ULB UNIVERSITÉ LIBRE DE BRUXELLES

BRUFACE
BRUSSELS FACULTY OF ENGINEERING

VUB VRIJE UNIVERSITEIT BRUSSEL

[ELEC-H410] Real-time computer system

SAVING THE WORLD FROM THE PANDEMIC - GROUP 10

2020-2021

Pr. François Quitin, Ir. Youssef Agram

Engineering Sciences

Contents

1 Introduction

The aim of the project is to optimise a pandemic game in order to make the people survive using a PSOC. The game starts with 100 healthy people. If the healthy population falls to 0%, the game is considered lost. The vaccination on the other hand starts at 0%. The game is won when the vaccination reaches 100% and the remaining population is greater than 0%. Another parameter is the medicine pills, which prevent a person from being contaminated. To work on the medicine or vaccine production, a lab is available 24/7 to save humanity without any kind of rest but it should be noted that the lab can not generate medicines and vaccines simultaneously.

Concerning the events that happen in the game, they are of three types:

- Every 3s, the gameTask will drop a clue that is needed to create 3 vaccines. If the clue is not used within the 3 seconds, it expires.
- Every 5s, the disease spreads by contaminating 5 people. If medicine pills are available, the number of contaminated people is 5 minus the number of available medicine pills.
- Randomly, the gameTask contaminates an individual. When this event occurs, the whole population must be placed in quarantine within 10ms or 20% of the population gets immediately infected.

It is also asked to create a task that writes the healthy population, the number of vaccines and the number of pills on the LCD screen.

2 Function explanation

In this section, the main tasks and their dependency with the others are explained. The **main.c** code is shown in the appendix A.

2.1 Quarantine in response to a contamination

When the game randomly contaminates an individual, it is important to start a quarantine in less than 10 *ms* to prevent the contamination of 20% of the population.

The function *releaseContamination()* is randomly called by *gameTask()*. When this occurs, a global variable is updated with the current tick count and a binary semaphore is increased.

To respect the 10 *ms* deadline, the task *launchQuarantine()* has the second highest priority value (*gameTask()* having the highest). The task starts by taking the semaphore increased by *releaseContamination()* and checks that less than 10 *ms* have elapsed since the contamination happened. If the semaphore was successfully taken and the timing constraint is respected, the quarantine is started.

2.2 Production and shipment of vaccines or medicines

To produce and ship vaccines or medicines, two tasks were created: *makeAndShipVaccine()* and *makeAndShipMedicine()*. As the production of vaccines is the determining event to win the game, the priority of the former is greater than the priority of the latter. However, a solution had to be found to prevent the lab from building vaccines and medicines simultaneously (without protection, the production of a medicine could be interrupted by the production of a vaccine due to the priority difference). This is done via a mutex.

When the *gameTask* releases a clue, it is stored in a queue in order to give it to the *makeAndShipVaccine()* task.

2.2.1 Medicine production and shipment

The task starts by taking the *labAvailableMutex* to ensure that the lab is not busy producing vaccines at that it won't be interrupted while building the medicine. When the production is done, the *labAvailableMutex* is freed.

The next step of the task concerns the shipment of the medicine. It should be noted that for this step, it was considered that the shipment of a medicine could be interrupted by the creation and shipment of a vaccine as they are considered more important than the medicines.

2.2.2 Vaccine production and shipment

The task starts by receiving the clue from the queue. After reception of the clue, the *labAvailableMutex* is taken to ensure that the lab is not busy producing medicines at that it will not be interrupted while making the vaccine. At this point, the lab is ready to make a vaccine but it does not mean that a created vaccine will help to win the game.

For the vaccine to be valid, it must be generated and shipped within 3 seconds of the clue release time. Knowing that, for the vaccines, the build and shipment process takes 2.5 seconds, the task must start with a maximum delay of 500 ms after the clue has been released. Therefore, after the clue has been received and the mutex taken, the delay is checked. If it is inferior than 500 ms, the vaccine is built and immediately after its production, it is shipped (even if a medicine shipment is underway as the shipment of a vaccine is considered more important so, in this specific case, the medicine shipment process is interrupted and resumed once the vaccine shipment is completed). After the creation and shipment process has completed or, if the delay exceeded 500 ms, the *labAvailableMutex* is freed.

2.3 LCD update

The LCD is updated every two seconds by the task *updateLCD()*. The remaining population, the number of produced vaccines and the number of produced medicines are sequentially printed using a counting variable and a modulo. When the game ends, remaining population and the number of vaccines are printed simultaneously on the screen and then, the task is definitively suspended as there is no need to keep it running.

It should be noted that the priority of this task is greater than the priorities of both *makeAndShipVaccine()* and *makeAndShipMedicine()* even though updating the LCD screen is generally not a high priority event. This choice is justified by the fact that the two tasks cited above use almost 100% of CPU time which, if they had higher priorities than *updateLCD()*, would prevent the LCD screen from being periodically updated.

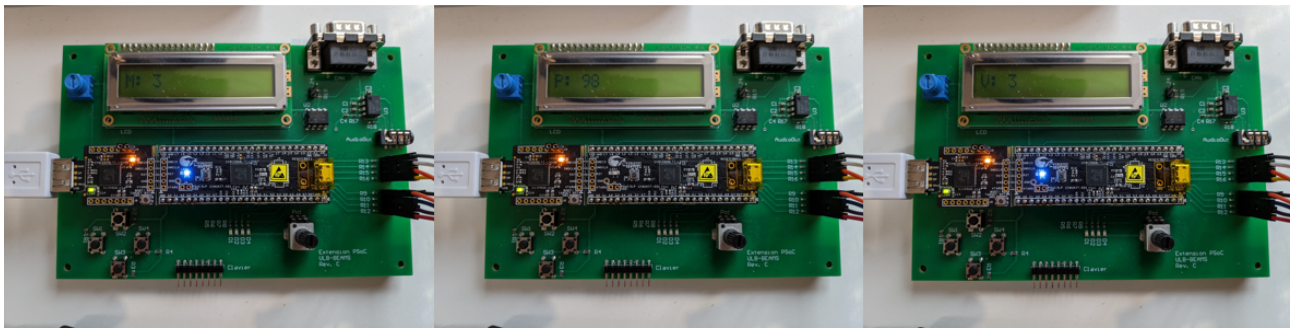

Figure 1: Sequential update of the LCD screen. Left: number of produced medicine. Centre: remaining population. Right: number of produced vaccines.

3 Results

The figure below shows a 30 second long sequence of the game in action.

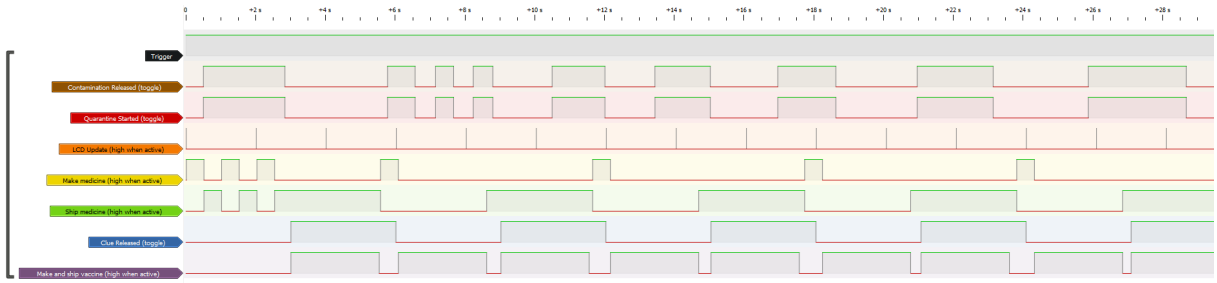

Figure 2: Game sequence (30 seconds).

The most interesting events shown on Figure 2 are detailed in the following subsections.

3.1 Quarantine in response to a contamination

The figures below show that when a contamination is released, the quarantine is started approximately 50 μs afterwards. This respects the maximum delay of 10 *ms* which prevents the infection of 20% of the population.

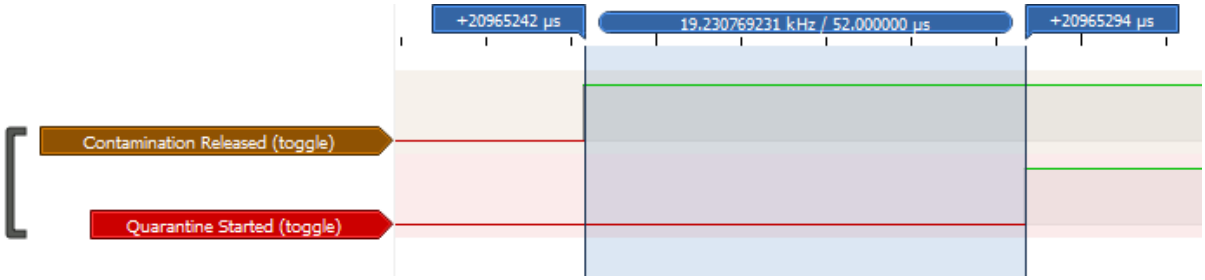

Figure 3: Quarantine in response to a contamination (1).

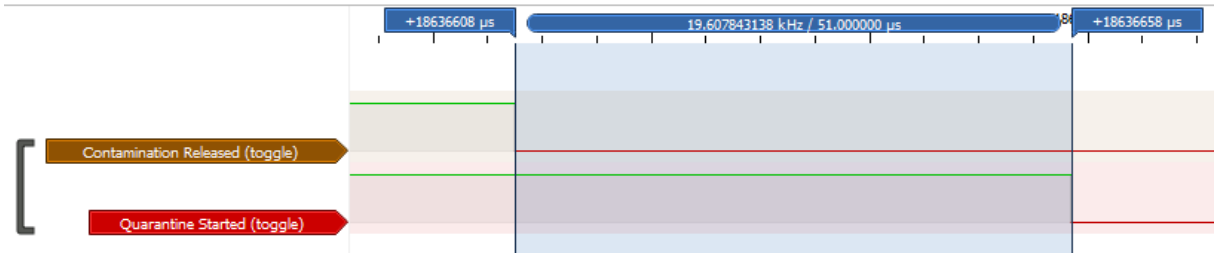

Figure 4: Quarantine in response to a contamination (2).

3.2 Production and shipment of vaccines or medicines

As explained in section 2.2, the laboratory creates and ships medicines or vaccines. The figures provided in this section help to understand how the tasks *makeAndShipMedicine()* and *makeAndShip-Vaccine()* work simultaneously.

The figures below show that when a clue is released, the maximum delay of 500 *ms* is respected which ensures that the vaccine is valid and helps to win the game.

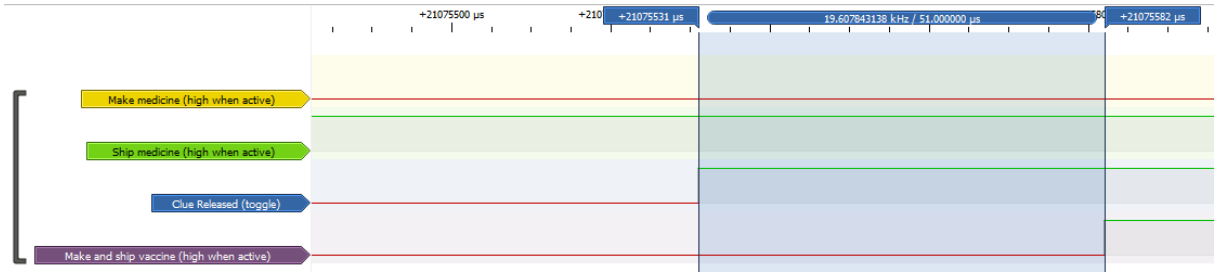

Figure 5: Delta time between the release of a clue and the making of a vaccine (1).

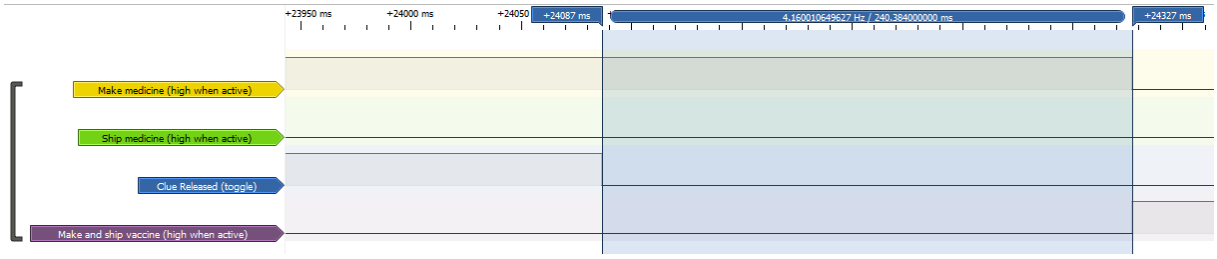

Figure 6: Delta time between the release of a clue and the making of a vaccine (2).

Figure 7 shows how the lab and the shipment process works for the vaccines and the medicines. We can notice two things: first, the lab is never interrupted (once in starts a task, it finishes it) and second, the shipment of a medicine is regularly interrupted by the building and shipment of a vaccine.

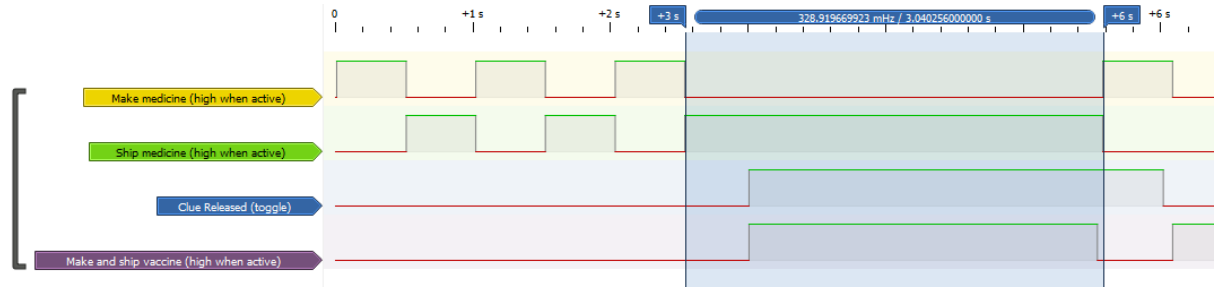

Figure 7: Creation and shipping process of the lab.

3.3 Number of survivors

After winning the game, we are left with 18 survivors as shown on Figure 8.

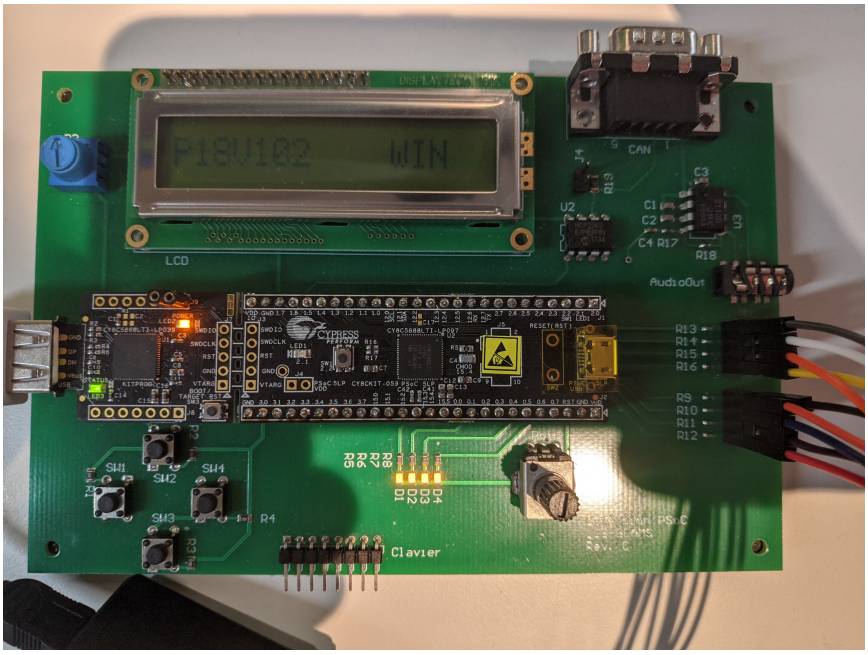

Figure 8: Number of survivors remaining at the end of the game.

4 Conclusion

In conclusion, after analysing the data acquired via the Logic Analyser, we noticed that the CPU utilisation is about 100% and we are left with 18 survivors. This project has allowed us to further improve our knowledge about Real-Time Computer Systems, Mutexes, Semaphores, Queues and task priorities.

Finally, we would like to express our gratitude to the assistant Youssef Agram and the professor François Quitin for their help and availability during the entire project despite the difficulties caused by the COVID-19 pandemic.

A Code

```c
/* =======================================
 *
 *              ELECH410 labs
 *          FreeRTOS pandemic project
 *
 *                   2020
 *
 * =======================================
*/
#include "project.h"

/* RTOS includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "timers.h"
#include "queue.h"
#include "semphr.h"

#include "pandemic.h"

#define TASK_STACK_SIZE (1024)

/* Task definitions */
#define GAME_TASK_NAME ("game_task")
#define GAME_PRIORITY (20)

#define LAUNCH_QUARANTINE_TASK_NAME ("launch quarantine")
#define LAUNCH_QUARANTINE_PRIORITY (19)
#define LAUNCH_QUARANTINE_TIMEOUT (10)

#define UPDATE_LCD_TASK_NAME ("update lcd")
#define UPDATE_LCD_PRIORITY (15)
#define LCD_UPDATE_PERIOD (2000)

#define MAKE_AND_SHIP_MEDICINE_TASK_NAME ("make and ship medicine")
#define MAKE_AND_SHIP_MEDICINE_PRIORITY (10)

#define MAKE_AND_SHIP_VACCINE_TASK_NAME ("make and ship vaccine")
#define MAKE_AND_SHIP_VACCINE_PRIORITY (13)

#define START_MAKING_VACCINE_TIMEOUT (500)

/* Task handlers */
TaskHandle_t gameHandler;
TaskHandle_t launchQuarantineHandler;
TaskHandle_t updateLCDHandler;
TaskHandle_t makeAndShipMedicineHandler;
TaskHandle_t makeAndShipVaccineHandler;

/* Task prototypes */
void launchQuarantine();
void updateLCD();
void makeAndShipMedicine();
void makeAndShipVaccine();

/* Mutexes, semaphores and queues */
SemaphoreHandle_t quarantineSem;
SemaphoreHandle_t labAvailableMutex;
```

```
59 xQueueHandle clueQueue;
60 const uint8_t clueQueueSize = 1;
61
62 /* Global variables */
63 uint32 releaseContaminationTime = 0;
64 uint32 clueReleaseTime = 0;
65
66 /*
67  * Installs the RTOS interrupt handlers.
68  */
69 static void freeRTOSInit( void );
70
71 int main(void)
72 {
73     /* Enable global interrupts. */
74     CyGlobalIntEnable;
75
76     freeRTOSInit();
77
78     /* Place your initialization/startup code here (e.g. MyInst_Start()) */
79     LCD_Start();
80     KB_Start();
81
82     // Toggle GPIOJ11 to trigger logic analyzer
83   GPIOJ11_Write(0u);
84   CyDelay(1);
85   GPIOJ11_Write(1u);
86
87     // Create tasks
88     xTaskCreate( gameTask, GAME_TASK_NAME, TASK_STACK_SIZE, NULL, GAME_PRIORITY,
       &gameHandler );
89     xTaskCreate( launchQuarantine, LAUNCH_QUARANTINE_TASK_NAME, TASK_STACK_SIZE,
       NULL, LAUNCH_QUARANTINE_PRIORITY, &launchQuarantineHandler );
90     xTaskCreate( updateLCD, UPDATE_LCD_TASK_NAME, TASK_STACK_SIZE, NULL,
     UPDATE_LCD_PRIORITY, &updateLCDHandler);
91     xTaskCreate( makeAndShipMedicine, MAKE_AND_SHIP_MEDICINE_TASK_NAME,
     TASK_STACK_SIZE, NULL, MAKE_AND_SHIP_MEDICINE_PRIORITY, &
     makeAndShipMedicineHandler);
92     xTaskCreate( makeAndShipVaccine, MAKE_AND_SHIP_VACCINE_TASK_NAME,
     TASK_STACK_SIZE, NULL, MAKE_AND_SHIP_VACCINE_PRIORITY, &
     makeAndShipVaccineHandler);
93
94     // Create mutexes, semaphores and queues
95     quarantineSem = xSemaphoreCreateBinary();
96     labAvailableMutex =  xSemaphoreCreateMutex();
97     clueQueue = xQueueCreate(clueQueueSize, sizeof(Token));
98
99     // Launch freeRTOS
100    vTaskStartScheduler();
101
102    for(;;)
103    {
104    }
105 }
106
107 void freeRTOSInit( void )
108 {
109    /* Port layer functions that need to be copied into the vector table. */
110    extern void xPortPendSVHandler( void );
111    extern void xPortSysTickHandler( void );
```

```
112     extern void vPortSVCHandler( void );
113     extern cyisraddress CyRamVectors[];
114 
115   /* Install the OS Interrupt Handlers. */
116   CyRamVectors[ 11 ] = ( cyisraddress ) vPortSVCHandler;
117   CyRamVectors[ 14 ] = ( cyisraddress ) xPortPendSVHandler;
118   CyRamVectors[ 15 ] = ( cyisraddress ) xPortSysTickHandler;
119 }
120 
121 /*
122  * When a contamination occurs gameTask calls this function.
123  *
124  */
125 void releaseContamination( void )
126 {
127     /*
128     * This function is called by gameTask() at random timing intervals.
129     * The binary semaphore (quarantineSem) is increased by one for
    synchronization purposes.
130     * The releaseContaminationTime is also updated to check whether the maximum
    delay between the contamination and the start of the quarantine is respected
    or not.
131     */
132     GPIOJ12_Write(~GPIOJ12_Read());
133     releaseContaminationTime = xTaskGetTickCount();
134     xSemaphoreGive(quarantineSem);
135 }
136 
137 /*
138  * When gameTask releases a vaccine clue it calls this function.
139  *
140  */
141 void releaseClue( Token clue )
142 {
143     /*
144     * This function is called by gameTask() when a clue is released.
145     * When this happens, the clue is added to a queue so it can be used by the
    makeAndShipVaccine() task.
146     * The clueReleaseTime is also updated to check whether the clue will expire
    before the end of the creation and shipment of the vaccine or not.
147     */
148     GPIOJ23_Write(~GPIOJ23_Read());
149     clueReleaseTime = xTaskGetTickCount();
150     xQueueSendToBack(clueQueue, &clue, 0);
151 }
152 
153 void launchQuarantine()
154 {
155     /*
156     * The semaphore (quarantineSem) is helpful for starting the task when a
    contamination has been released by gameTask().
157     * The delay between the release of the contamination and the current tick
    count is checked.
158     * If it does not exceed 10 ms, the population is placed in quarantine.
159     */
160     uint32 dt = 0;
161 
162     for(;;)
163     {
164         xSemaphoreTake(quarantineSem, portMAX_DELAY);
```

```
165         dt = xTaskGetTickCount() - releaseContaminationTime;
166         if ( dt < LAUNCH_QUARANTINE_TIMEOUT )
167         {
168             GPIOJ13_Write(~GPIOJ13_Read());
169             quarantine();
170         }
171     }
172 }
173
174 void updateLCD(void)
175 {
176     /*
177     * This task updates the LCD screen every 2 seconds.
178     * The remaining population, the vaccine progression and the medicine
    production are sequentially printed.
179     * At the end of the game, the final number of population and the final
    number of vaccines are printed and the task is suspended.
180     */
181     uint8_t LcdPrintCntr = 0;
182     uint32 taskStartTime = 0;
183
184     for(;;)
185     {
186         taskStartTime = xTaskGetTickCount();
187
188         GPIOJ14_Write(1u);
189         LCD_ClearDisplay();
190         LCD_Position(0,0);
191         if (LcdPrintCntr == 0)
192         {
193             LCD_PrintString("P: ");
194             LCD_PrintNumber(getPopulationCntr());
195         }
196         else if (LcdPrintCntr == 1)
197         {
198             LCD_PrintString("V: ");
199             LCD_PrintNumber(getVaccineCntr());
200         }
201         else
202         {
203             LCD_PrintString("M: ");
204             LCD_PrintNumber(getMedicineCntr());
205         }
206         LcdPrintCntr = (LcdPrintCntr + 1) % 3;
207         GPIOJ14_Write(0u);
208
209         if ( getPopulationCntr() <= 0 || getVaccineCntr() >= 100)
210         {
211             /* END OF GAME */
212             LCD_ClearDisplay();
213             LCD_Position(0, 0);
214             LCD_PrintString("P");
215             LCD_PrintNumber(getPopulationCntr());
216             LCD_PrintString("V");
217             LCD_PrintNumber(getVaccineCntr());
218
219             vTaskSuspend( NULL );
220         }
221
222         vTaskDelay(LCD_UPDATE_PERIOD - (xTaskGetTickCount() - taskStartTime));
```

```
223     }
224 }
225
226 void makeAndShipMedicine()
227 {
228     /*
229     * The task waits for the lab to be available.
230     * Once the lab is available, the task procudes a medicine pill.
231     * After the production of the medicine pill, the mutex is freed.
232     * Then, the medicine pill is shipped.
233     */
234     Token medicine;
235
236     for(;;){
237         xSemaphoreTake(labAvailableMutex, portMAX_DELAY);
238         GPIOJ21_Write(1u);
239         medicine = assignMissionToLab(0);
240         GPIOJ21_Write(0u);
241         xSemaphoreGive(labAvailableMutex);
242
243         GPIOJ22_Write(1u);
244         shipMedicine(medicine);
245         GPIOJ22_Write(0u);
246     }
247 }
248
249 void makeAndShipVaccine()
250 {
251     /*
252     * The task starts by receiving a clue from a queue which was added.
253     * When the clue has been received, the task waits for the lab to be
    available.
254     * Once the clue has been received and the lab is available, it checks the
    delta time between the clueReleaseTime and the current tick count.
255     * If the delta time is inferior than 500 ms, the vaccine will help win the
    game. Therefore, it is built and shipped.
256     * At the end of the task, the mutex is freed.
257     */
258     Token clue;
259     uint32 dt = 0;
260     Token vaccine;
261
262     for(;;){
263         xQueueReceive(clueQueue, &clue, portMAX_DELAY);
264         xSemaphoreTake(labAvailableMutex, portMAX_DELAY);
265         dt = xTaskGetTickCount() - clueReleaseTime;
266
267         if (dt < START_MAKING_VACCINE_TIMEOUT)
268         {
269             GPIOJ24_Write(1u);
270             vaccine = assignMissionToLab(clue);
271             shipVaccine(vaccine);
272             GPIOJ24_Write(0u);
273         }
274         xSemaphoreGive(labAvailableMutex);
275     }
276 }
277
278 /* [] END OF FILE */
```