ECOLE POLYTECHNIQUE DE BRUXELLES

ELEC-H410

Université Libre de Bruxelles

École Polytechnique de Bruxelles

Pandemic Project

Authors of Group 13:
JANKE Nico (540076)
VAN Dyck Emile (540174)
WOJTACH Kacper (513025)

Professor:
QUITIN François

May 7, 2025

Contents

1 Introduction **3**

2 Tasks and Synchronization **3**
2.1 Quarantine . . . 3
2.2 Vaccine . . . 3
2.3 Medicine . . . 3
2.4 LCD . . . 3

3 Logic Analyzer Traces **4**
3.1 Startup and Periodic Execution . . . 4
3.1.1 Startup Phase . . . 4
3.1.2 Periodic Execution . . . 4
3.1.3 Aperiodic Contamination Events . . . 5
3.2 Timing Drift Due to Contaminations and Mutex Timeout Limit . . . 5

4 Conclusion **7**

1 Introduction

The goal of the pandemic game is to produce enough vaccines to eradicate the virus before the population collapses. We have an interface (`pandemic.h`) to communicate with the game routine.

- Every 3 seconds a vaccine clue is dropped.
- Every 5 seconds, the percentage of healthy people in the population is updated.
- On random occasions, an individual gets contaminated, and we need to quarantine the population in less than 10 milliseconds.

To manage these events and win the game, we created one task for each with different priorities. We also have a task dedicated to printing the state of the game on a LCD screen.

2 Tasks and Synchronization

2.1 Quarantine

This task has the highest priority of all our tasks. All it does is wait for the `QuarantineStart` binary semaphore to be available. This is triggered by contamination. When the semaphore is available, the task starts the quarantine. This is done on time because of the priority of the task.

2.2 Vaccine

The `vaccineProductionTask` has the third highest priority. It waits for a clue to be released and the `lab` semaphore to be freed to start producing a vaccine. That way we start producing as soon as we get a clue. This is needed because we have 3 seconds to produce a vaccine after release. If we need to wait on the lab for more than 475ms we decide to not start the vaccine production because we will not have the time to do it.
When the vaccine is produced, we ship it immediately. Once this is done, we give the `lcd` and `lab` semaphores. The first one is to notify the LCD task that the game state has changed, and the second one is to notify the Medicine task that it can produce or ship a medicine.

2.3 Medicine

In order to alternate between producing and shipping the medicine, the `medicineProductionTask` contains two consecutive sections, each beginning by waiting for the `lab` semaphore and ending by releasing it. The first one handles production, and the second one handles shipping. Since it has a lower priority, it will only run if `vaccineProductionTask` is waiting for the `vaccineStart` semaphore. This will be more visible in the logic analyzer traces.

2.4 LCD

The last task has the second highest priority. This is needed because if it is not the case the LCD will never update. It is notified by the vaccine and medicine tasks when the game state has changed. It will print the state of the game as specified in the specification.

3 Logic Analyzer Traces

3.1 Startup and Periodic Execution

Fig.1 shows the early behavior of the FreeRTOS-based pandemic simulation. The six digital signals (D0–D5) represent GPIO pins toggled by specific tasks:

- D0: `releaseContamination` + `quarantineTask`
- D1: `medicineProductionTask` : Production segment
- D2: `medicineProductionTask` : Shipping segment
- D3: `vaccineProductionTask`
- D4: `lcdTask`
- D5: `releaseClue`

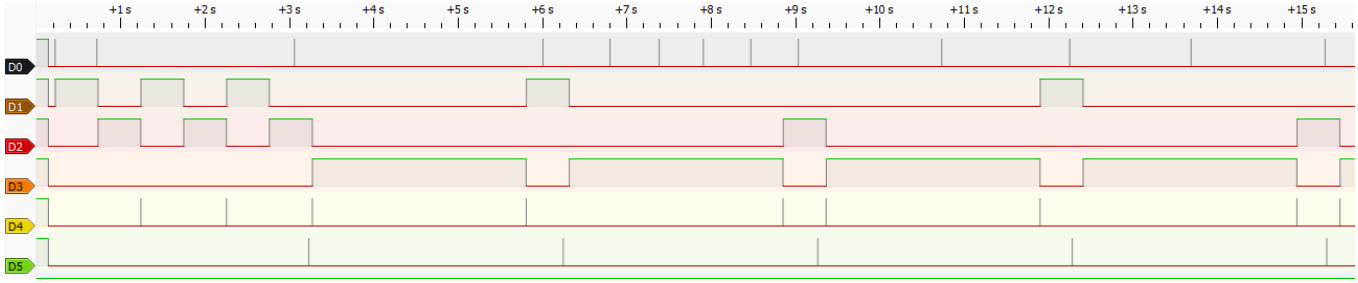

Figure 1: Early Behavior of the simulation

3.1.1 Startup Phase

At the beginning of the trace, all signals are low. This shows that the system starts in a clean and reset state.
Immediately after startup:

- D1 (Medicine Production) goes high, indicating that the medicine production task runs first
  - This matches the behavior in the code: `medicineProductionTask()` runs in a loop and starts as soon as it can get the lab mutex.
- Then D2 (Medicine Shipment) is activated, showing that the medicine is shipped.
- D4 (LCD Update) follows, confirming that the LCD is refreshed right after the medicine shipment — triggered by xSemaphoreGive(lcd) in the medicine task.

. There is no sign of D5 (Clue Release) or D3 (Vaccine Production) at the beginning, meaning the game task hasn't triggered a clue yet.

3.1.2 Periodic Execution

Later in the trace, a regular pattern begins:

- D5 – Clue Release: The game task releases a clue using `releaseClue()`.

- D3 – Vaccine Production: The vaccine task picks up the clue, produces a vaccine, and gives the LCD semaphore.
- D4 – LCD Update: The display is updated with the new vaccine, medicine, population counters.
- D1/D2 – Medicine Production/Shipment: If allowed, the medicine is produced again/After production, the medicine is shipped.
- D4 – LCD Update: Again, the LCD is refreshed.

3.1.3 Aperiodic Contamination Events

Around t = 0.7s, D0 goes high: a contamination is released and the quarantine task is executed.

- This does not follow a fixed interval, confirming that contamination is aperiodic, as controlled by game logic in the gameTask.

This full loop repeats roughly every 3 seconds, showing synchronized task execution using FreeRTOS semaphores and mutexes.

3.2 Timing Drift Due to Contaminations and Mutex Timeout Limit

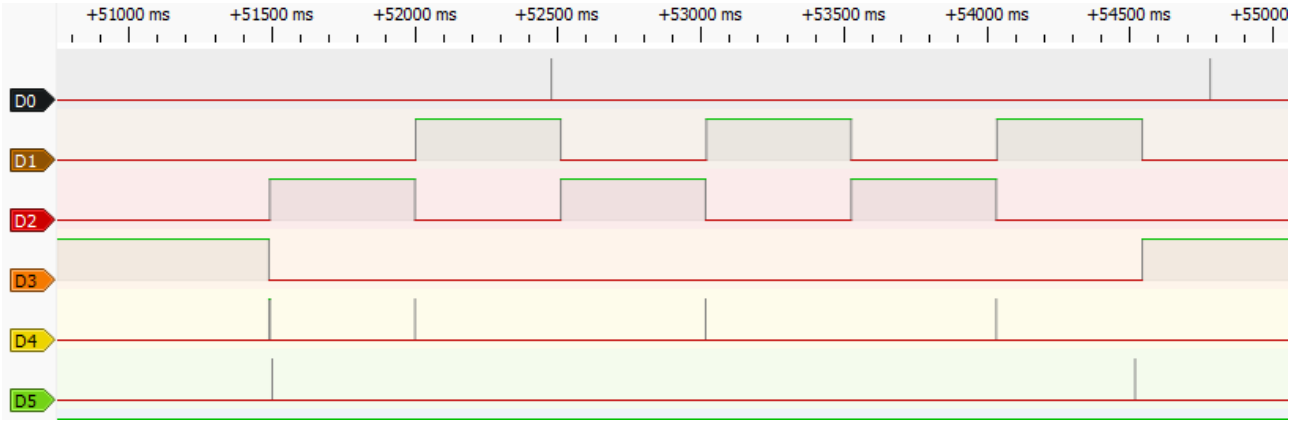

Figure 2: Reset of the "periodicity" during the simulation

In the observed trace (Fig.1), the system initially maintains a regular cycle of medicine production (D1), shipment (D2), clue release (D5), and vaccine production (D3). However, this periodicity is disrupted over time due to delays caused by the aperiodic contamination events (D0), the lcd task (D4) and the OS itself.
Each contamination triggers the quarantine task, which preempts other tasks due to its high priority. This causes delays in the execution of lower priority tasks, such as medicine production (D1), shipment (D2), vaccine production (D3), and LCD updates (D4). As this delay accumulates, the system eventually reaches a point where the clue is released more than 475 ms before the end of an ongoing shipment or medicine production.

Since `vaccineProductionTask()` waits for access to lab mutex (shared with the production and shipment of medications), it may miss its 475-ms time-out window. When this happens (see Fig.4), the vaccine task cannot start immediately after the clue release, breaking the original rhythm.
This resets the alignment between the drop of the clue and the start of the vaccine production. After that, the drift builds up again and causes another shift. This phenomenon is visible in the trace in Fig.2:

- In earlier cycles, the delay between D5 (clue) and D1 or D2 is lower than 475 ms (see Fig.3), so a vaccineProduction will follow.

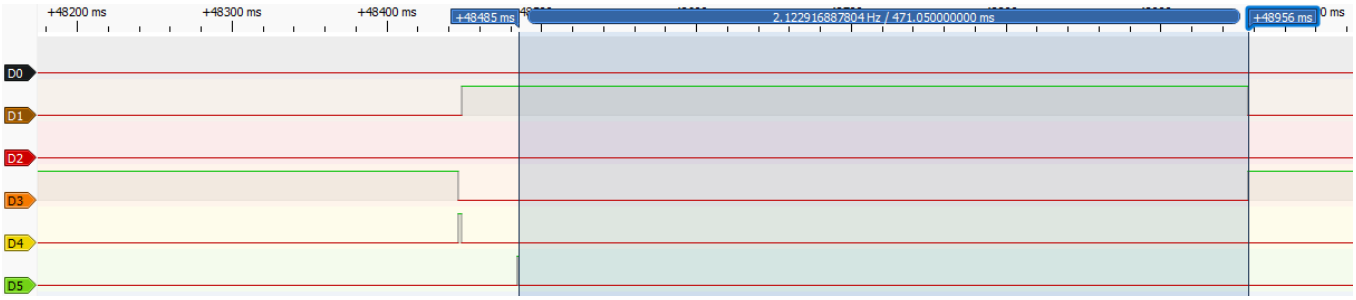

Figure 3: Time between the release of the clue and the end of the medicine production/shipment is lower than 475ms.

- At a critical point, this delay becomes greater than 475 ms(Fig.4), causing the vaccine task to miss the mutex and restart the timing cycle.(Fig.2)

Figure 4: An instance in which the duration between clue release and shipment completion exceeds 475 ms. Therefore, the vaccineProduction related to this clue is skipped.

4 Conclusion

At the end of the simulation after around 111 seconds, the system displays (Fig.5) a win and the three game variables.
These results confirm that our use of FreeRTOS tasks, priorities, and synchronization mechanisms allowed us to respond efficiently to both periodic and aperiodic events throughout the simulation.
The code can be found in our Github repo: https://github.com/kw0jtach/RTOS_Pandemic_Project

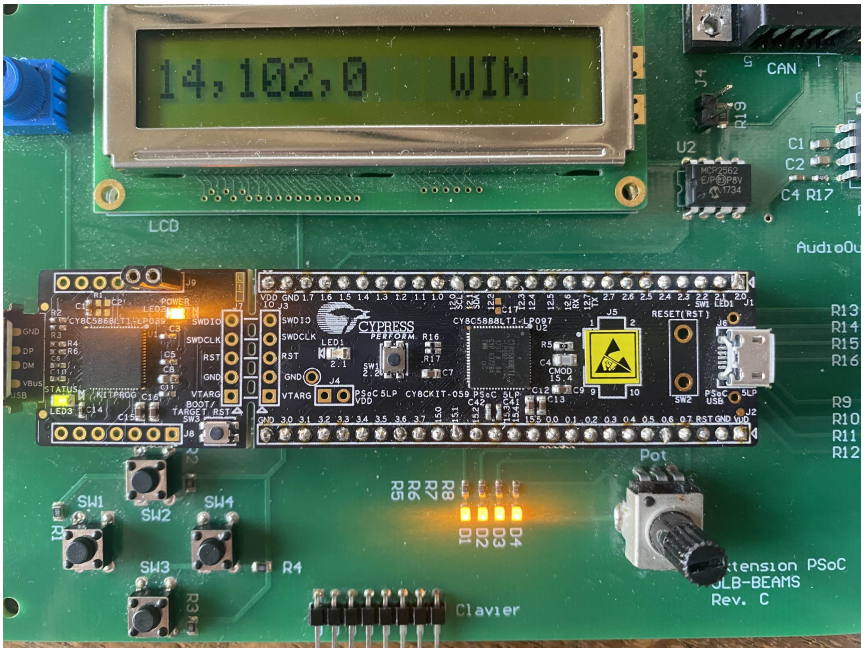

Figure 5: End Results