Lab Assignment 4—*"Free Tonight? Let Me Check My Schedule…"* Round-Robin and Function Queue Scheduling

Overview:

In this lab you'll create a software-plus-hardware system capable of simulating both Round Robin and Function Queue scheduling. The system will include an interactive terminal interface allowing users to define the desired type of scheduling and individual device processing times. The terminal interface will also display the simulation results. Users will press pushbuttons to simulate device service demands. The goal of this lab is to build a system that allows users to analyze the overall performance of both Round Robin and Function Queue scheduling algorithms.

> Note: This assignment is to create a *simulator*, not an *emulator*. In other words, you do not need to create actual tasks or a function queue. Instead, you must accurately simulate how such schedulers would behave.

Specifics:

The simulation must include up to four devices. At any time, a given device may be **Blocked** (waiting for a button press), **Ready** (waiting to run), or **Running**. As devices move between these states, you must inform the user via the serial terminal. After a device has finished running, its **total response time** must be calculated and displayed. Response time includes both the time spent waiting to run (i.e. while in the Ready state) and the time spent running (as specified by the device's "Service Time").

The terminal program (PuTTY, 9600N81, no local echo, no LF added to CR) should display a simple "**>**" prompt when it expects input from the user. The following commands may be entered at the prompt:

`fq` – Set the simulation to Function Queue scheduling.
`rr` – Set the simulation to Round Robin scheduling.
`show` – Show the scheduler type and the service times for all four tasks.
`reset` – Reset all service times to zero (the default).
`help` – Show a help screen with all commands explained.

Also, the user must be able to set service times at the command prompt by entering the device letter (a-d) followed by a space and the service time in milliseconds. It must be possible to specify multiple devices and service times on a single line by separating names and times with spaces. For example, entering

```
>a 3200 b 2500 c 1250 d 1234
```

would set task A's service time to 3.2 seconds (3200 mS), B's service time to 2.5 seconds, etc. Note that all characters received by the simulator should first be converted to lower case and immediately echoed back to the terminal program. **The user must be able to enter device service times in any order**.

The user must press the button associated with a particular device to demand service for that device, and that action should move the device from Blocked to Ready. When a device is Ready or Running, pressing its button shall have no effect. Also, pressing the button for a device with a service time of zero shall have no effect, but you must notify the user that the service request has been ignored due to zero service time. *NB:* **Service requests should occur as soon as a button is pressed** (i.e. don't wait for it to be released).

The hardware for this lab is simply a NodeMCU ESP-12E board with four attached pushbutton switches. As shown in Figure 1, each pushbutton is attached between a specific GPIO pin and ground.

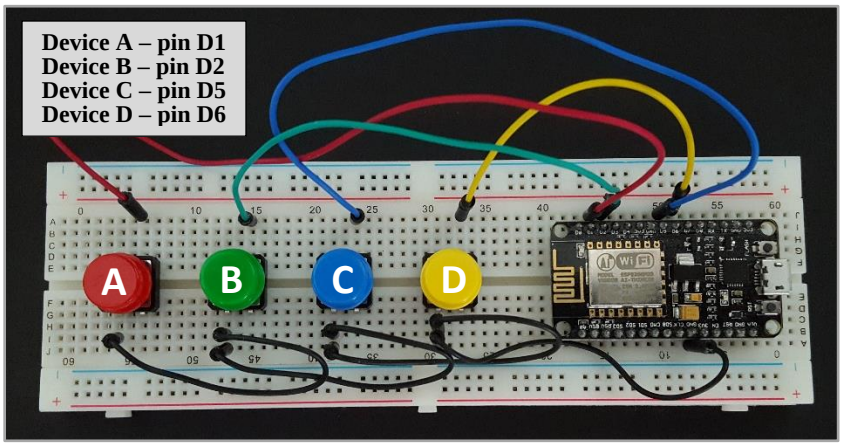


Figure 1. Pushbuttons used to demand service for devices A, B, C and D

Debouncing is required for this lab. Please use the "Bounce2" Arduino library—**no hardware debouncing allowed for this lab**. This handy library lets you create "Bounce" objects which your program can use to periodically sample switch states (pressed, high, low, rose, fell). The library should be installable through your library manager. The Library Git location has some documentation and examples (https://github.com/thomasfredericks/Bounce2, I had reasonable luck with the Bottom example, but the Bounce example looks reasonable as well.)

To help you understand how the simulator should behave, **a demonstrator version of this program is included the project repo**. Appendix A explains how to flash the demo code into your NodeMCU. *You must have your NodeMCU and switches wired exactly as shown in Fig. 1 above for the demonstrator code to work properly.* **And BTW, your simulator should look and behave exactly like the demonstrator. Exactly**.

Also, to help you during the design process, some **pseudocode** is included in Appendix B that shows at least one way of structuring the code. You are under no obligation to use this approach.

Finally, it will be of **great help** to you to work out a set of tests to fully show the functionality of both Round Robin and Function Queue scheduling in the presence of variously ordered interrupts. They most interesting of these should be included in your working demo.

Lab Report

You have **two weeks** for this lab.

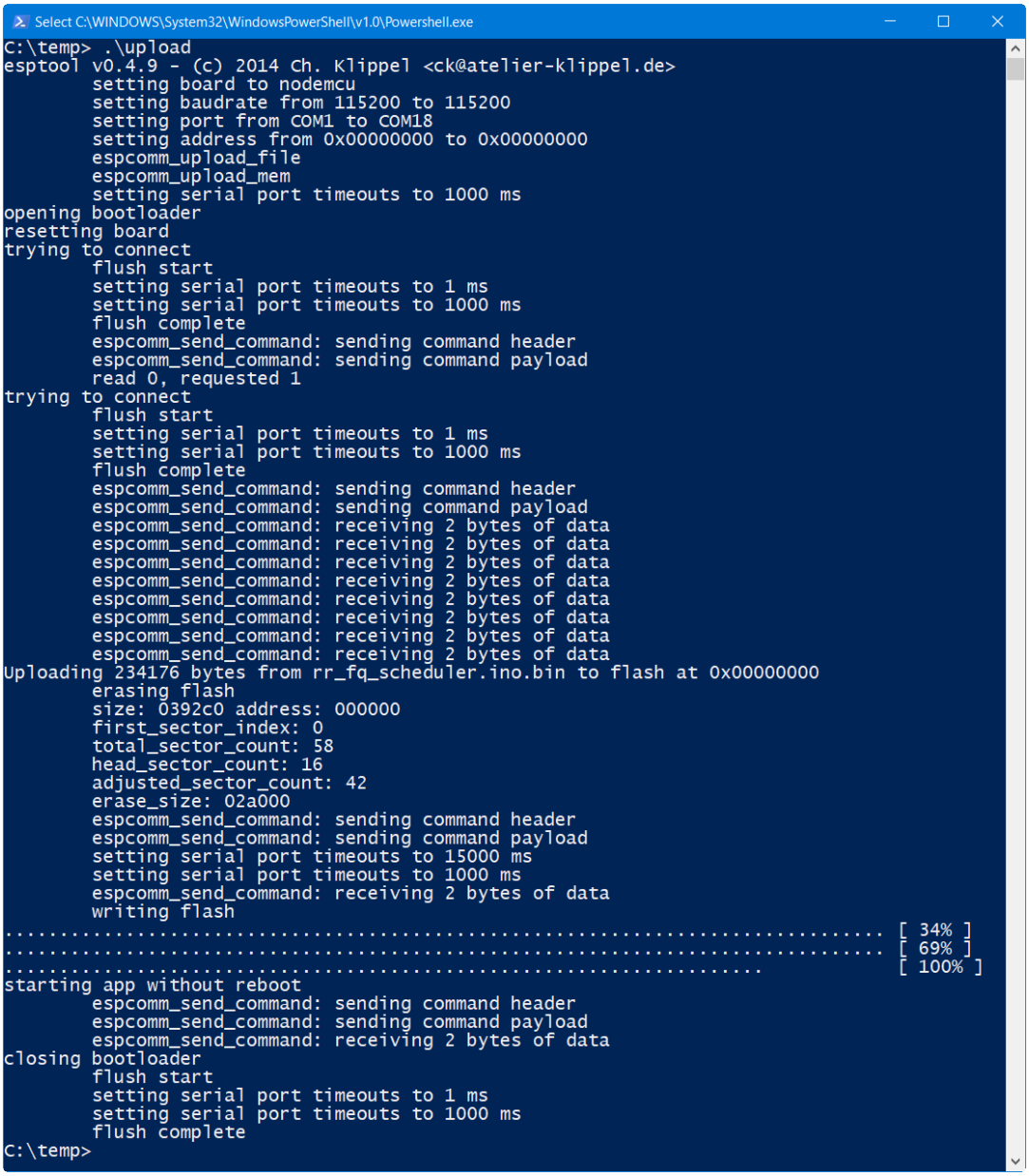
- A lab report is required.
  - Use the template on Canvas (we will review in Lab).
- **Push your final code to your project repo**.
- **Demonstrate your working program at or before the start of lab two weeks from today**.

Appendix A—Flashing Demo Code Into the NodeMCU

A demonstration simulator is available so you can see exactly how your program should operate. The simulator is a binary file that must be loaded directly into the NodeMCU using a special executable program named `esptool.exe`. The `esptool.exe` can be found in the project repo for this lab. You can run `esptool.exe` directly from PowerShell or the Windows command line, but it requires several parameters to work correctly. To make things easier, a batch file named `upload.bat` has been created (also in the project repo) **but you must make one change before running it**. This is explained below.

After cloning the project repo, open PowerShell and use the `cd` command to change to the local repo directory. Verify (via the `dir` command) that `esptool.exe`, `rr_fq_scheduler.ino.bin`, and `upload.bat` are in the current directory.

Connect the NodeMCU to your computer and use the `.\upload_com3.bat` to start the upload. If you have a different COM port, use a text editor to **edit the `upload.bat` batch file, changing the `COM18` parameter to match the COM port of your NodeMCU**. Save your changes. Finally type `.\upload` at the PowerShell command prompt. After a few moments the demonstrator code will be loaded into your NodeMCU, as shown below:



Appendix B—Scheduling Simulator Pseudocode

```
Initialize serial port (9600 N81)
Configure pushbutton pins as inputs with internal pull-up resistors enabled
Create and attach a SW debouncer to each switch
Configure a Ticker timer and a callback routine to update each debouncer's state
Display an initial welcome message
Loop:
  Display the command prompt
  Read a character from serial input
  Convert character to lower case
  Echo the lower case character back to terminal program
  If character != CR
     Add character to current command array
  Else
     Echo back a LF char
     Process the command (help, show, reset, rr, fq) or set device service time(s)
  EndIf

  If RoundRobin
     If "next device to run" is A, and A is Ready and no other devices are Running,
        Move A to Running, display message, and record start- and end times
     Else
        Set "next device to run" to B

     If "next device to run" is B, and B is Ready and...
     ...
  Else
     If A is Ready and no other devices are running
        Move A to Running, display message, and record start- and end times
     Else if B is Ready and...
        ...
  EndIf

  If Device A is Running and Device A's end time <= current time
     Calculate and display A's response time
     Set Device A to Blocked and display "blocked" message
  Else
     If Device B is Running and Device B's end time <= current time
        Calculate and display B's response time
        Set Device B to Blocked and display "blocked" message
     Else
        If Device C is Running and...
            ...
     Else
        Display "All devices blocked" message
     EndIf
  EndIf
EndLoop
```

This pseudocode is not detailed enough to simply translate each line to code. You must also use the demonstration simulator to explore what each action and command does. Together, they should give you a clear idea of how your code should operate.