Faculté des Sciences & Techniques — Université de Limoges

Master 1st year

Embedded Systems

Examination — january 2023

Duration : 1h30 — Documents allowed

1 – 5pts Let be the following program :

```
 1 static SemaphoreHandle_t lock;
 2 TickType_t cs_wait = 250;
 3 TickType_t med_wait = 5000;
 4
 5 void doTaskL(void *parameters) {
 6   TickType_t timestamp;
 7   while (1) {
 8     timestamp = xTaskGetTickCount() * portTICK_PERIOD_MS;
 9     xSemaphoreTake(lock, portMAX_DELAY);
10     timestamp = xTaskGetTickCount() * portTICK_PERIOD_MS;
11     while ( (xTaskGetTickCount() * portTICK_PERIOD_MS) - timestamp < cs_wait);
12     xSemaphoreGive(lock);
13     vTaskDelay(500 / portTICK_PERIOD_MS);
14   }
15 }
16 void doTaskM(void *parameters) {
17   TickType_t timestamp;
18   while (1) {
19     timestamp = xTaskGetTickCount() * portTICK_PERIOD_MS;
20     while ( (xTaskGetTickCount() * portTICK_PERIOD_MS) - timestamp < med_wait);
21     vTaskDelay(500 / portTICK_PERIOD_MS);
22   }
23 }
24 void doTaskH(void *parameters) {
25   TickType_t timestamp;
26   while (1) {
27     timestamp = xTaskGetTickCount() * portTICK_PERIOD_MS;
28     xSemaphoreTake(lock, portMAX_DELAY);
29     timestamp = xTaskGetTickCount() * portTICK_PERIOD_MS;
30     while ( (xTaskGetTickCount() * portTICK_PERIOD_MS) - timestamp < cs_wait);
31     xSemaphoreGive(lock);
32     vTaskDelay(500 / portTICK_PERIOD_MS);
33   }
34 }
35 void setup() {
36   lock = xSemaphoreCreateBinary();
37   xSemaphoreGive(lock);
38   xTaskCreate(doTaskL,"Task L",1024,NULL,1,NULL);
39   vTaskDelay(1 / portTICK_PERIOD_MS);
40   xTaskCreate(doTaskH,"Task H",1024,NULL,3,NULL);
41   xTaskCreate(doTaskM,"Task M",1024,NULL,2,NULL);
42   // Delete "setup and loop" task
43   vTaskDelete(NULL);
44 }
45 void loop() {
46 }
```

a. What is the purpose of line 37 ? *(1pt)*

b. What is the difference between a `vTaskDelay` and the use of a timestamp in a task ? *(1pt)*

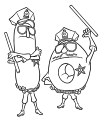
c. Describe how the different tasks will be executed ? *(2pt)*

d. Is it normal ? *(1pt)*

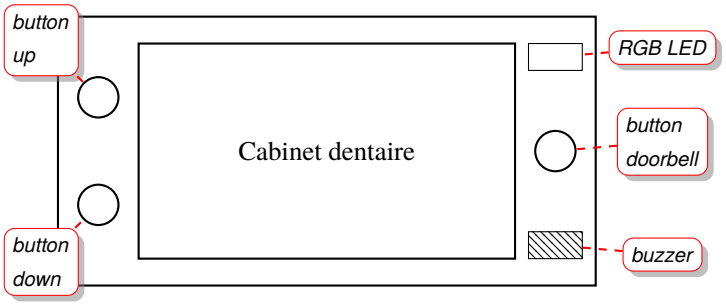
2 – 2pts

a. In assembly programming, are registers used differently in « *baremetal* » than in Linux and why ? *(1pt)*

b. Why in the linker, we make a difference between `lma` and `vma`,? *(1pt)*



3 – 13pts Let be the following building intercom :



It has :

- ☐ a 1-line display ;
- ☐ 3 buttons connected to PINs 3,4,5 ;
- ☐ an RGB LED connected to the PIN 7 ;
- ☐ a buzzer ;
- ☐ an opening relay on PIN 8.

The operation is as follows :

- ▷ when the intercom is not in use, the display is off ;
- ▷ as soon as you press one of the buttons « down » and « up », the display lights up and shows the names of the occupants of the building :
  - ◇ the name of the first occupant of the list of occupants is displayed ;
  - ◇ you press the « upper » and « lower » buttons to scroll through the names in the list ;
  - ◇ if we go too high, i.e. we already display the first name of the list, the RGB LED lights up in red and the buzzer goes off once during 1s ;
  - ◇ if we go too low, i.e. we already display the last name of the list, the RGB LED lights up in red and the buzzer goes off once during 1s ;
- ▷ when the doorbell is pressed :
  - ◇ if the display is off, nothing happens ;
  - ◇ or :
    - ⋆ we call the occupant whose name is displayed ;
    - ⋆ the buzzer is triggered in cycle 1s active, 500ms not active repeated 3 times ;
    - ⋆ the LED lights up green ;
    - ⋆ depending on the return value of the calling function :
      - ▷ if it is `true` the LED remains green and the door is opened (level 1 on the PIN associated with the relay) ;
      - ▷ if it is `false` we put the LED in red during 2s and we switch off the intercom.

The display can be used with the following functions :

`void afficheur_switch(bool etat);`	if `status=true` then the display is on, and off otherwise
`void afficheur_texte(char *texte);`	displays the `text` of up to 32 characters

The buzzer :

`void buzzer_bip(bool etat);`	if `status=true` the buzzer *beep*, otherwise it does nothing

The RGB LED :

`void LED_RGB(uint8_t rouge,uint8_t vert,uint8_t bleu);`	displays the requested value, the value (0,0,0) turns off the LED

The call to the selected occupant :

`bool appel_occupant(uint8_t numero);`	triggers a ring in the apartment identified by the rank in the list of names and returns a boolean if the occupant opens the door or not

The list of occupants is in an array of `char` :

Monsieur Durand	Madame Dupont	Cabinet dentiste	…

Where each entry corresponds to 32 characters filled with spaces.

a. Describe how the « up » and « down » buttons will be managed ? *(2pts)*
How will we make the link between the button, the occupant list management and the buzzer ?
What elements of FreeRTOS will you use ?
Is pressing both buttons simultaneously a problem? ?

b. How can we manage the different waiting times in an efficient way (LED, buzzer) ? *(1pt)*

c. How can we manage the variable and human waiting time of the call function ? *(1pt)*

d. Give the code performing the work of the intercom using the best features offered by FreeRTOS. *(7pts)*

e. We want to set up an anti-prankster system that requires you to press at least $2s$ on the button before initiating the call to the apartment concerned. *(2pts)*
How can you set up this system in your code ?