# Real-Time Systems Assignment Report

A.Y. 2024/2025

**Group Members:**
Alessandro Binci – 0001189831 — Gianluca Vairo – 0001192707 — Pietro Luzzi – 0001176819

## 1. Application Description and Feasibility Analysis

This real-time application simulates a robotic bartender system that prepares and delivers drinks to customers. The main goal is to coordinate multiple robotic operations with shared resource access, under real-time constraints.

### Task Structure

The application uses four periodic tasks responsible for processing and delivering ordered drinks. Each task models a specific phase of the robotic bar service pipeline and has a distinct period and worst-case execution time (WCET):

- **OrderProcessingTask** – receives the main drink order (gin, wine, or beer).
  *Period (Ti): 6 ticks, WCET (Ci): 1.0*
- **DrinkOptionTask** – handles the optional specification (e.g., gin tonic vs gin lemon, red vs white wine, light vs IPA beer).
  *Period (Ti): 9 ticks, WCET (Ci): 2*
- **DrinkPourTask** – pours the selected drink.
  *Period (Ti): 12 ticks, WCET (Ci): 3*
- **FeedbackTask** – notifies the user that their drink is ready.
  *Period (Ti): 15 ticks, WCET (Ci): 1.0*

### Shared Resources and Resource Access Protocols

Three shared resources are used:
**Ice Machine** ($a$) – required for gin-based drinks; **Serving Counter** ($b$) – accessed by all drinks before delivery; **Pouring Pump** ($c$) – used by all types of drinks to pour the beverage.
Four resource access protocols are supported and selectable at simulation start:
**NOR** - no resources; **NOP** – no synchronization; **NPP** – non-preemptive protocol to minimize interference; **PIP** – priority inheritance protocol to control inversion.

### Feasibility Analysis

The set of periodic tasks for the robotic bar application uses four tasks, each with different periods $Ti$, worst-case execution times $Ci$ (WCET) and the phase $\phi i$. These tasks model the stages of the preparation of drinks (Table 1 and Figure 1).

The feasibility analysis was performed under Rate Monotonic (1) scheduling using utilization bound test and the extended response time analysis (2). The Table 2 reflects (approximated) blocking times due to shared resource access under two different protocols.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{1/n} - 1\right) \tag{1}$$

$$R_i^{(k+1)} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j \tag{2}$$

| Task | $T_i$ | $C_i$ | $\phi_i$ | $\delta a$ | $\delta b$ | $\delta c$ |
|---|---|---|---|---|---|---|
| OrderProc. | 6 | 1.0 | 0 | – | – | 0.3 |
| DrinkOpt. | 9 | 2.0 | 2 | – | 0.5 | – |
| DrinkPour | 12 | 3.0 | 4 | 0.5 | 1.0 | 0.3 |
| Feedback | 15 | 1.0 | 5 | – | – | 0.3 |

Table 1: Periodic Tasks and Shared Resource Usage (in ticks)

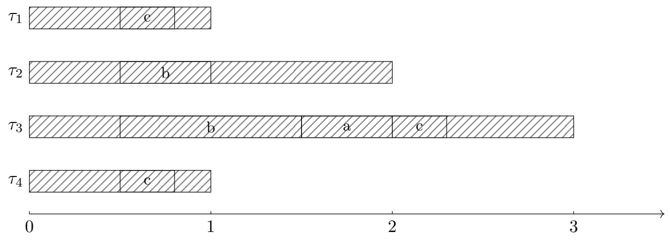


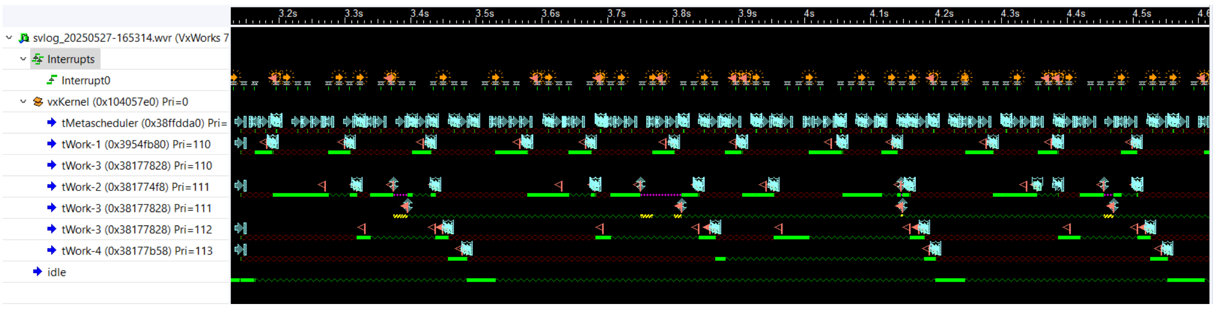
Figure 1: Task execution and shared resource usage. Legend: a = Ice Machine, b = Serving Counter, c = Pouring Pump

| Protocol | Schedulable | Max Blocking Time | Total Utilization |
|---|---|---|---|
| PIP | Yes | 1.0 ticks | 0.7056 |
| NPP | Yes | 3.0 ticks | 0.7056 |

Table 2: Feasibility Comparison of Access Protocols (RM Scheduling)

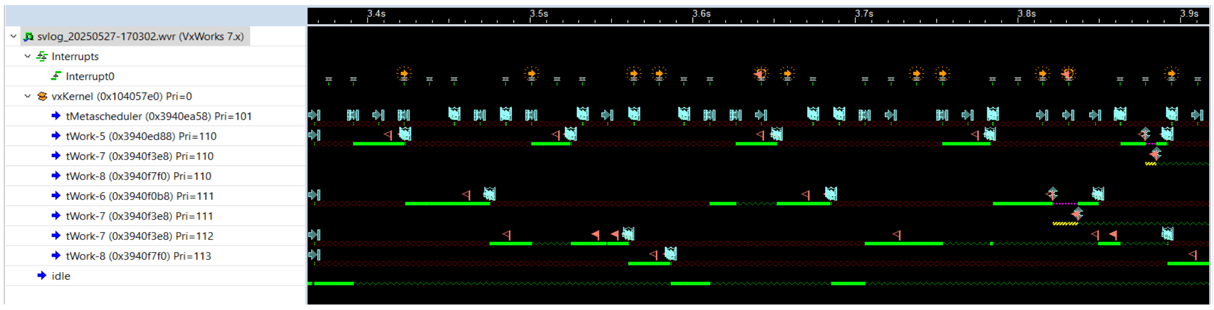
## 2. SystemViewer Analysis

We performed four simulations: two with **PIP** and two with **NPP** (bot in the synchronous and phased cases). The **NOP** and **NOR** cases were intentionally excluded from this analysis. Since they do not prevent concurrent access to shared resources, their inclusion would not provide meaningful information on schedulability or protocol effectiveness (for this type of analysis).



svlog_20250527-165314.wvr (VxWorks 7
Interrupts
Interrupt0
vxKernel (0x104057e0) Pri=0
tMetascheduler (0x38ffdda0) Pri=
tWork-1 (0x3954fb80) Pri=110
tWork-3 (0x38177828) Pri=110
tWork-2 (0x381774f8) Pri=111
tWork-3 (0x38177828) Pri=111
tWork-3 (0x38177828) Pri=112
tWork-4 (0x38177b58) Pri=113
idle
3.2s
3.3s
3.4s
3.5s
3.6s
3.7s
3.8s
3.9s
4s
4.1s
4.2s
4.3s
4.4s
4.5s

Figure 2: **PIP - Synchronous job Activation**



svlog_20250527-170302.wvr (VxWorks 7.x)
Interrupts
Interrupt0
vxKernel (0x104057e0) Pri=0
tMetascheduler (0x3940ea58) Pri=101
tWork-5 (0x3940ed88) Pri=110
tWork-7 (0x3940f3e8) Pri=110
tWork-8 (0x3940f7f0) Pri=110
tWork-6 (0x3940f0b8) Pri=111
tWork-7 (0x3940f3e8) Pri=111
tWork-7 (0x3940f3e8) Pri=112
tWork-8 (0x3940f7f0) Pri=113
idle
3.4s
3.5s
3.6s
3.7s
3.8s
3.9s

Figure 3: **PIP - Phased job Activation**

From the observation of the synchronized simulation (Figure 2), it is evident that the simultaneous start of tasks leads to a higher density of concurrent executions in the initial phase, with more contention over shared resources. However, the PIP protocol has a correcr behavior: lower-priority tasks accessing shared resources (e.g., $\tau 3$ and $\tau 4$) dynamically inherit the higher priority, allowing execution to proceed without deadline violations. The scheduling stabilizes after the first second, indicating that the system reaches a schedulable state. In the phased version (Figure 3), the introduction of initial phases reduces interferences: tasks are distributed through the timeline, minimizing overlap and initial blocking. In both cases, no big latencies or starvation phenomenon are observed, and the presence of idle windows confirms proper resource management. The visual analysis, is consistent with theoretical calculations, confirms that the system is fully schedulable with the PIP protocol, both in the synchronized and phased configurations.

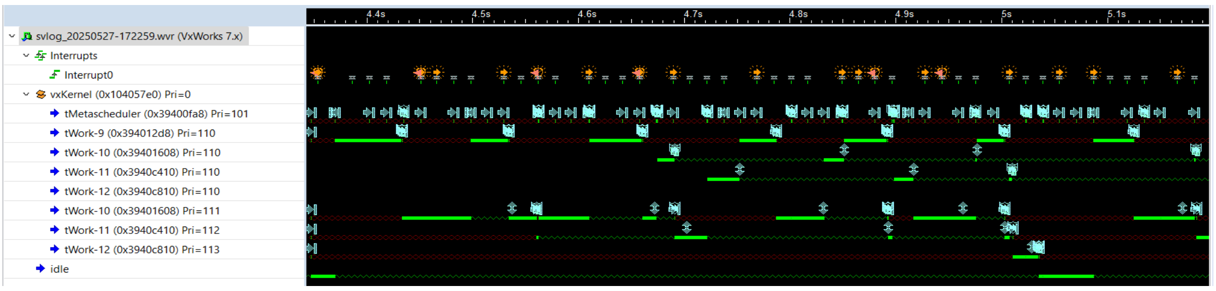

svlog_20250527-172259.wvr (VxWorks 7.x)
Interrupts
Interrupt0
vxKernel (0x104057e0) Pri=0
tMetascheduler (0x39400fa8) Pri=101
tWork-9 (0x394012d8) Pri=110
tWork-10 (0x39401608) Pri=110
tWork-11 (0x3940c410) Pri=110
tWork-12 (0x3940c810) Pri=110
tWork-10 (0x39401608) Pri=111
tWork-11 (0x3940c410) Pri=112
tWork-12 (0x3940c810) Pri=113
idle
4.4s
4.5s
4.6s
4.7s
4.8s
4.9s
5s
5.1s

Figure 4: **NPP - Synchronous job Activation**

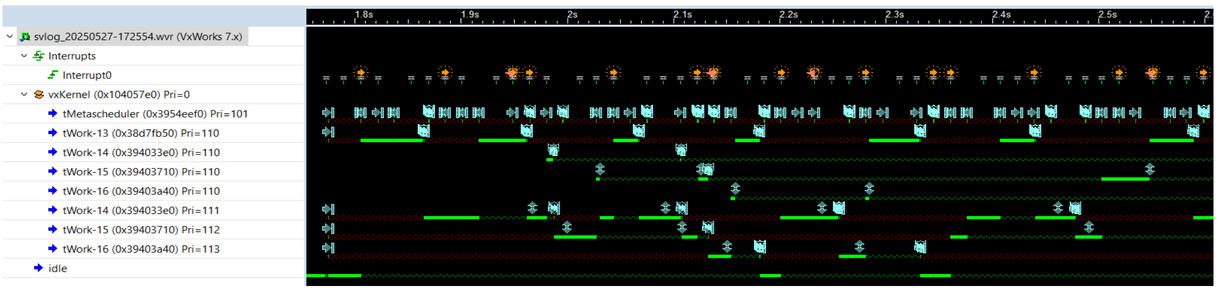

svlog_20250527-172554.wvr (VxWorks 7.x)
Interrupts
Interrupt0
vxKernel (0x104057e0) Pri=0
tMetascheduler (0x3954eef0) Pri=101
tWork-13 (0x38d7fb50) Pri=110
tWork-14 (0x394033e0) Pri=110
tWork-15 (0x39403710) Pri=110
tWork-16 (0x39403a40) Pri=110
tWork-14 (0x394033e0) Pri=111
tWork-15 (0x39403710) Pri=112
tWork-16 (0x39403a40) Pri=113
idle
1.8s
1.9s
2s
2.1s
2.2s
2.3s
2.4s
2.5s

Figure 5: **NPP - Phased job Activation**

In the synchronized configuration (Figure 4), all tasks begin execution simultaneously, and due to the non preemption of the protocol, lower priority tasks block higher priority tasks. These blocks are visible as green execution extended segments, even when a higher-priority task is ready. However, the system remains schedulable: there are no observable deadline violations, and idle slots appear (even if not regularly), indicating that tasks complete in time. In the phased version (Figure 5), due to the latest activations, execution is more uniformly distributed across the timeline, and the number of blocking situations is reduced. Also the idle slots apperas more regularly, meaning that the phased activation is a good solution. While the response times remain a little bit higher than PIP due to the non-preemptive blocking, the simulation confirms that the system is still schedulable.

## 3. Final observations on modified files.

We modified, for some small error or issues, almost all the files we get from the last lab lecture. We didn't modified `dummyTask.c`, `metascheduler.c` and `metascheduler.h`, but to be sure that all works well, we advice to use all the files we have uploaded in the submission. Anyway, the files where you can see more changese are `application.c` and `application.h`.