Multitasking and Real-Time

Lars Rikard Rådstoga | 223786

2022-03-13

\*figure\*

Contents

[1 Introduction 1](#_Toc98065196)

[2 Results 2](#_Toc98065197)

[2.1 Theory 2](#_Toc98065198)

[2.2 Evaluation of a multitasking system 2](#_Toc98065199)

[2.3 Development of a multitasking system 3](#_Toc98065200)

[2.4 Time requirements of a real-time system 3](#_Toc98065201)

[3 Summary/Conclusion 3](#_Toc98065202)

[4 Appendix A: OPC Client with Python 2](#_Toc98065203)

# Introduction

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.

Explain the context

Explain the problem/motivation

Explain the aims/purpose

Outline the report structure

Graphical user interface

Description automatically generated

Figure 1‑1 Scheduler setup tab.

# Results

## Theory

Figure 2‑1 shows a program that has picked out four topics to explain.

Graphical user interface, text, application, email

Description automatically generated

Figure 2‑1 Machine-picked theory exercises.

### 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].

### 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].

### 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].

### 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].

## Evaluation of a multitasking system

### 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.

### Code 0

Graphical user interface, text, application

Description automatically generated

Figure 2‑2 Start of the output.

Graphical user interface, text

Description automatically generated

Figure 2‑3 End of the output.

First time stamp 11:48:29:554

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 |

Average time per thread is 25/4 = 6.25 seconds. Time per loop is 6.25s/48 = 0.13 seconds. Delay in each loop is then 0.13s - sum of print functions time = delay.

### Code 1

Graphical user interface, table

Description automatically generated

Figure 2‑4

Graphical user interface

Description automatically generated

Figure 2‑5

First time stamp 13:20:54:811

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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 |

### Code 2

Looks like code 2 generates some threads but they are eventually all waiting. CPU time stops together will the thread count.

Table

Description automatically generated with medium confidence

Figure 2‑6

Graphical user interface, text

Description automatically generated

Figure 2‑7

## Development of a multitasking system

## Time requirements of a real-time system

# Summary/Conclusion

Appendices

# Appendix A: OPC Client with Python