CS 342302 Operating Systems
Fall Semester 2024
Prof. Pai H. Chou

Programming Project Checkpoint 4

What to submit: One zip file named <studentID>-`ppc4.zip` (replace <studentID> with your own student ID). It should contain:

- one PDF file named **ppc4.pdf** for Part 4 of this checkpoint. It should explain how your code works. Write your answers in **English**. Check your spelling and grammar. Include your name and student ID.
- (Part 1) Turn in the source files for preemptive multithreading to be compiled using SDCC and targets EdSim51.
    - `test3threads.c`, which contains the startup code and sets up the 2-producers, 1-consumer example.
- (Part 2) Turn in the screenshot for compiling your code using Makefile from CheckPoint3.
    - File named `ppc4.pdf` that includes the screenshots and explanations as instructed below.

For this programming project checkpoint, you are to test your preemptive multithreading and semaphore code by extending the classical bounded-buffer example to two producers and one consumer. The two producers will compete with each other for writing.

You should make a new directory and copy the source files from the previous checkpoint. Make a copy of `testpreempt.c` and name it `test3threads.c`. This is your opportunity to stress test your semaphore and threads code. If your previous checkpoints worked but this one doesn't, then something is wrong with your code. Debug here and once it works, then apply those changes to your previous checkpoints to see if they still work.

1. [40 points] `test3threads.c`

Your `test3threads.c` can be based on `testpreempt.c` for CheckPoint3. Continue the use of the 3-deep bounded-buffer (3-deep) structure.

- Instead of `Producer()` function, make `Producer1()` and `Producer2()` functions. `Producer1` should output letters `'A'` to `'Z'` and repeat forever; `Producer2` should output `'0'` to `'9'` and repeat forever.
- `Consumer()` can remain unchanged.
- `main()` should spawn threads so that `Producer1`, `Producer2`, and `Consumer` run in three threads.

Try different orders of spawning threads. What do you get?

2. [40 points] Fairness

How “fair” is your scheduler based on the output that you observe? If you are using the default round-robin (RR) scheduling policy, chances are you will see the output of only one producer; and if you change the order of spawning threads, you may see the output of the other producer. What kinds of output patterns do you consider to be fair? Does your code have the problem of starvation?

Either

- show that your scheduling policy is “fair” and explain why, or
- propose your own solution to make it fair for these threads. Explain the changes that you made.

3. [20 points] Screenshots

3.1 [2 points] Screenshot for compilation

Turn in screenshots showing compilation of your code using the `Makefile` from Checkpoint3 (but extended for `test3threads.c`). 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 `test3threads.*` 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

3.2 [18 points] Screenshots and explanation

- Take screenshots when the `Producer1` and `Producer2` running and show semaphore changes.
- Take screenshots when the `Consumer` is running and show semaphore changes.
- Show and explain UART output to show the unfair version, if any, and the fair version.