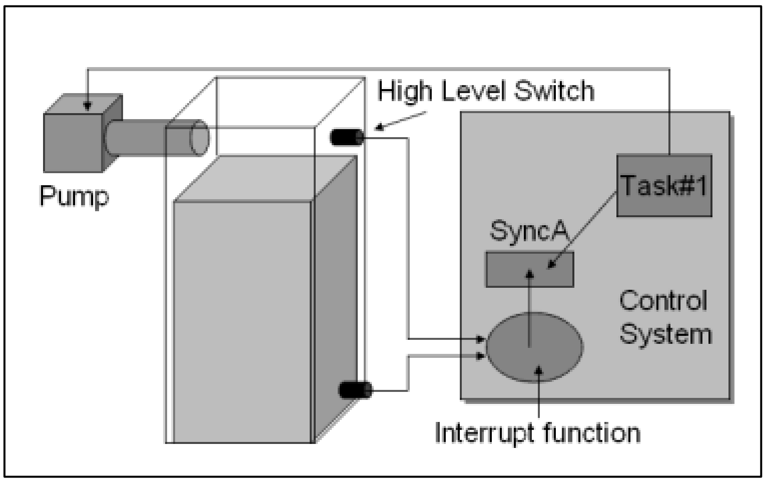
Multitasking and Real-Time

Lars Rikard Rådstoga | 223786

2022-03-13

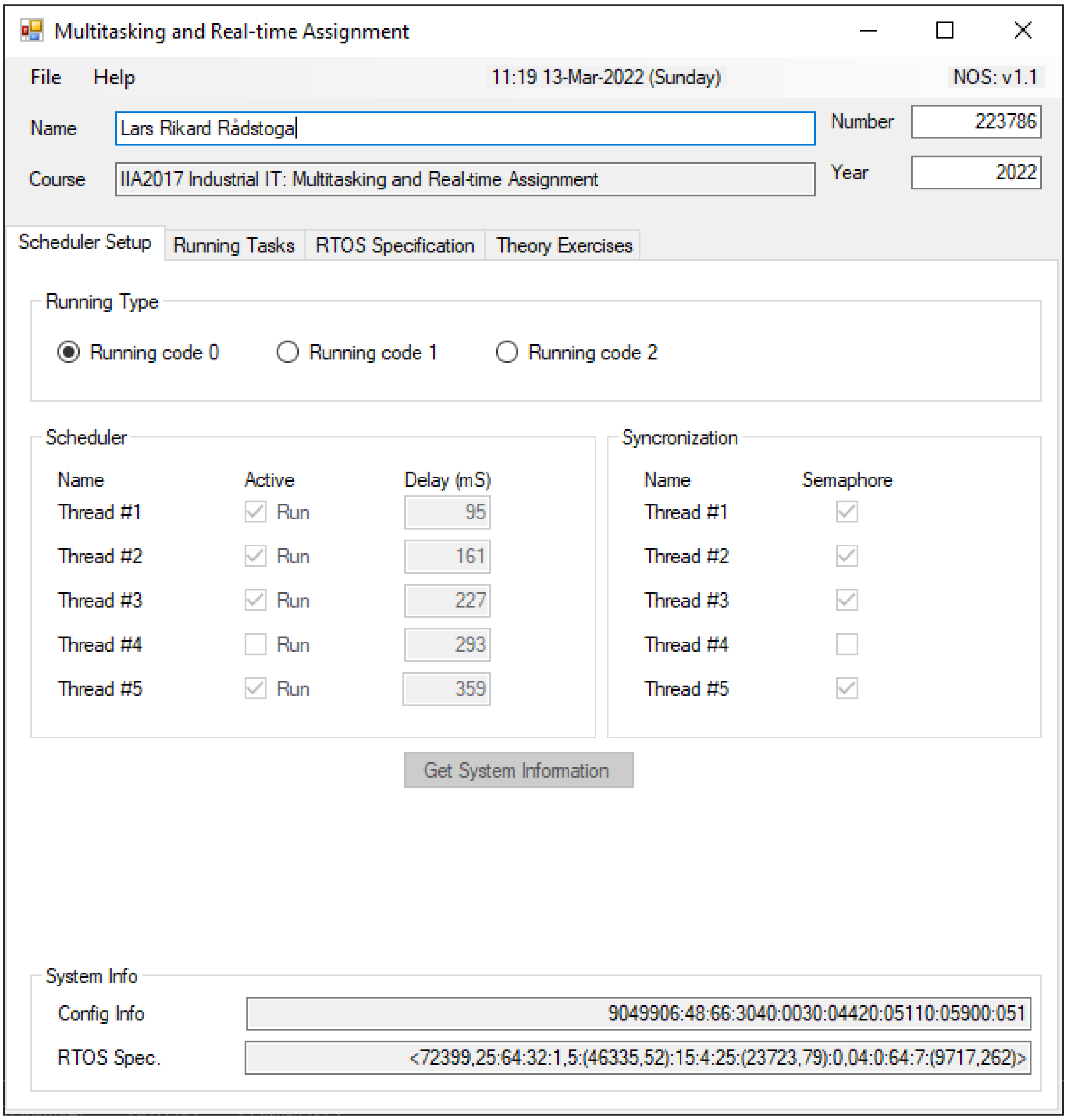


Contents

1 Introduction

Multitasking is very useful in real-time systems as they have strict time requirements. It is therefore good practice to split the program into smaller tasks which have responsibilities of each task containing such time requirements. The multitasking system is then used to schedule and prioritize these smaller tasks to meet time requirements and react to critical events in a systematic manner. The motivation of this report is therefore to investigate and learn about such systems.

Through this report the following subjects will be explored: Run time of tasks, resource sharing, how to develop a simple multitasking application and how to estimate time requirements of a process. Figure 1-1 contains setup information about the assignment.



11:19 13-Mar-2022 (Sunday) NOS: v1.1

Name	Lars Rikard Rådstoga	Number	223786
Course	IIA2017 Industrial IT: Multitasking and Real-time Assignment	Year	2022

Running Type

(●) Running code 0 ( ) Running code 1 ( ) Running code 2

Scheduler

Name	Active	Delay (mS)
Thread #1	✓ Run	95
Thread #2	✓ Run	161
Thread #3	✓ Run	227
Thread #4	Run	293
Thread #5	✓ Run	359

Syncronization

Name	Semaphore
Thread #1	✓
Thread #2	✓
Thread #3	✓
Thread #4	
Thread #5	✓

System Info

Config Info	9049906:48:66:3040:0030:04420:05110:05900:051
RTOS Spec.	<72399,25:64:32:1,5:(46335,52):15:4:25:(23723,79):0,04:0:64:7:(9717,262)>

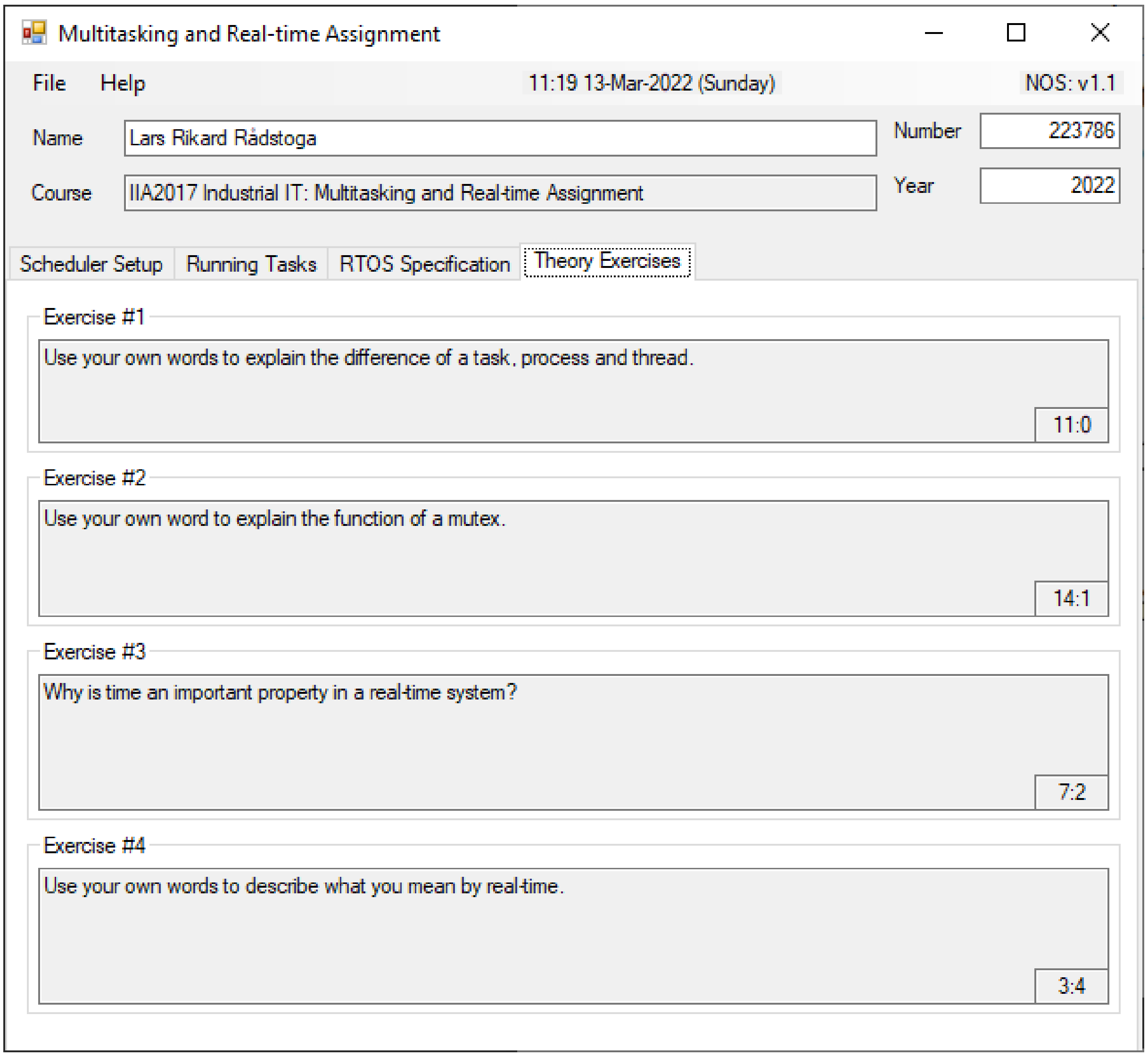
Figure 1-1 Scheduler setup tab.

2 Results

The following subchapters contain proposed solutions for the given exercises.

2.1 Theory

Figure 2-1 shows a program containing 4 exercise texts, these are answered in the following subchapters with their respective titles referencing each exercise.



11:19 13-Mar-2022 (Sunday) NOS: v1.1

Name: Lars Rikard Rådstoga
Number: 223786
Course: IIA2017 Industrial IT: Multitasking and Real-time Assignment
Year: 2022

Exercise #1
Use your own words to explain the difference of a task, process and thread.
11:0

Exercise #2
Use your own word to explain the function of a mutex.
14:1

Exercise #3
Why is time an important property in a real-time system?
7:2

Exercise #4
Use your own words to describe what you mean by real-time.
3:4

Figure 2-1 Machine-picked theory exercises.

2.1.1 Exercise 1: Difference between task, process, and thread

A process is a computer program that is activated in memory. Processes do not share memory resources with other processes but can instantiate other processes and contain multiple threads [1].

A thread is a smaller program part which is contained by a process. Multiple threads contained by the same process can access the same resources such as memory [1].

A task is the vaguest term of the three as it is often used as a synonym for process, light-weight process, thread, step, request, and query. Interestingly, both atomic work units within a thread or the threads themselves can also be referred to as “tasks” [2]. Though in microprocessors of 4, 8 and 16 bits, where the presence of a memory management unit is lacking, tasks refer to a subtype of process with less strict memory control. Advantages of systems with tasks are speed, ease of data exchange and ease of synchronization. Disadvantages are difficulty with problem root tracing, side effects and a strict need to synchronize common data access [3].

2.1.2 Exercise 2: The function of a mutex

A mutex is basically the bouncer of a computer resource. If the resource is busy, the mutex flag says 0 and the bouncer gets the scheduler to order the queue. When the resource is ready the flag says 1 and the mutex will grant access to the next member in the queue, if any, before decrementing the flag back to 0 [3].

2.1.3 Exercise 3: Time as an important property of real-time systems

Real-time systems are generally connected to real-world processes. In a production setting, whether it is a more discrete factory or a continuous process, time requirements and whether they are satisfied can affect both efficiency and quality of the production. Imagine a serial production line: a product is modified n number of times along the production line. If each modification takes the same amount of time, all time requirements are met, then the production line can move smoothly and on time. On the contrary, if a modification is delayed, then the whole production line is delayed. If there is a clear time requirement defined for a modification, then the system can be designed to meet this [3].

2.1.4 Exercise 4: What does real-time mean?

Most of the internet will tell you that real-time means computing or processing something as quick as possible. But, more practically, real-time should mean that computations should meet a defined deadline [3].

2.2 Evaluation of a multitasking system

2.2.1 Analysis

Following is a list of real-time requirements that probably match the system:

Multiple threads shall run in parallel.

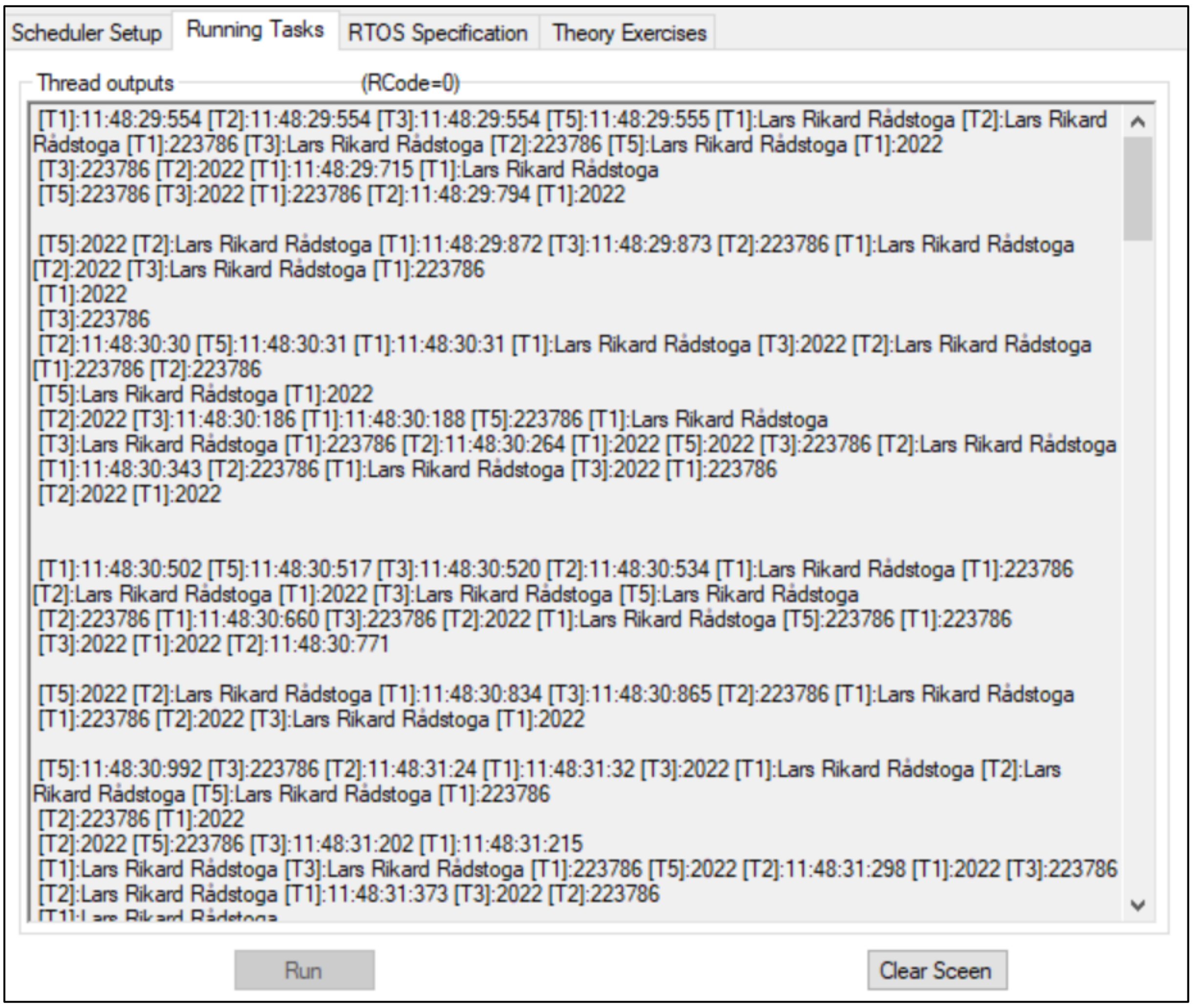
Scheduler shall select which task to run.

Only one thread can write to the output window at a time.

A synchronization service such as a mutex or semaphore is needed to share the output window as a resource.

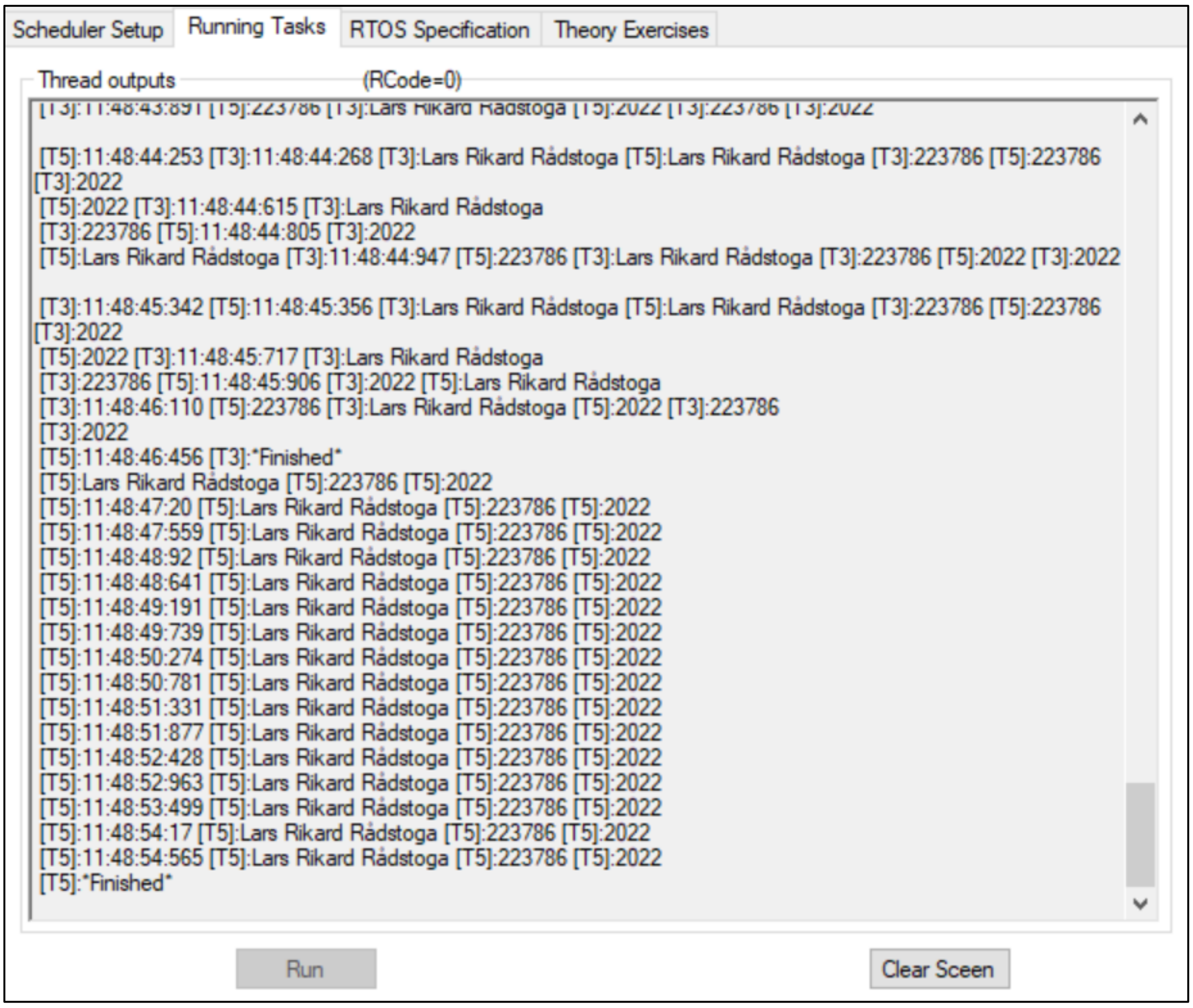
2.2.2 Code 0

The code 0 configuration gives outputs as seen in Figure 2-2 and Figure 2-3. The first printed timestamp is 11:48:29:554. Each loop in all the threads print 4 things. A text editor was used to find the number of items printed per thread; this information was saved in Table 1. Average time per thread is 25s/4 = 6.25s. Time per loop is 6.25s/48 = 0.13s. Delay in each loop is then 0.13s - sum of print functions time = delay.



```
 [T1]:11:48:29:554 [T2]:11:48:29:554 [T3]:11:48:29:554 [T5]:11:48:29:555 [T1]:Lars Rikard Rådstoga [T2]:Lars Rikard
Rådstoga [T1]:223786 [T3]:Lars Rikard Rådstoga [T2]:223786 [T5]:Lars Rikard Rådstoga [T1]:2022
 [T3]:223786 [T2]:2022 [T1]:11:48:29:715 [T1]:Lars Rikard Rådstoga
 [T5]:223786 [T3]:2022 [T1]:223786 [T2]:11:48:29:794 [T1]:2022

 [T5]:2022 [T2]:Lars Rikard Rådstoga [T1]:11:48:29:872 [T3]:11:48:29:873 [T2]:223786 [T1]:Lars Rikard Rådstoga
[T2]:2022 [T3]:Lars Rikard Rådstoga [T1]:223786
 [T1]:2022
 [T3]:223786
 [T2]:11:48:30:30 [T5]:11:48:30:31 [T1]:11:48:30:31 [T1]:Lars Rikard Rådstoga [T3]:2022 [T2]:Lars Rikard Rådstoga
[T1]:223786 [T2]:223786
 [T5]:Lars Rikard Rådstoga [T1]:2022
 [T2]:2022 [T3]:11:48:30:186 [T1]:11:48:30:188 [T5]:223786 [T1]:Lars Rikard Rådstoga
 [T3]:Lars Rikard Rådstoga [T1]:223786 [T2]:11:48:30:264 [T1]:2022 [T5]:2022 [T3]:223786 [T2]:Lars Rikard Rådstoga
 [T1]:11:48:30:343 [T2]:223786 [T1]:Lars Rikard Rådstoga [T3]:2022 [T1]:223786
 [T2]:2022 [T1]:2022


 [T1]:11:48:30:502 [T5]:11:48:30:517 [T3]:11:48:30:520 [T2]:11:48:30:534 [T1]:Lars Rikard Rådstoga [T1]:223786
[T2]:Lars Rikard Rådstoga [T1]:2022 [T3]:Lars Rikard Rådstoga [T5]:Lars Rikard Rådstoga
 [T2]:223786 [T1]:11:48:30:660 [T3]:223786 [T2]:2022 [T1]:Lars Rikard Rådstoga [T5]:223786 [T1]:223786
 [T3]:2022 [T1]:2022 [T2]:11:48:30:771

 [T5]:2022 [T2]:Lars Rikard Rådstoga [T1]:11:48:30:834 [T3]:11:48:30:865 [T2]:223786 [T1]:Lars Rikard Rådstoga
 [T1]:223786 [T2]:2022 [T3]:Lars Rikard Rådstoga [T1]:2022

 [T5]:11:48:30:992 [T3]:223786 [T2]:11:48:31:24 [T1]:11:48:31:32 [T3]:2022 [T1]:Lars Rikard Rådstoga [T2]:Lars
Rikard Rådstoga [T5]:Lars Rikard Rådstoga [T1]:223786
 [T2]:223786 [T1]:2022
 [T2]:2022 [T5]:223786 [T3]:11:48:31:202 [T1]:11:48:31:215
 [T1]:Lars Rikard Rådstoga [T3]:Lars Rikard Rådstoga [T1]:223786 [T5]:2022 [T2]:11:48:31:298 [T1]:2022 [T3]:223786
 [T2]:Lars Rikard Rådstoga [T1]:11:48:31:373 [T3]:2022 [T2]:223786
 [T1]:Lars Rikard Rådstoga
```

Figure 2-2 Start of the output.



```
[T3]:11:48:43:891 [T5]:223786 [T3]:Lars Rikard Rådstoga [T5]:2022 [T3]:223786 [T3]:2022

[T5]:11:48:44:253 [T3]:11:48:44:268 [T3]:Lars Rikard Rådstoga [T5]:Lars Rikard Rådstoga [T3]:223786 [T5]:223786
[T3]:2022
[T5]:2022 [T3]:11:48:44:615 [T3]:Lars Rikard Rådstoga
[T3]:223786 [T5]:11:48:44:805 [T3]:2022
[T5]:Lars Rikard Rådstoga [T3]:11:48:44:947 [T5]:223786 [T3]:Lars Rikard Rådstoga [T3]:223786 [T5]:2022 [T3]:2022

[T3]:11:48:45:342 [T5]:11:48:45:356 [T3]:Lars Rikard Rådstoga [T5]:Lars Rikard Rådstoga [T3]:223786 [T5]:223786
[T3]:2022
[T5]:2022 [T3]:11:48:45:717 [T3]:Lars Rikard Rådstoga
[T3]:223786 [T5]:11:48:45:906 [T3]:2022 [T5]:Lars Rikard Rådstoga
[T3]:11:48:46:110 [T5]:223786 [T3]:Lars Rikard Rådstoga [T5]:2022 [T3]:223786
[T3]:2022
[T5]:11:48:46:456 [T3]:*Finished*
[T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:47:20 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:47:559 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:48:92 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:48:641 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:49:191 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:49:739 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:50:274 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:50:781 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:51:331 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:51:877 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:52:428 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:52:963 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:53:499 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:54:17 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:11:48:54:565 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:*Finished*
```

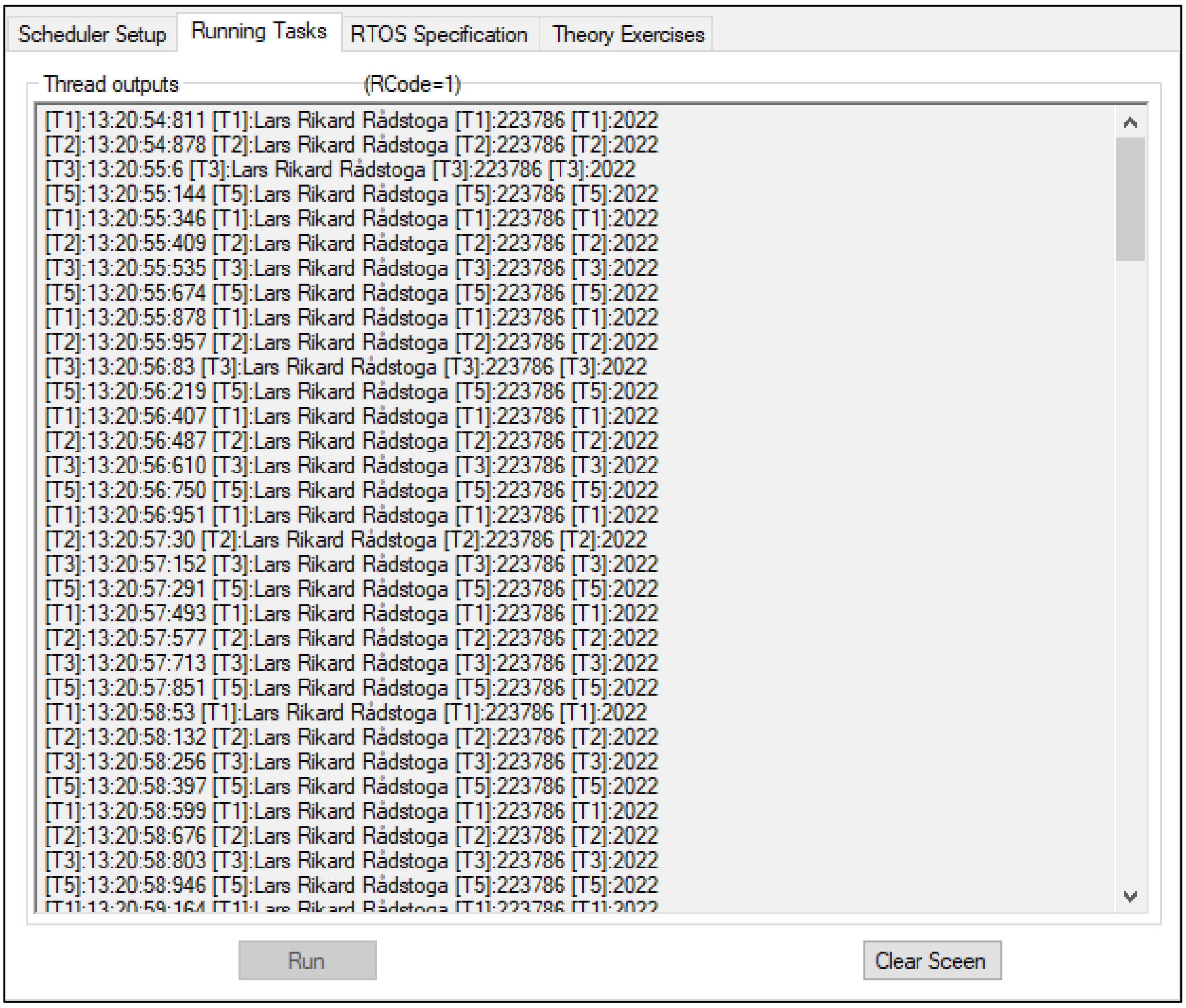
Figure 2-3 End of the output.

Table 1 Thread analysis.

	Prints	Loops	Last timestamp	Δ time from start
Thread 1	193	48	11:48:37:887	8 seconds
Thread 2	193	48	11:48:42:351	13 seconds
Thread 3	193	48	11:48:46:456	17 seconds
Thread 4	None	None	-	-
Thread 5	193	48	11:48:54:565.	25 seconds

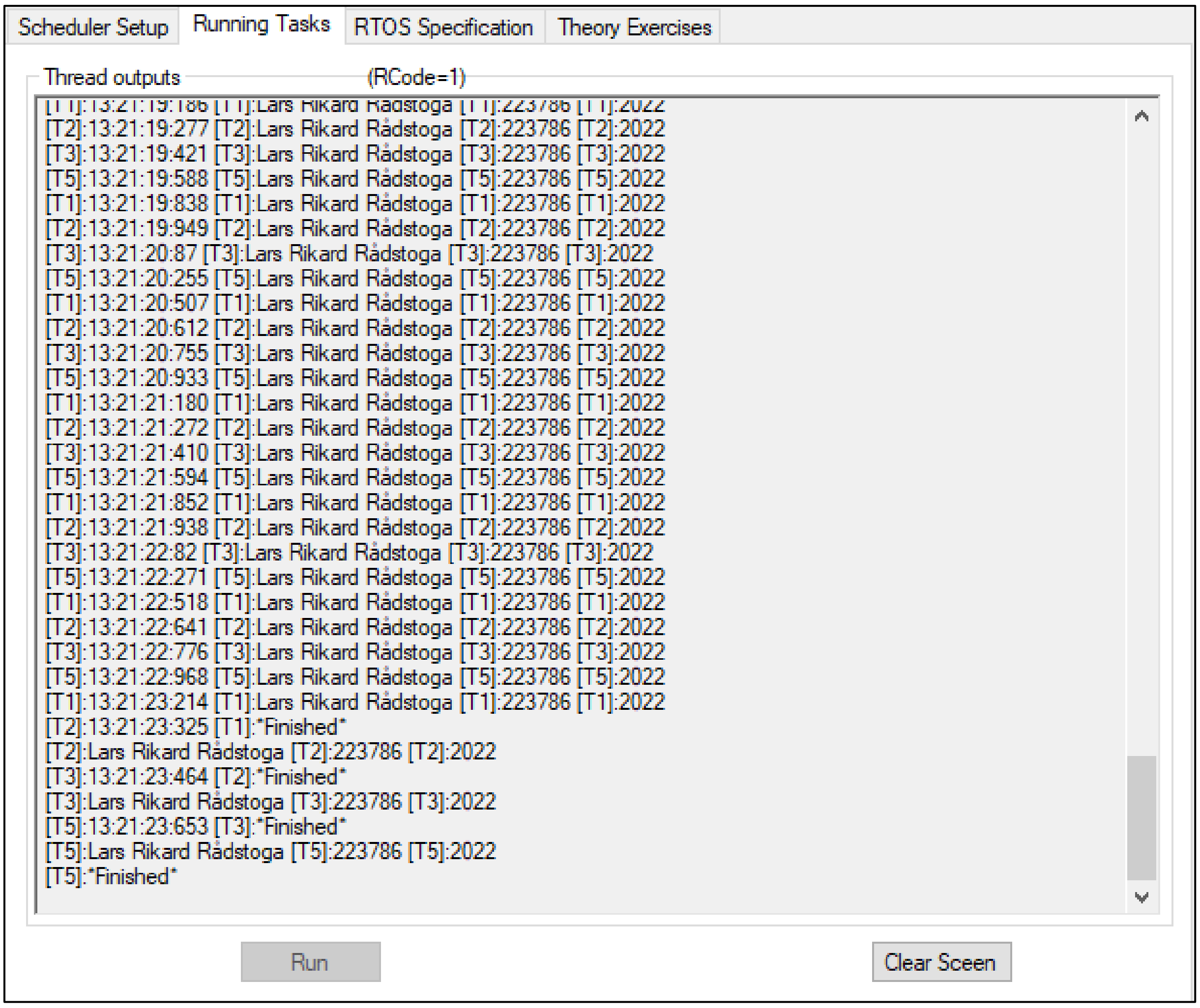
2.2.3 Code 1

The code 1 configuration gives outputs as seen in Figure 2-4 and Figure 2-5. The first printed timestamp is 13:20:54:811. Each loop in all the threads print 4 things. A text editor was used to find the number of items printed per thread; this information was saved in Table 2. Average time per thread is 29s/4 = 7s. Time per loop is 7s/48 = 0.146 seconds. Delay in each loop is then 0.146s - sum of print functions time = delay.



```
[T1]:13:20:54:811 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:20:54:878 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:20:55:6 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:20:55:144 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:20:55:346 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:20:55:409 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:20:55:535 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:20:55:674 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:20:55:878 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:20:55:957 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:20:56:83 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:20:56:219 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:20:56:407 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:20:56:487 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:20:56:610 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:20:56:750 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:20:56:951 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:20:57:30 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:20:57:152 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:20:57:291 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:20:57:493 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:20:57:577 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:20:57:713 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:20:57:851 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:20:58:53 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:20:58:132 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:20:58:256 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:20:58:397 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:20:58:599 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:20:58:676 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:20:58:803 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:20:58:946 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:20:59:164 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
```

Figure 2-4 Start of the output.



```
[T1]:13:21:19:186 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:21:19:277 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:21:19:421 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:21:19:588 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:21:19:838 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:21:19:949 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:21:20:87 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:21:20:255 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:21:20:507 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:21:20:612 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:21:20:755 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:21:20:933 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:21:21:180 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:21:21:272 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:21:21:410 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:21:21:594 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:21:21:852 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:21:21:938 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:21:22:82 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:21:22:271 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:21:22:518 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:21:22:641 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:21:22:776 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:21:22:968 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:13:21:23:214 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:13:21:23:325 [T1]:"Finished"
[T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:13:21:23:464 [T2]:"Finished"
[T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:13:21:23:653 [T3]:"Finished"
[T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T5]:"Finished"
```

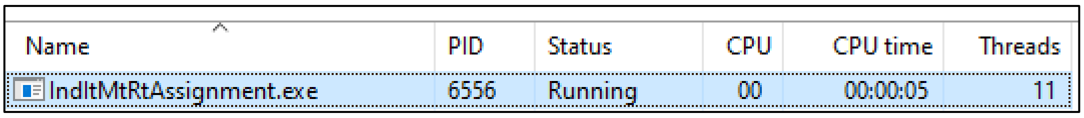
Figure 2-5 end of output.

Table 2 results from code 1’s outputs.

	Prints	Loops	Last timestamp	Δ time from start
Thread 1	193	48	13:21:23:325	29 seconds
Thread 2	193	48	13:21:23:464	29 seconds
Thread 3	193	48	13:21:23:653	29 seconds
Thread 4	None	None	-	-
Thread 5	193	48	13:21:23:653	29 seconds

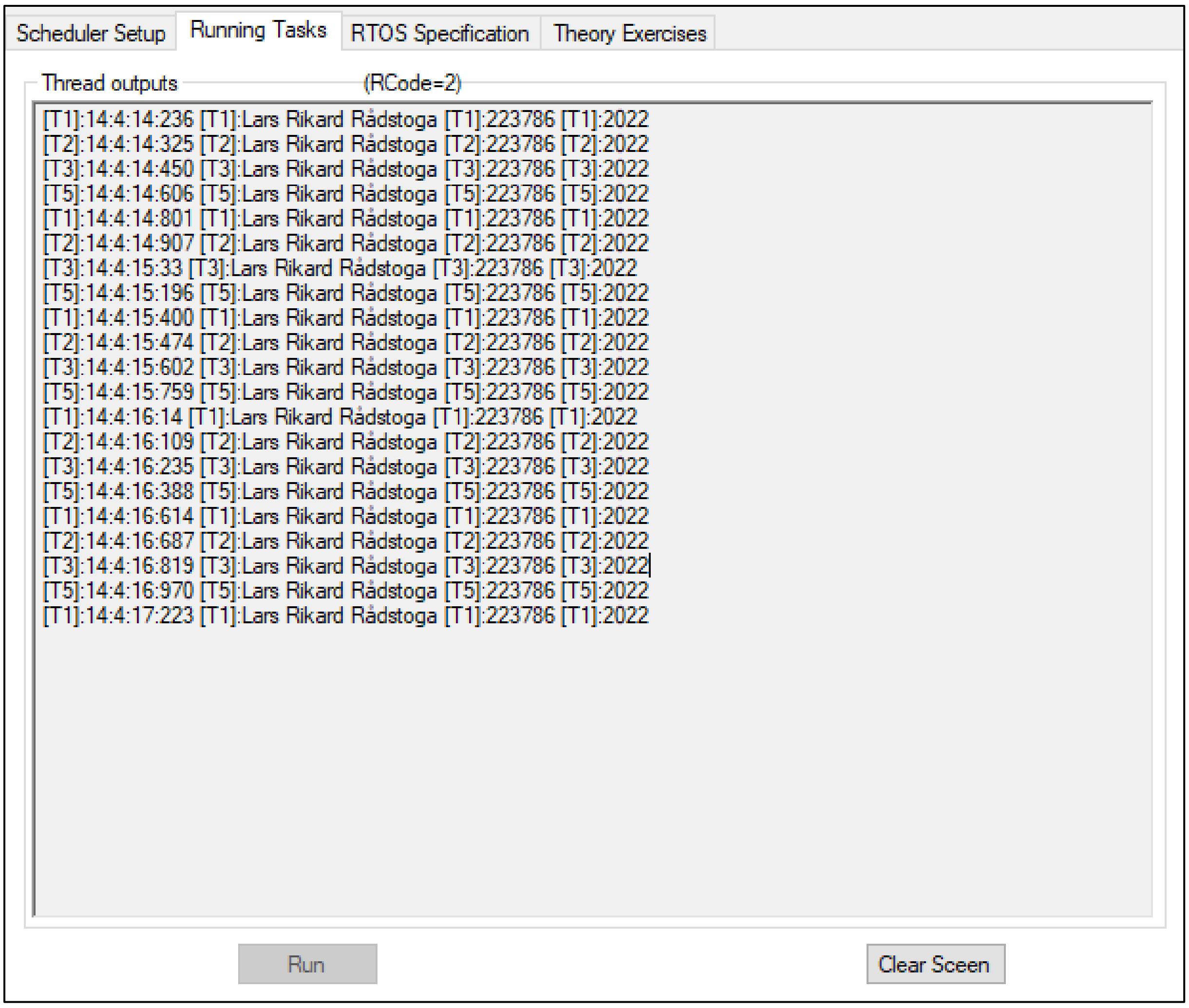
2.2.4 Code 2

Looks like code 2 generates some threads but they are eventually all waiting. CPU time stops together will the thread count, see Figure 2-6. Figure 2-7 shows



Name	PID	Status	CPU	CPU time	Threads
IndItMtRtAssignment.exe	6556	Running	00	00:00:05	11

Figure 2-6 task manager details.



(RCode=2)

```
[T1]:14:4:14:236 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:14:4:14:325 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:14:4:14:450 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:14:4:14:606 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:14:4:14:801 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:14:4:14:907 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:14:4:15:33 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:14:4:15:196 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:14:4:15:400 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:14:4:15:474 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:14:4:15:602 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:14:4:15:759 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:14:4:16:14 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:14:4:16:109 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:14:4:16:235 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:14:4:16:388 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:14:4:16:614 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
[T2]:14:4:16:687 [T2]:Lars Rikard Rådstoga [T2]:223786 [T2]:2022
[T3]:14:4:16:819 [T3]:Lars Rikard Rådstoga [T3]:223786 [T3]:2022
[T5]:14:4:16:970 [T5]:Lars Rikard Rådstoga [T5]:223786 [T5]:2022
[T1]:14:4:17:223 [T1]:Lars Rikard Rådstoga [T1]:223786 [T1]:2022
```

Figure 2-7 Output window is frozen.

2.2.5 Conclusion

Syntax error: can’t conclude without initial hypothesis.

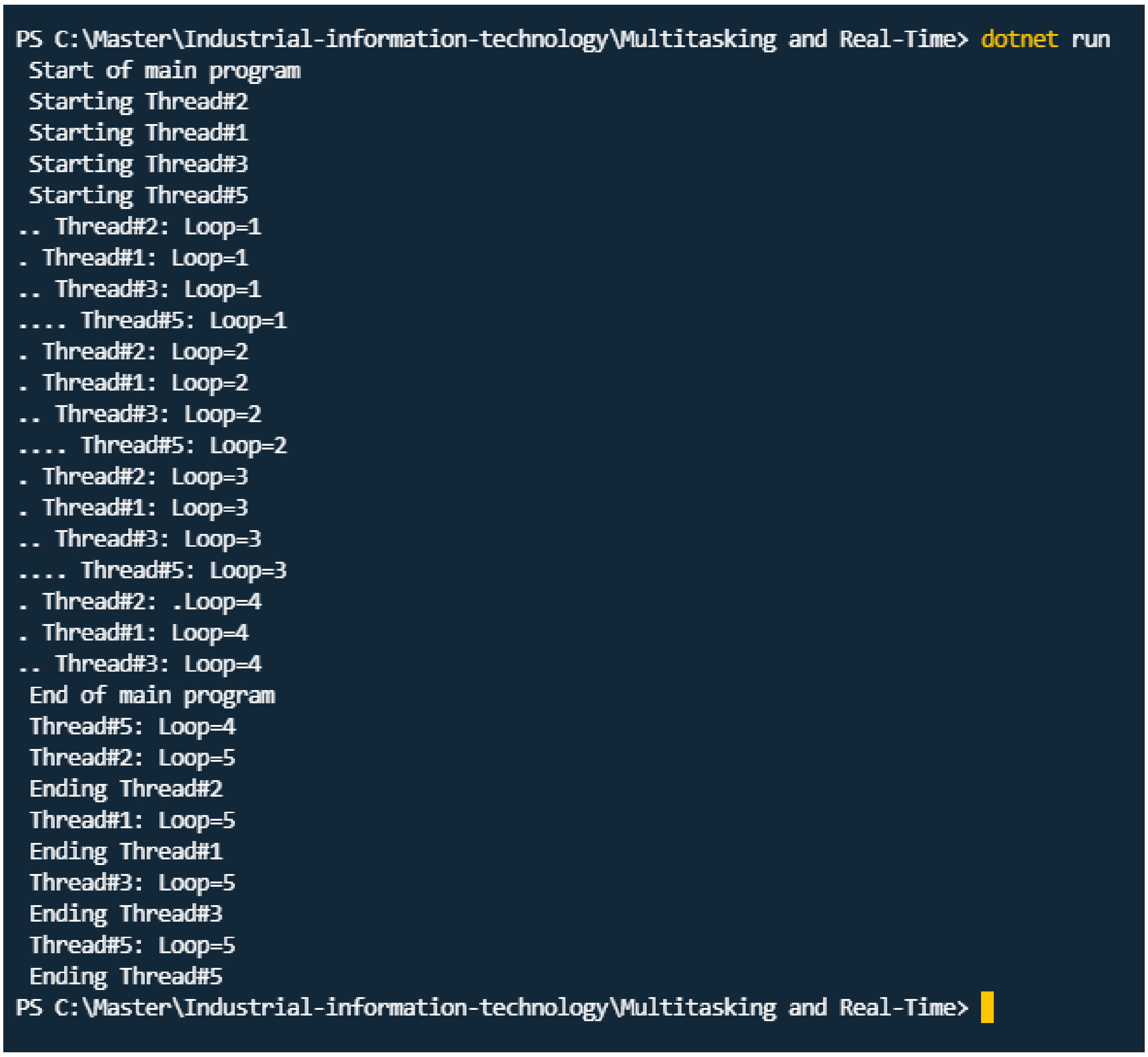
2.3 Development of a multitasking system

A multitasking system was created to replicate some of the behavior seen in Code 1. The code is based upon source code found on page 438 in [3]. The following pseudo code describes the functionality of the program which uses a semaphore for resource sharing:

1. The main class creates a semaphore object with a maximum count of 1.
2. The required 4 threads are created by initializing the ThreadClass as objects with names, semaphore reference and delay as specified in the scheduler setup page.
    a. ThreadClass objects create the threads.

3. The main thread prints "." to the console every 100ms.
4. Simultaneously the threads within the ThreadClass objects each contain loops that first sleeps a specified amount and then try to print some info to the screen. Each iteration of the loop calls WaitOne at the start and Release on the semaphore at the start and end of the loop, respectively.

The source code of the program can be found in Appendix A: Multitasking program with semaphore and a screenshot of the console output using the built in PowerShell console in VS Code can be found in Figure 2-8.

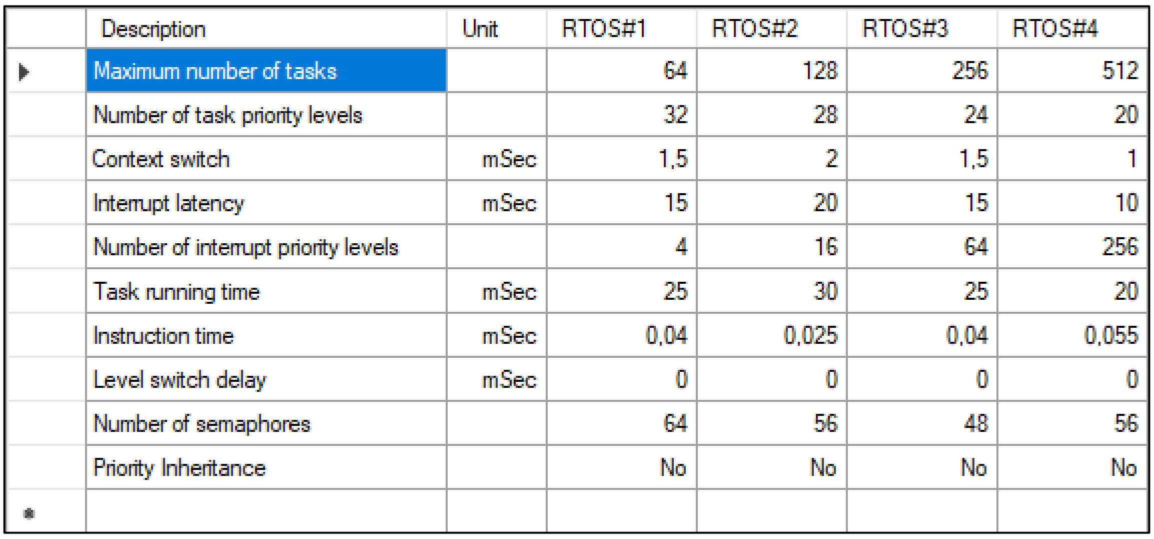


```
PS C:\Master\Industrial-information-technology\Multitasking and Real-Time> dotnet run
 Start of main program
 Starting Thread#2
 Starting Thread#1
 Starting Thread#3
 Starting Thread#5
.. Thread#2: Loop=1
. Thread#1: Loop=1
.. Thread#3: Loop=1
.... Thread#5: Loop=1
. Thread#2: Loop=2
. Thread#1: Loop=2
.. Thread#3: Loop=2
.... Thread#5: Loop=2
. Thread#2: Loop=3
. Thread#1: Loop=3
.. Thread#3: Loop=3
.... Thread#5: Loop=3
. Thread#2: .Loop=4
. Thread#1: Loop=4
.. Thread#3: Loop=4
 End of main program
 Thread#5: Loop=4
 Thread#2: Loop=5
 Ending Thread#2
 Thread#1: Loop=5
 Ending Thread#1
 Thread#3: Loop=5
 Ending Thread#3
 Thread#5: Loop=5
 Ending Thread#5
PS C:\Master\Industrial-information-technology\Multitasking and Real-Time>
```

Figure 2-8 Output from the multitasking system.Appendix A:

2.4 Time requirements of a real-time system

The system needs to perform maximum 20 instructions in the interrupt function. A synchronization mechanism is used to inform the control task, task#1, to control the pump. Task #1 needs to perform between 150 and 200 instructions before the state of the pump can be changed. The state change delay of the pump is maximum 200ms. Task #1 will be the only task at the highest priority running level. But other tasks can also run with a set run time. The pump must be turned off within 250ms after an active signal from any of the level switches. Calculations regarding the total time to change a pump state has been calculated in Table 3 using information from the RTOS specification seen in Figure 2-9. The calculation has been performed using the formula as seen in eq. **2**-**1**. The calculations show that only the fourth OS, RTOS#4, satisfy the real time requirements.



	Description	Unit	RTOS#1	RTOS#2	RTOS#3	RTOS#4
▶	Maximum number of tasks		64	128	256	512
	Number of task priority levels		32	28	24	20
	Context switch	mSec	1,5	2	1,5	1
	Interrupt latency	mSec	15	20	15	10
	Number of interrupt priority levels		4	16	64	256
	Task running time	mSec	25	30	25	20
	Instruction time	mSec	0,04	0,025	0,04	0,055
	Level switch delay	mSec	0	0	0	0
	Number of semaphores		64	56	48	56
	Priority Inheritance		No	No	No	No
*						

Figure 2-9

Table 3

Description	Unit	RTOS#1	RTOS#2	RTOS#3	RTOS#4
Total pump state change time	mSec	251,8	259,5	251,8	244,1

$$250ms \geq task\ running\ time + 2 \times context\ switch + interrupt\ latency + 220 \times instruction\ time + 200ms \qquad 2\text{-}1$$

3 Summary

Through this report the following subjects have been explored: Run time of tasks, resource sharing, how to develop a simple multitasking application and how to estimate time requirements of a process.

4 Appendices

Appendix A: Multitasking program with semaphore

5 Appendix A: Multitasking program with semaphore

```
using System;
using System.Threading;
namespace ThreadSys
{
   // The application
   class ThreadSys
   {
      //Sempahore to simulate limited resource
      private static Semaphore resource;
      /// <summary>
      /// Start of the main program
      /// </summary>
      static void Main(string[] args)
      {
         Console.WriteLine(" Start of main program ");
         // Making sempahore that can only serve one thread at a time
         resource = new Semaphore(1, 1);
         // Making 4 threads according to the scheduler tab
         ThreadClass ct1 = new ThreadClass("Thread#1", resource, 95);
         ThreadClass ct2 = new ThreadClass("Thread#2", resource, 161);
         ThreadClass ct3 = new ThreadClass("Thread#3", resource, 227);
         ThreadClass ct5 = new ThreadClass("Thread#5", resource, 359);
         // Wait while the threads are running ...
         for (int cnt = 0; cnt < 30; cnt++)
         {
            Console.Write(".");
            Thread.Sleep(100);
         }
         // End of main program
         Console.WriteLine(" End of main program ");
      }
```

```
}

/// <summary>
/// Threadclass
/// </summary>
class ThreadClass
{
    int loopCnt, loopDelay;
    Semaphore resource;
    Thread cThread;
    public ThreadClass(string name, Semaphore resource, int delay)
    {
        loopCnt = 0;
        loopDelay = delay;
        this.resource = resource;
        cThread = new Thread(new ThreadStart(this.run));
        cThread.Name = name;
        cThread.Start();
    }
    // The main function in the ThreadClass
    void run()
    {
        Console.WriteLine(" Starting " + cThread.Name);
        do
        {
            resource.WaitOne();
            loopCnt++;
            Thread.Sleep(loopDelay);
            Console.Write(" ");
            Console.Write(cThread.Name);
            Console.Write(": ");
            Console.WriteLine("Loop=" + loopCnt);
            resource.Release();
        } while (loopCnt < 5);
```

```
            // Ending of the thread
            Console.WriteLine(" Ending " + cThread.Name);
        }
    }
}
```