Project 3 FRQ

1. Initially the xv6 has the scheduling policy of round robin
2. In the context of xv6, multiplexing means to share the limited resources, such as a single CPU, among multiple processes or threads. The challenges can include (but not limited to):

- fairness across different processes or theads
- data synchronization: some resources are shared and require certain synchronization mechanism to ensure the correctness of the programs
- overhead: we use context switches to share a single CPU for multiple processes for example, and this will incur overhead to store current register values and pull the new process's register values and we should try to minimze such overhead
- deadlock: under certain situations, dead lock may happen and lead to the non-termination of the programs.

3. The diagram shows the context switch from shell to cat. The shell process starts from user space and transits to kernel space, stores its register values into its (own) kernel stack via `save`. It then switches to the kernel stack of the scheduler via `swtch` and scheduler picks the cat process to be the next running process. It switches to the kernel stack of the cat process via `swtch`, restores its register values via `restore`, and switch to user space to continue to run the cat process.
4. `switchkvm` means switch kernel virtual memory and `switchuvm` means switch user virtual memory. The `switchkvm` focuses on managing the kernel stack of a process to ensure that the kernel-specific data are properly preserved and restored during context switches. The `switchuvm` focuses on the user stack of a process to make sure that the virtual memory mappings, code and data segments for user processes are correct.
5. Using the diagram below as an example (X means not RUNNABLE and empty just means empty):

Q0: X | X | 8 | | |

Q1: X | X | X | | |

Q2: 37 | 3 | 5 | | |

We start with 8 in Q0. After running its timer slice, it is then shifted to end of Q1.

Q0: X | X | | | |

Q1: X | X | X | 8 | |

Q2: 37 | 3 | 5 | | |

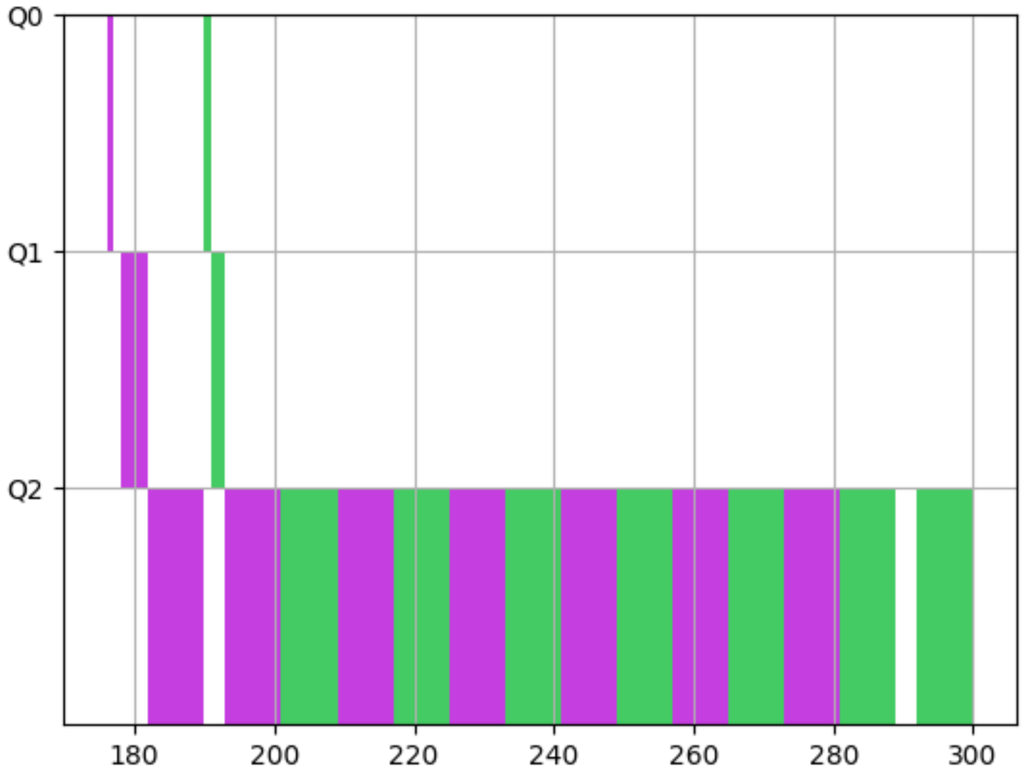
8 is now at the end of Q1 and since it is the first RUNNABLE process in Q1, it consumes its timer slice and is shifted to end of Q2 as below. While the processes are running, all the processes calculate how long they have been waiting and once it reaches the threshold, eg 50 in project3, it will get a boost and be shifted to end of Q0. For example, 37 waiting time reaches the threshold after 8 runs in Q1 (3 and 5 does not yet), so it gets boosted to Q0. The remaining processes in Q2 gets shifted forward as below:

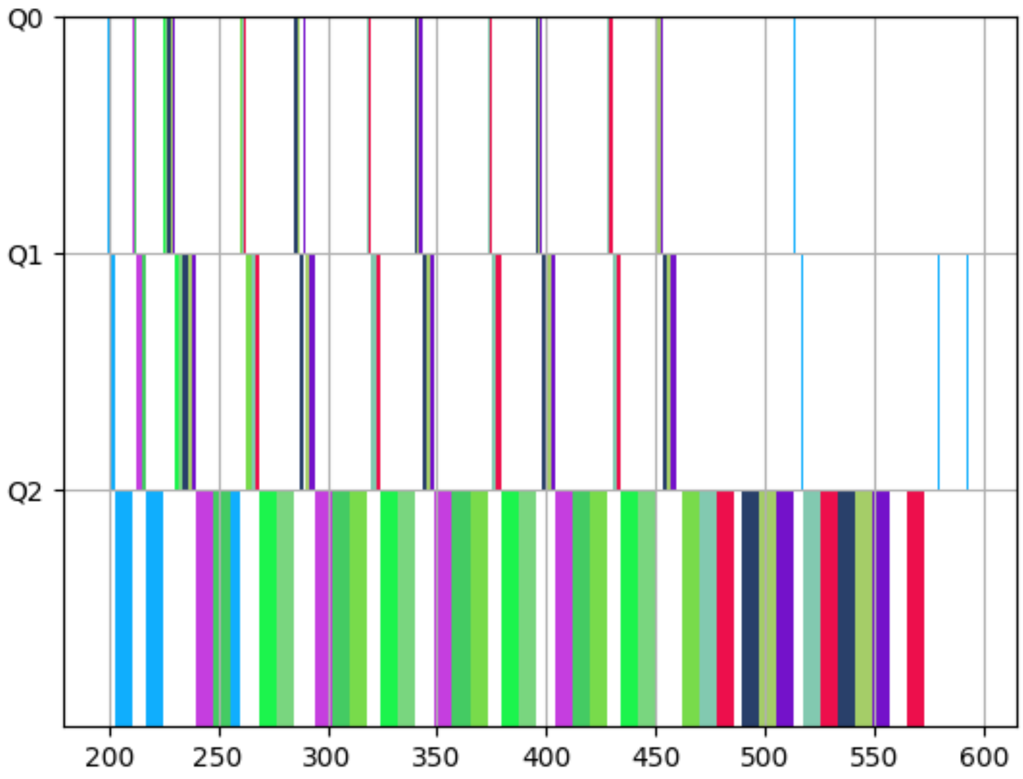
Q0: X | X | 37 | | |

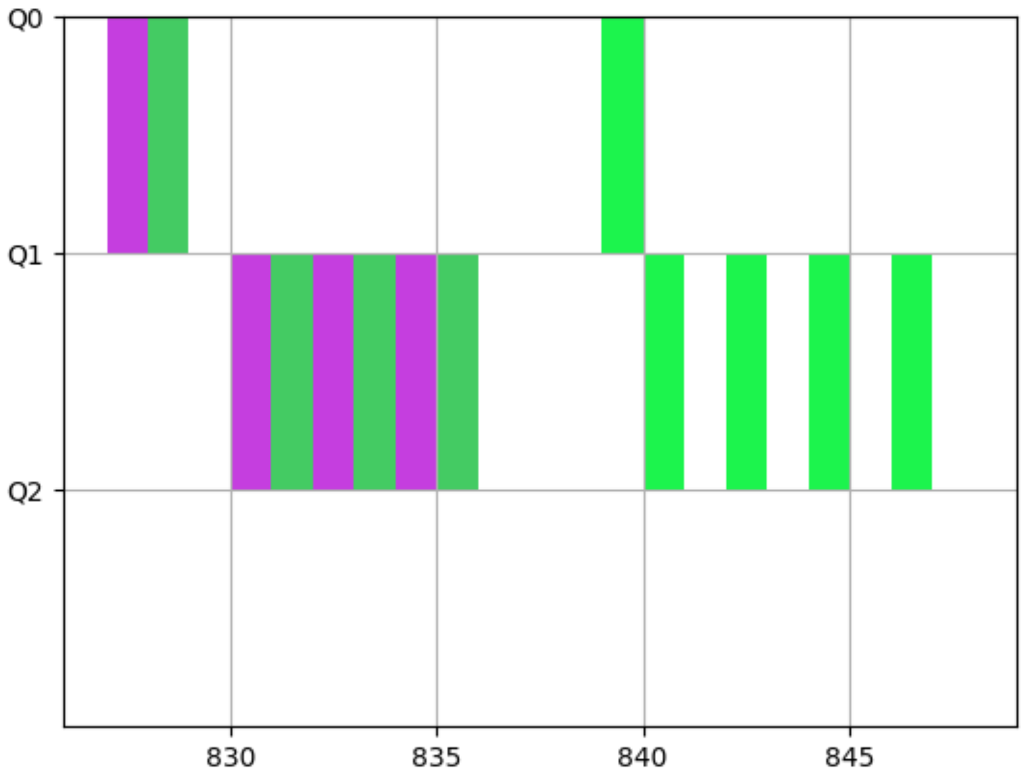
Q1: X | X | X | | |

Q2: 3 | 5 | 8 | | |

In the next few pages are namely the plots for `test1`, `test2`, and `test3`.


Q0
Q1
Q2
180
200
220
240
260
280
300


Q0
Q1
Q2
200
250
300
350
400
450
500
550
600


Q0
Q1
Q2
830
835
840
845