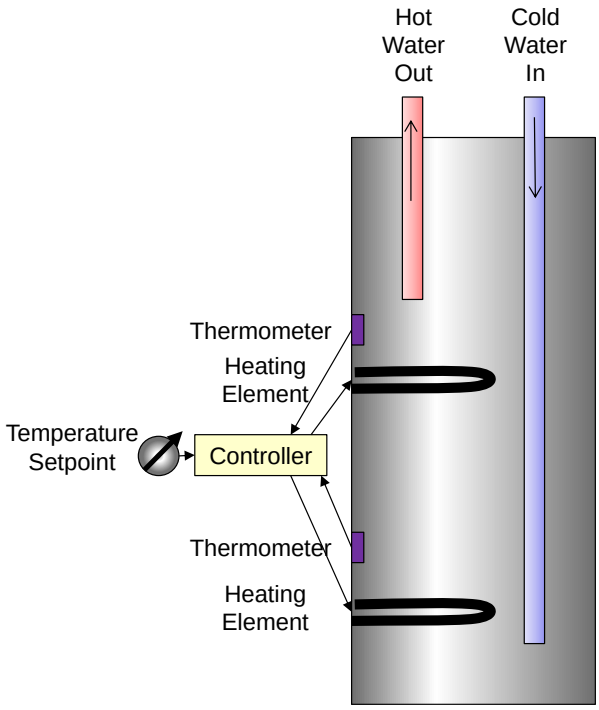
EE3220 Embedded System Design

Assignment 1 – Suggested solution

1. Consider a controller for an electric water heater (as shown below) as an embedded computer system. (20points)



a. What are the inputs devices?

User control (e.g rotary knob) specifying desired (setpoint) temperature for water
Two thermometers.

b. What are the outputs?

Power control for each heating element.
Optional: user display of setpoint and actual temperatures.

c. Name at least two safety features to include, and specify what hardware and/or software is needed for each.

Over-temperature – software needs to monitor the temperature sensor and detect if it is over-range.
Over-pressure – software needs to monitor an additional pressure sensor.
Detect empty tank – could have a pressure switch, or could use software to detect that water temperature isn't changing when heater switches on or off.

d. Describe a useful feature which could you add in software without requiring additional hardware.

Freeze prevention – when in standby mode, prevent water from cooling enough to freeze

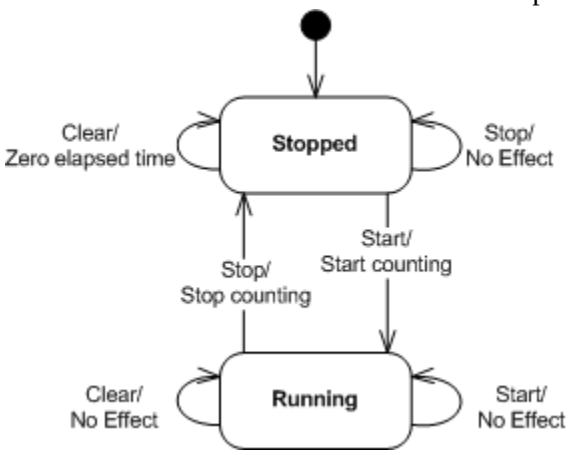
e. Describe two useful energy-saving features which could you add in software if the controller could keep track the time of day.

Use a timer to adjust setpoint based on time (e.g. let water cool down at midnight, start to heat it at 6 AM, or learn from past use of water)
Use a timer to heat water when electricity is less expensive (e.g. at 3 AM)

f. Describe a useful feature which could you add in software if the water heater included an internet connection.

Remote control and monitoring of water heating.
Demand-side management – allows power company to temporarily shut down water heater to reduce peak power requirements.

Q2 – Q4: consider a stopwatch as described here. The state machine below presents the desired behavior.



It has the following hardware.

- Buttons for start, stop and clear functions.
    - Pressing Start starts the stopwatch running. If pressed multiple times, stopwatch continues running without resetting elapsed time.
    - Pressing Stop stops the stopwatch from counting.
    - Pressing Clear zeroes out the elapsed time if the stopwatch is not running. If it is running, the clear button is ignored.
- A timer which triggers an interrupt every 1 ms. The timer drives a counter which counts milliseconds since system start-up, and can be read as elapsed_time_counter.
- A display to show elapsed time with 1 ms resolution. The display must be updated 10 times per second.

2. Design pseudocode for the software using event-triggered scheduling with interrupts. Assume that each button can generate an interrupt. (20 points)

- Use a variable called state to indicate whether the stopwatch is stopped or running

- Use a variable called elapsed_time to track how much time has elapsed since the start button was pressed.
- Use a variable called display_delay to track how many milliseconds remain until the display needs to be updated again.

```
Main thread:
    state = stopped
    display_delay = 100
    elapsed_time = 0

Start ISR:
    state = running

Timer ISR:
    if state == running
        elapsed_time += 1 ms
    display_delay -= 1
    if display_delay == 0 {
        display_delay = 100
        display elapsed_time
    }

Stop ISR:
    state = stopped

Clear ISR
    if state == stopped
        elapsed_time = 0
```

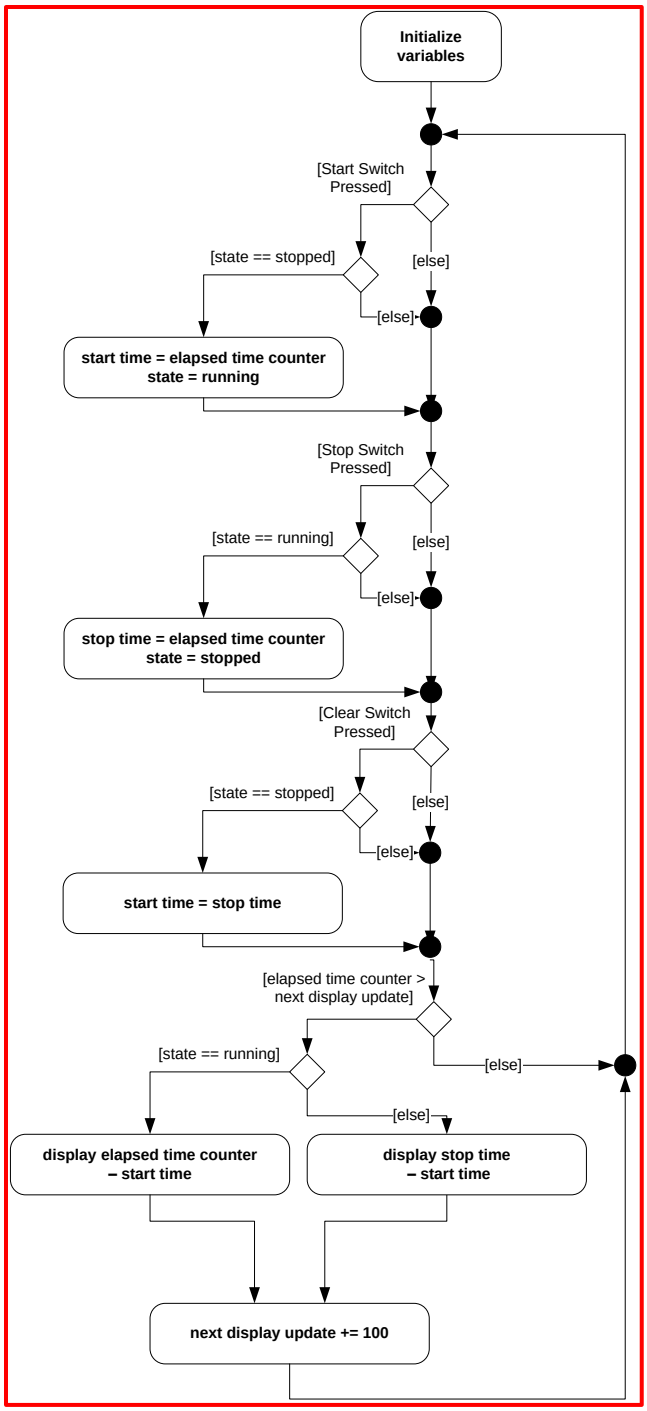
3. Now design pseudocode for the software using a static scheduler without using any interrupts. Assume that the timer updates a hardware register called elapsed_time_register every millisecond. (20 points)

- Use a variable called state to indicate whether the stopwatch is stopped or running
- Use a variable called start_time to record when the start button was pressed.
- Use a variable called stop_time to record when the stop button was pressed.
- Use a variable called next_display_update to indicate when the display needs to be updated next.

```
state = stopped
display elapsed_time_counter
next_display_update = elapsed_time_counter + 100
while (1) {
    if start switch pressed {
        if state == stopped {
            start_time = elapsed_time_counter
```

```
                    state = running
                }
        }
        if stop switch pressed {
                if state == running {
                    stop_time = elapsed_time_counter
                    state = stopped
                }
        }
        if clear switch pressed {
                if state == stopped {
                    start_time = stop_time
                }
        }
        if elapsed_time_counter > next_display_update {
                if (state == running)
                    display elapsed_time_counter - start_time
                else
                    display stop_time – start_time
                next_display_update = next_display_update + 100
        }
}
```

4. **Create a flowchart to represent your solution to the previous question. (10 points)**



5. **Consider a system with the following tasks. We wish to minimize the response time for task C. For each type of scheduler, describe the sequence of processing activities which will lead to the minimum and the maximum response times for task C. Assume that each task is ready to run and there are no further task releases. (10 points)**

Task	Duration
A	3

B	1
C	2

a. Static, non-preemptive scheduler

Best Case: Task C starts immediately (at time 0). $T_r = 0 + 2 = 2$
Worst Case: Task A and Task B run first. $T_r = 0 + 3 + 1 + 2 = 6$

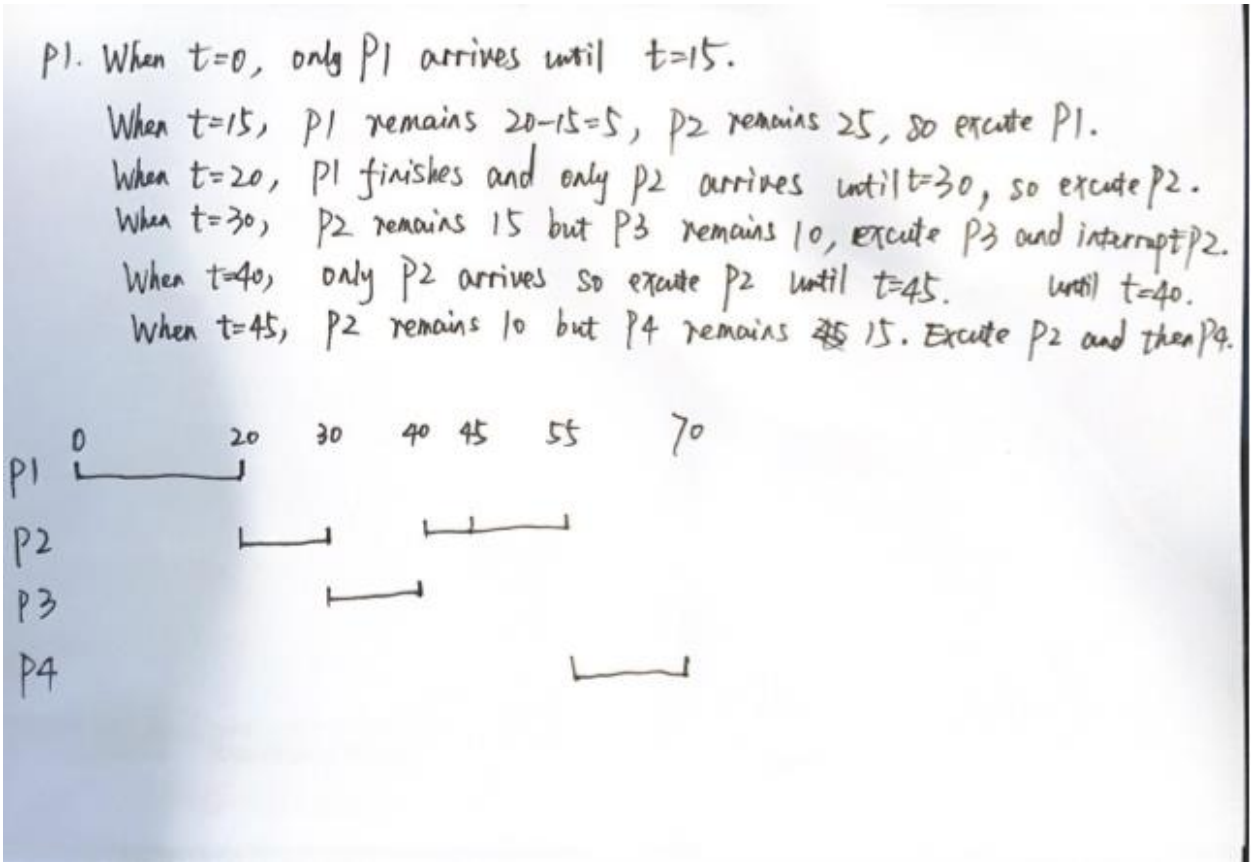
b. Dynamic, non-preemptive scheduler

Best Case: Task C starts immediately (at time 0).
Worst Case: Longest task (A) just started running ε time units ago, so C won't run until it finishes. $T_r = 0 + 3 - \varepsilon + 2 = 5 - \varepsilon$

c. Dynamic, preemptive scheduler

Best Case: Task C starts immediately (at time 0).
Worst Case: Longest task (A) just started running ε time units ago, but it is preempted by C.
$T_r = 0 + 2 = 2$

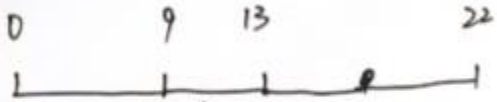
6. (10points)



7. (10 points)

P2. SJF ⇒ Non-preemptive

The figure is



service time − arrival time

For non-preemptive process, Wait time = ~~service~~

Process	Arrival	Service	Wait
P0	0	0	0
P1	1	9	8
P2	2	13	11

Average wait time = $\frac{0+8+11}{3} = \frac{19}{3}$