behavior that needs to be fixed by your work for hw1 Part 2.

To include the debug_log output, please do:

```
md 300000
<CR>
<CR>
<CR>
<CR>
```

at the end of your run. Save it under testio_mod-username.script.

Then reproduce remgdb-testio_mod.script, that is, run the provided testio.lnx under control of remote gdb with several breakpoints and "next" commands, and successfully enter input in the console window. Additionally, display the contents of buf after it has been filled in. If you are using remote gdb, use the command:

```
x/16bx 0x300000
……
x/16bx 0x300010
……
```

to display 16 memory locations starting at address 0x300000 and starting at 0x300010. Leave this in hw1_part1/remgdb-testio_mod-username.script. This part is due before Part 2 **.**

3. A standalone I/O package for the tutor VM (Part 2)

By "standalone" we mean there is one and only one program running on the machine, so the whole machine is owned by that program.

We start from an almost-complete but imperfectly coded device-independent standalone I/O package provided in /courses/cs444/f21/cheungr/hw1. Copy these files over to your hw1_part2. Modify these files as follows:

a. We provide a queue module in subdir hw1/queue that provide the following functions:

```
init_queue(Queue *q,  int max_chars);  /*make a new empty queue, filling in *q */
int enqueue(Queue *q,char ch);          /* enqueue char ch on q, or return -1 if can't */
int dequeue(Queue *q);                  /*  dequeue a char and return it, or return -1 if can't */
int queuecount(Queue *q);               /*  return # chars currently in queue q */
```

Use it to replace all the ring-buffer code in this I/O package with proper queues, one Queue for the readqueue, another for the writequeue, etc. Use just 6-char Queues so you are sure to test it fully. Don't edit or replace queue.c, just *use* it.

The code should now be easier to understand. Note: we don't have malloc in the tutor library. Therefore you will need to use "Queue" type variables to hold the queue state, not Queue * pointers + malloc'd space as often done in cs310. Once you have a variable of type Queue, it's easy to get a Queue pointer by using &.

b. Type ahead is a feature that enables users to continue typing regardless of the computer operation. If the receiver is busy at the time, it will be called to handle this later without dropping characters. The original read function takes in only the characters before the read function is called. A longer delay loop before the read function is called will cause more characters to be read. We want to fix the read function such that it will read in all the characters the user types ahead.

c. The write function only polls--let's make it interrupt driven too. Set up an output queue for each port, again only 6 chars capacity. Chars from a user write call are enqueued, or, if the queue is or becomes full, the caller has to wait for space to show up. Set up a second queue for echoes, and output preferentially from the echo queue.

Note: because this is a standalone I/O package, and not a real OS, it is allowable and necessary for the waits to be busy loops. Once we are writing OSs, this is not a good idea to waste CPU time that could be used by another program.

4. Testing

Testing is always an important part of any code writing. Generally, you need to write many tests. Here, I've provided a comprehensive test for you in testio.c. Your hw1 submission should include the output of a test run. Use the 'script' command to capture the output of your test runs in the VM and then file transfer them to your course directories (/courses/cs444/f21/your_username/hw1_part1/ and /courses/cs444/f21/your_username/hw1_part2/) in the umb server for grading.
The following files need to be in your cs444/hw1_part1 directory:

i) README: with authorship statement; names of team members
ii) tty.c, tty.h,, syms and testio.lnx
iii) remgdb-testio_mod-username.script-include debug log outputs

iv) testio_mod-username.script- include debug_log outputs

The following files need to be in your cs444/hw1_part2 directory:

i) README: include any operational instructions; names of team members
ii) tty.c, tty.h,, syms and testio.lnx
iii) testio_mod-username.script- include debug_log outputs