Programming Project Checkpoint 4

[112006234] [Justin Thiadi 程煒財]

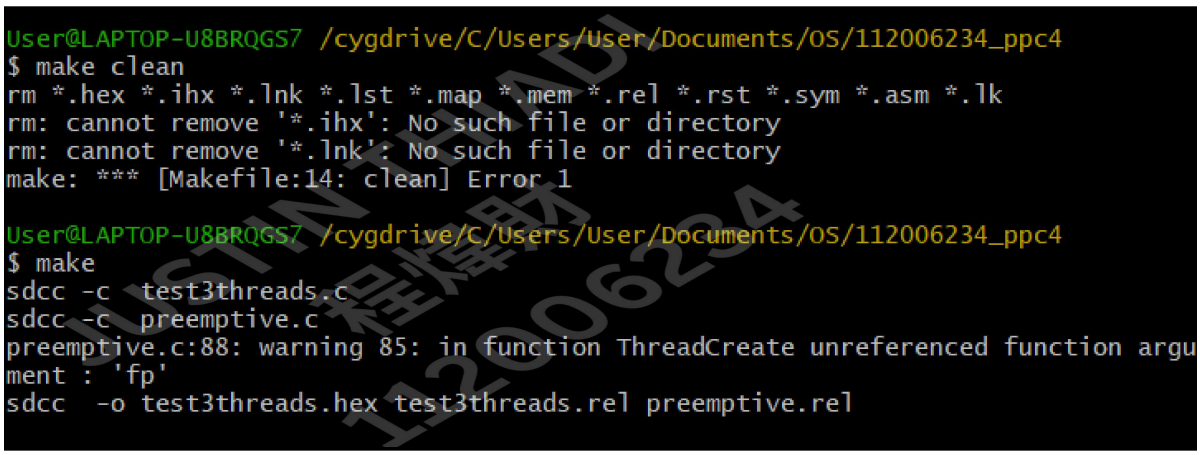
Task 1:

Turn in Screenshots showing compilation of your code using a modified Makefile (same as for cooperative except the file names are changed to the preemptive version). You should use the following two commands (Note: $ is the prompt displayed by the shell and is not part of the command that you type.) The first one deletes all the compiled files so it forces a rebuild if you have compiled before. The second one compiles it.

```
$ make clean
$ make
```

It should show actual compilation, warning, or error messages. Note that not all warnings are errors. The compiler should generate several `testpreempt.*` files with different extensions:

- the `.hex` file can be opened directly in EdSim51
- the `.map` file shows the mapping of the symbols to their addresses after linking



Task 2:

- Answering (1) Try different orders of spawning threads. What do you get?
  - Changing the order of ThreadCreate() calls changes **which thread gets which ID**, affecting the **scheduling sequence**.
  - Changing creation order changes **which producer runs first** after context switches
    - When we call ThreadCreate(Producer2) first, then the output will be 012ABC345DEF…

Order of ThreadCreate Calls	Thread ID 0	Thread ID 1	Thread ID 2	Thread ID 3
Producer1, Producer2, Consumer	Consumer	Producer1	Producer2	unused
Producer2, Producer1, Consumer	Consumer	Producer2	Producer1	

HOWEVER, creating the Consumer thread before producers can cause:

- **Premature** system **exit** (if Consumer completes before producers start)
- **Deadlock** situations (if Consumer waits for data that never arrives)

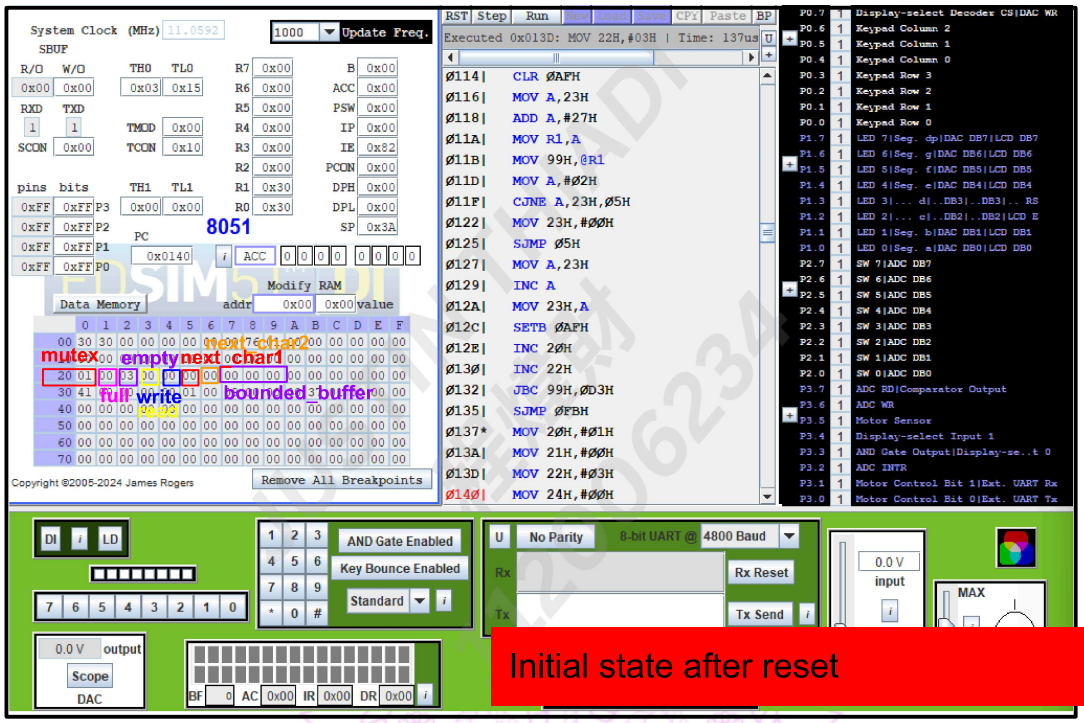
So, always create all threads before starting the main logic in thread 0.

Look up the addresses for your symbols (i.e., functions, variables, etc) in the file testpreempt.map. Set one or more breakpoints in EdSim51's assembly code window after you have assembled it.

```
 Value  Global                          Global Defined In Module
 -----  -------------------------------  ------------------------
00000081  _Producer1                      test3threads
000000BF  _Producer2                      test3threads
000000FD  _Consumer                       test3threads
00000137  _main                           test3threads
00000158  __sdcc_gsinit_startup           test3threads
0000015C  __mcs51_genRAMCLEAR             test3threads
0000015D  __mcs51_genXINIT                test3threads
0000015E  __mcs51_genXRAMCLEAR            test3threads
0000015F  _timer0_ISR                     test3threads
00000165  _Bootstrap                      preemptive
0000018B  _ThreadCreate                   preemptive
00000211  _ThreadYield                    preemptive
0000026B  _ThreadExit                     preemptive
000002CA  _myTimer0Handler                preemptive
```

```
 Value  Global                          Global Defined In Module
 -----  -------------------------------  ------------------------
00000000  .__.ABS.
00000020  _mutex                          test3threads
00000021  _full                           test3threads
00000022  _empty                          test3threads
00000023  _read                           test3threads
00000024  _write                          test3threads
00000025  _next_char1                     test3threads
00000026  _next_char2                     test3threads
00000027  _bounded_buffer                 test3threads
00000030  _saved_SP                       preemptive
00000034  _current_thread                 preemptive
00000035  _valid_thread                   preemptive
00000036  _new_thread                     preemptive
00000037  _old_SP                         preemptive
00000038  _dir                           preemptive
```

- Take screenshots when the Producer is running and show semaphore changes.

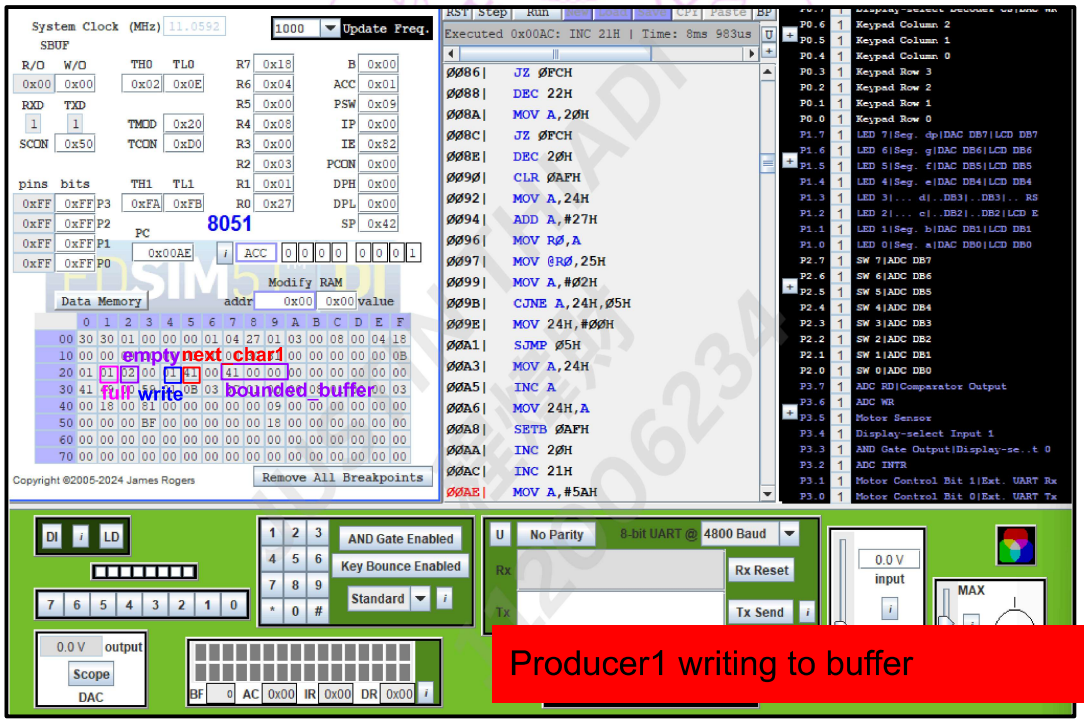


All semaphores initialized correctly

mutex (0x20) = 1 **full** (0x21) = 0

empty (0x22) = 3 read (0x23) = 0

write (0x24) = 0 bounded_buffer (0x27 to 0x29) = 0

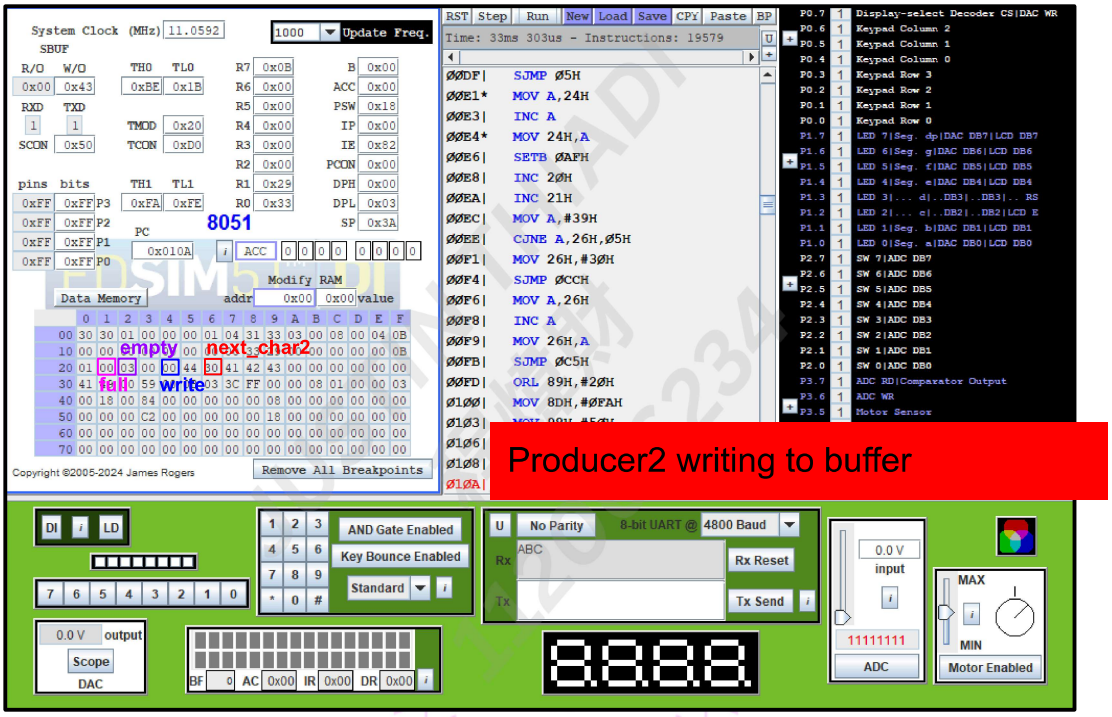


At this point, we:

- Step into Producer1
- Stop just after bounded_buffer[write] is updated

Observation:

- 'A' (0x41) at 0x27 (bounded_buffer[0])
- write incremented (from 0 to 1)
- **empty** decremented (from 3 to 2)
- **full** incremented (from 0 to 1)

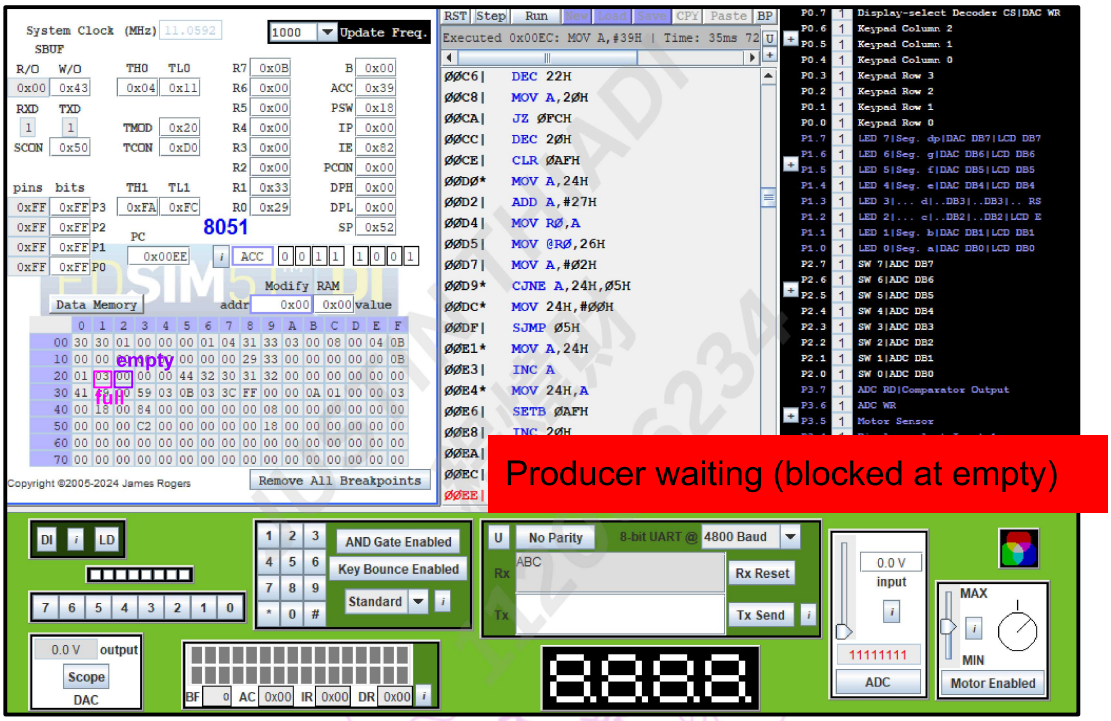


At this point, we:

- Step into Producer2 (after a context switch).
- Stop after it writes '0' into the buffer.

Observation:

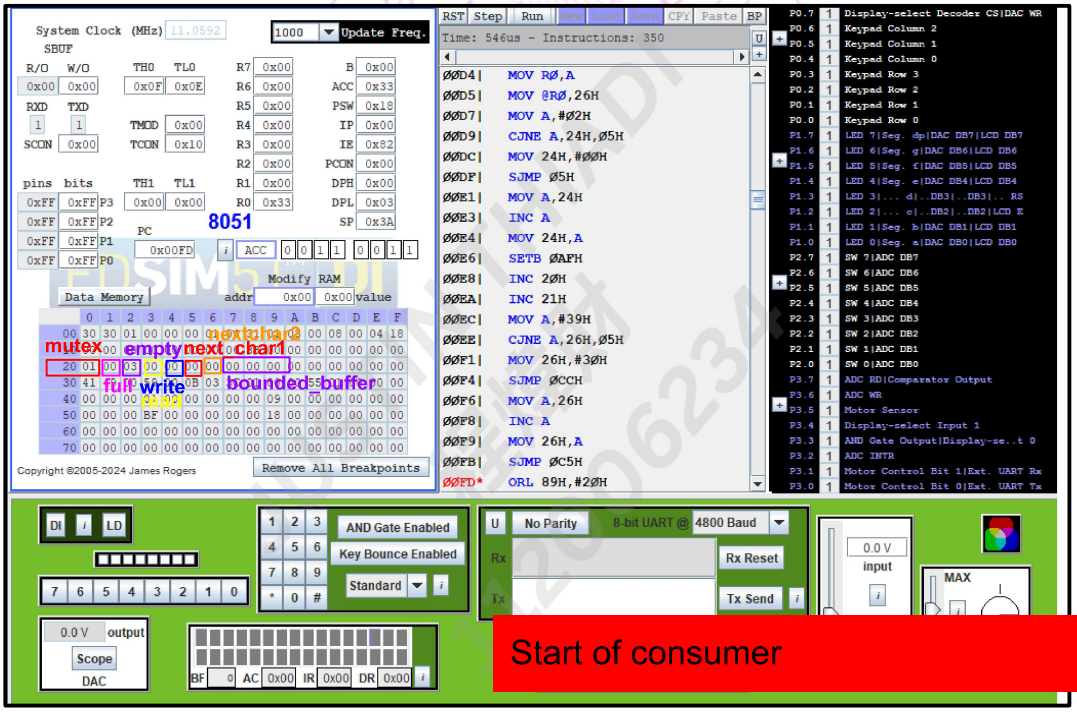
- '0' (0x30) at next buffer slot
- write incremented again (from 1 to 0) - since if (write == 2) write = 0; - refer to code
- empty and full updated (**full to 0**, **empty to 3**)



When letting both producers run until the buffer is full (empty = 0).

- bounded_buffer has 3 characters
- **empty** = 0
- Producer1 or Producer2 is stuck in SemaphoreWait(empty)

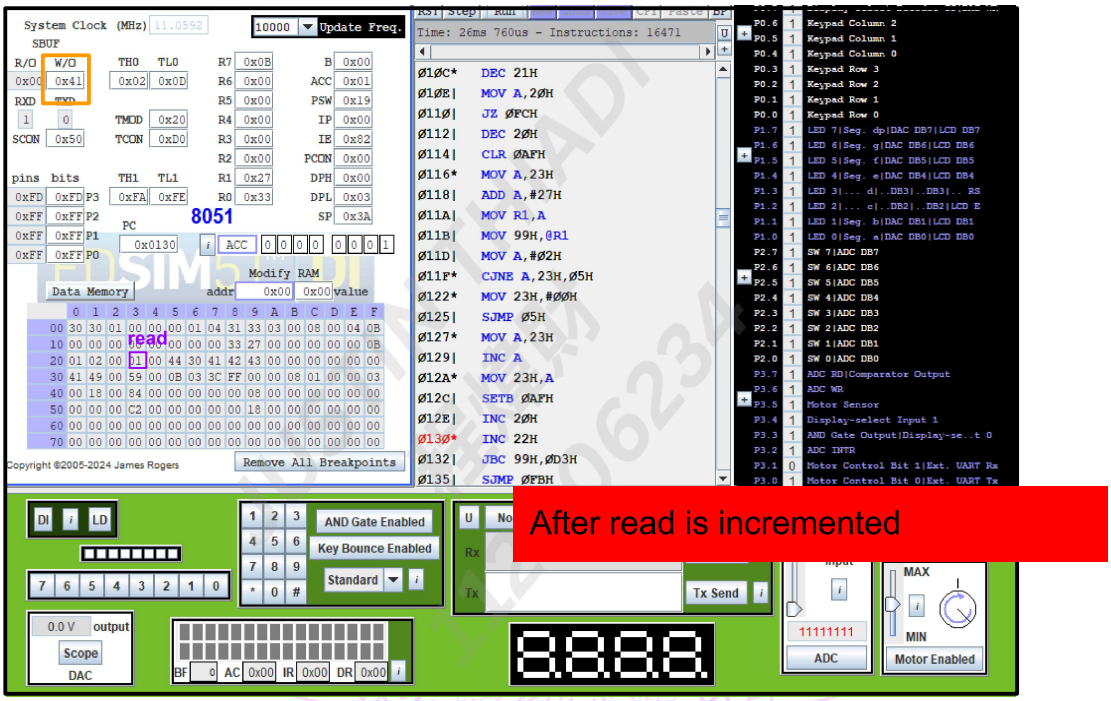
- Take screenshots when the Consumer is running and show semaphore changes



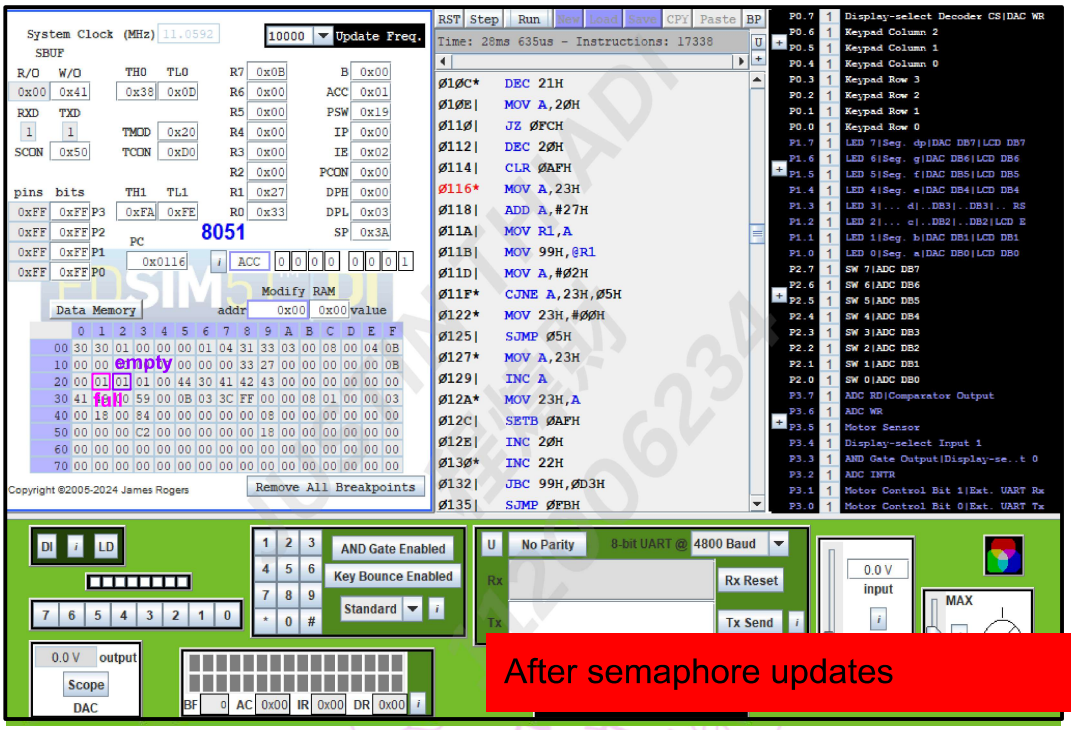
- mutex = 1
- full = 0
- empty = 3

Consumer running the buffer:

- SBUF gets a value (at this point, 'A')

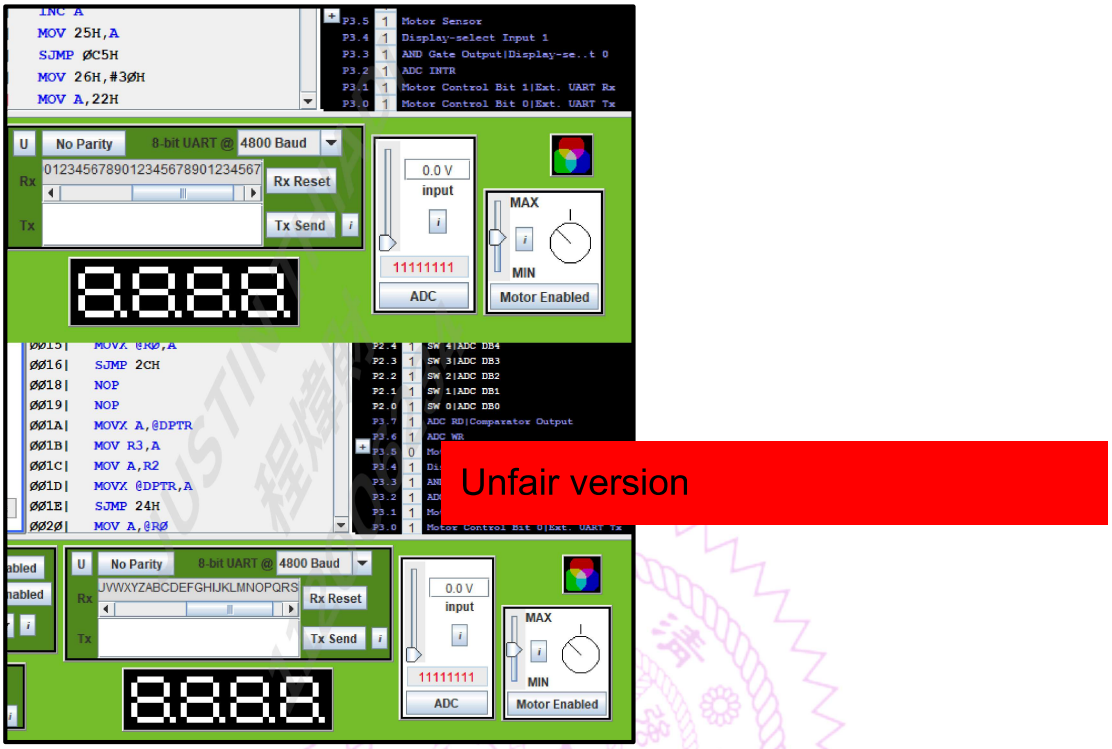


This shows that read is being updated correctly (read++, becomes 1 from 0)



- **full**– (from 2 to 1)
- **empty**++(from 0 to 1)
- shows the consumer is updating the semaphores properly

- Show and explain UART output to show the unfair version, if any, and the fair version.



If only one kind of character (letters only, or digits only) fills the UART, it shows unfairness. In the UART log (can be seen from screenshot above), we observe that only Producer1's characters (A, B, C...) are being consumed. This indicates Producer2 is **not getting scheduled fairly**. The system is therefore showing **unfair behavior**.

The *Timer0Handler* in the code is fair. It performs **round-robin scheduling while alternating direction** using *dir*, switching between forward and backward traversal of thread IDs. This **prevents starvation** and ensures all valid threads get CPU time evenly over multiple cycles. It also skips over invalid (exited) threads, maintaining fairness among active ones.