CS 342302 Operating Systems
Fall Semester 2022
Prof. Pai H. Chou

Programming Project Checkpoint 5[1]

Due Date:Friday, January 13 , 2023, 11:59pm
Submit electronically on eLearn

What to submit: One zip file named <studentID>-`ppc5.zip` (replace <studentID> with your own student ID). It should contain:

- one PDF file named **ppc5.pdf** for Part 5 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 the function to be compiled using SDCC and targets EdSim51.
    - `preemptive.c`, which
        - contains the new code for `delay(`*n*`)` function, where *n* is the number of time units to delay.
        - in case you have not done so already, `ThreadCreate()`, `ThreadExit()` need to allow a thread to be recycled after the function returns normally or after it calls `ThreadExit()` function.
    - `testparking.c`, which contains the parking lot example to test your updated threads package.
    - `preemptive.h`, which is also a required file to compile the project
- (Part 2) Turn in the typescript for compiling your code using the updated `Makefile`.
    - `typescript`,
    - File named `ppc5.pdf` that includes the screenshots and explanations as instructed below.

For this programming project checkpoint, you are to (1) add a `delay(`*n*`)` function to your preemptive multithreading and semaphore code, (2) ensure your threads can terminate so its space can be recycled by another thread. (3) test your delay and thread recycling based on (but not identical) to the parking lot example you did earlier in the semester in Python.

There will be less hints for this CheckPoint.

---

[1] A project programming checkpoint is a self-assessed assignment that does not count towards your assignment grade. You are not required to turn in anything; however, you are responsible for understanding the concepts for the subsequent lecture materials, assignments, and exams. Therefore, you are strongly urged to do it as a regular assignment and "grade yourself" according to the breakdown. You should be able to complete this entire checkpoint in one week, since plenty of hints are given; if not, you should ask for help and keep up, or else you will have a difficult time with the project. You will receive one score at the end of the semester for the programming project part of your grade.

1. [50 points] `delay(n)`, `now()`

You are to implement the `void delay(unsigned char n)` and `unsigned char now(void)` functions. The former delays the thread by *n* time units; the latter returns the "current time" (i.e., number of time units that has elapsed since the system reboots). There are a number of considerations:

- In Python, the time unit for `Thread.sleep()` is in seconds, but it can be a floating point. On 8051, we limit the time unit to be an unsigned char (0-255), but what is a good time unit? (e.g., number of seconds, ms, μs, or some multiple of the 8051 timer?) Note that any time unit that is larger than 1 second is not useful.
- An important consideration is that `delay()` is not an exact delay but is a delay for "at least *n* time units" and "less than (*n* + 0.5) time units" for it to be acceptable (otherwise, it rounds up to *n*+1 time units, which would not be correct). Of course, the more accurate the better, but there is an inherent limit on how accurate it can be.
- Based on the above requirement, state your choice of time unit and provide your justification for how you think you can implement a `delay()` that meets the requirement above.

Explain how you would implement the delay function in relation to the timer used for preemption. By default, we set it up for a 13-bit timer for the quantum. Considering that each thread gets its own delay call independently:

- what does your timer-0 ISR have to do to support these multiple delays and now()?
- what if all threads call delay() and happen to finish their delays all at the same time? How can you ensure the accuracy of your delay? (i.e., between *n* and *n*+0.5 time units)?
- How does the worst-case delay completion (i.e., all threads finish delaying at the same time) affect your choice of time unit?

You may assume it is okay for `now()` to wrap around to 0 after it exceeds 255 time units.

Create your own test case to be sure your delay function works before proceeding to the next part.

2. [50 points] Robust Thread Termination and Creation

You may have already implemented the thread termination already, but if you have not, you should modify your code so that a thread can safely terminate by either calling `ThreadExit()` or return normally from the function. This is easy, because all you have to do is to push the return address of `ThreadExit()` on the stack. `ThreadExit()` just has to mark the thread as unallocated.

One other thing you need to do is to guard thread creation and termination. We have a maximum limit of 4 threads. This means you need to use a semaphore to allow creation of threads up to the max, and any attempt to create additional threads will block until some

thread has exited. Add the proper code in your `ThreadCreate()` and `ThreadExit()` code to make use of the semaphore(s) similar to the bounded-buffer example.

One thing about `ThreadExit()` is that if you are the last thread to exit, then it should enter an infinite loop, instead of returning to nowhere.

3. [50 points] Parking Lot Example

Consider the parking simulation example you in Python for Assignment 8, Part 2. Instead of making 15 cars and 5 spots, make 5 cars competing for 2 spots. Make thread for each car. However, since our threads package supports at most 4 threads (including main), your main() will use a semaphore, similar to the bounded-buffer example to create threads up to the maximum number available.

Instead of immediately writing out the result, you should maintain a "log" for the events:

- when a car gets the parking spot (what time, which spot)
- when a car exits the parking lot (what time)

Minimally, show the memory dump of your table to reflect the content of this log.

Extra credit [30 points] Display the output of your log to UART in a human-readable text format.

4. [20 points] Typescript and screenshots

Turn in a typescript showing compilation of your code [2 points], screen shots and explanations for all the parts above [18 points].