PROJECT 3

Project Requirements

- Build a Rate Monotonic Scheduler with four threads
- Scheduler details:
  - Threads T_1 through T_4
  - Thread T_1 has a period of 1 unit
  - Thread T_2 has a period of 2 units
  - Thread T_3 has a period of 4 units
  - Thread T_4 has a period of 16 units
  - A unit shall be anywhere from 10-100 ms (depending on operating system options)
  - Each thread will execute the same doWork method but run it a different number of times:
    - Thread T_1 executes doWork 100 times
    - Thread T_2 executes doWork 200 times
    - Thread T_3 executes doWork 400 times
    - Thread T_4 executes doWork 1600 times

Project Requirements

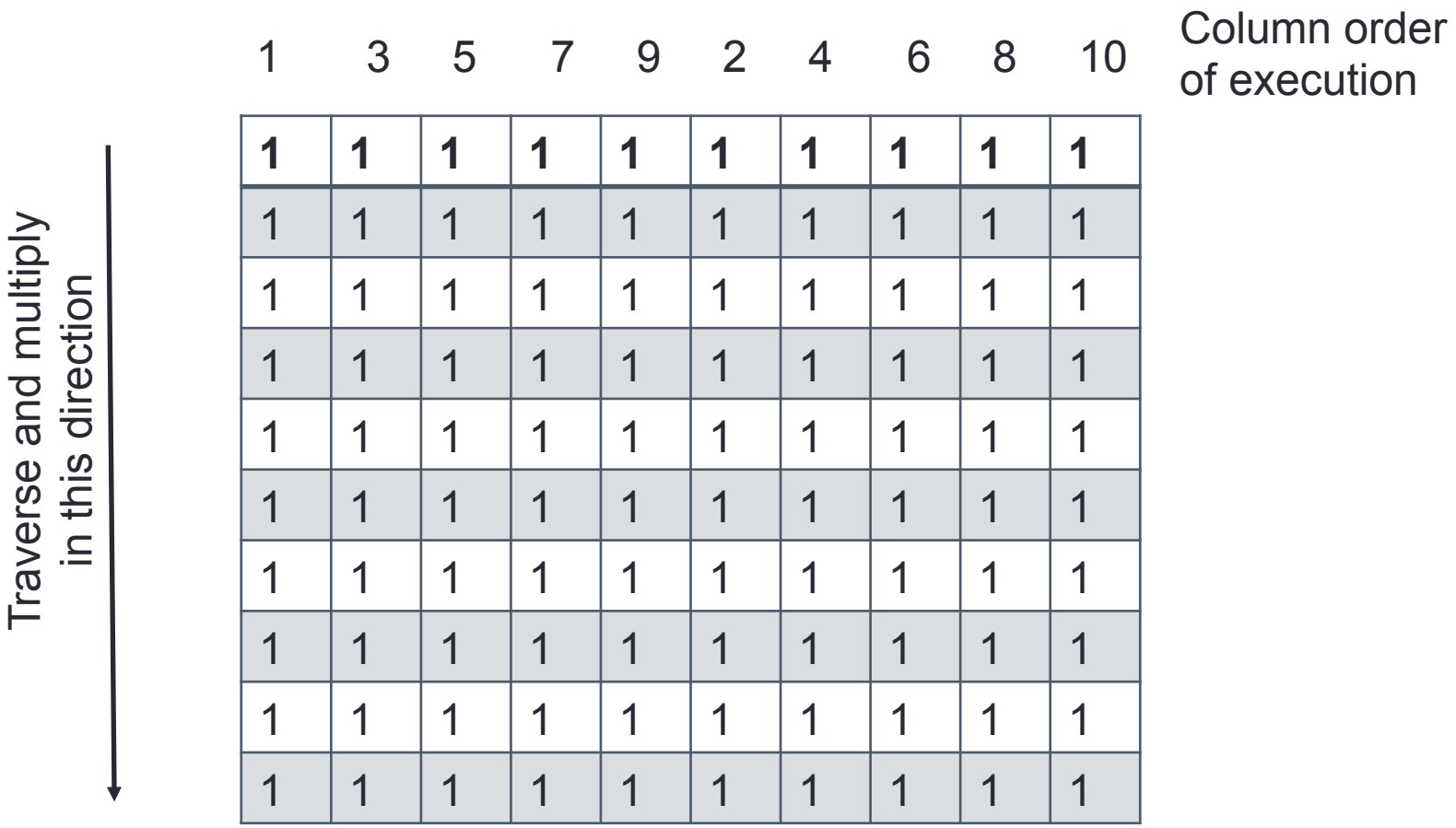
- Scheduler shall have a major frame period of 16 units of time
- Program shall start scheduler and four threads that are to be scheduled
  - Scheduler needs to be woken up by a periodic source (signal/timer/etc) and it shall schedule the threads
  - The program shall run for 10 periods and then terminate, but not before printing out how many times each thread ran
- Each thread shall increment a dedicated counter each time it runs
- The scheduler shall be able to identify if a thread has missed its deadline and keep track of how many times it happens

Project Requirements

- The following test cases shall be demonstrated
  - Nominal case with no overruns
  - Failed case where the doWork function is called as many times as required to lead to an overrun condition in T_2 – what happens to other threads?
  - All results are printed out at the completion of the run to not effect the timing
  - When an overrun condition occurs, the scheduler shall not schedule the thread and skip a period

Project Requirements

- doWork function will do the following:
  - Will multiply the content of each cell of a 10x10 matrix starting with column 0 followed by 5, then 1 followed by 6, etc



Hints

- Remember that you have a scheduler that is orchestrating everything else – separate thread
- Priorities are essential – remember rules of RMS
- Semaphores needed for synchronization
- May need mutex to protect shared data between scheduler and threads – remember priority inversion
- You need to use processor affinity on all your threads (including the scheduler)
- For the overrun conditions, you should **not** schedule the thread that has missed its deadline
  - You will be skipping it for that execution period
- You can initially use a sleep( ) or similar function to set the timing on your scheduler until you work out the synchronization with the other threads and then replace with a timer

Project Artifacts

- Demonstrate by outputting the counters for each thread that shows how many times each one ran and how many times an overrun occurred per thread
  - Can be printed to the screen or sent to a file
- Students must turn in the following:
  - Source code
  - Output of the program
  - A brief design description that explains the design, how were the threads synchronized and dispatched