LEHRSTUHL FÜR RECHNERARCHITEKTUR
UND PARALLELE SYSTEME

TECHNISCHE UNIVERSITÄT MÜNCHEN

Semniar report

Sensor report

Contents

1 Software architecture **1**
1.1 Architecture 1
1.1.1 main.c/.h 1
1.1.2 mySetup.c 1
1.1.3 nvs.c 1
1.1.4 webFunctions.c 3
1.1.5 countingAlgo.c 3
1.2 Difficulties and Discarded Ideas 4

2 Algorithm Comparison **5**

Bibliography **6**

1 Software architecture

1.1 Architecture

1.1.1 main.c/.h

The `main.h` file contains important, global definitions. For example, if we want to send data to elastic search or not with `SEND_DATA`. It is possible with the `SEND_EVERY_EVENT` definition, to send every event to the database. However, this idea is discarded (for further details see section 1.2).
The `main.c` only calls `start_counting_algo()` after `my_setup()`, as the reader can see in figure 1.1.

1.1.2 mySetup.c

`my_setup()` is defined in the `mySetup.h`. This header file contains all definitions relevant for the whole program. For example, `SIZE_BUFFER` which is the size of the buffer, that buffers events registered by the barriers. In addition, it imports `main.h` such that every file has access to relevant information. The programmer preferred a central file of definitions instead of distribution over all header files.
As an example, one interesting global function is `error_message()`. This function sends adds a string to an error-array in the IoT-platform.
`mySetup.c` contains the implementation for this definition. A remark, this implementation needed a function that replaces spaces with underscores because of the HTTP GET message. Further, `mySetup.c` initializes following: the TaskHandles, that are needed in the test mode; the LCD display; NVS storage; SNTP; MQTT; and the GPIO pins. The programmer placed the update function for the external display `displayCountPreTime()` in `mySetup.c` to make it possible to update the external display everywhere in the project.

1.1.3 nvs.c

The NVS is used to realize sending the count every 5 minutes to the elastic search database. Every key in the NVS can cover 4000 Byte (= 4000 characters). However, for storing strings it can occur the ESP is not able to copy the string to the next pages to a git issue [1], which causes a `ESP_ERR_NVS_NOT_ENOUGH_SPACE`. That is why, the programmer choose `sizeBuffer = 1500` Bytes for each of the `NUM_KEY_WORDS 7` many keys. Tests showed that no page errors occur with this size. Additional tests showed that about 25 count-events can be stored in the NVS. Before going into details, the author points out that the NVS might be a reason for the bad performance of the first counting algorithm (for further details see section 1.2).

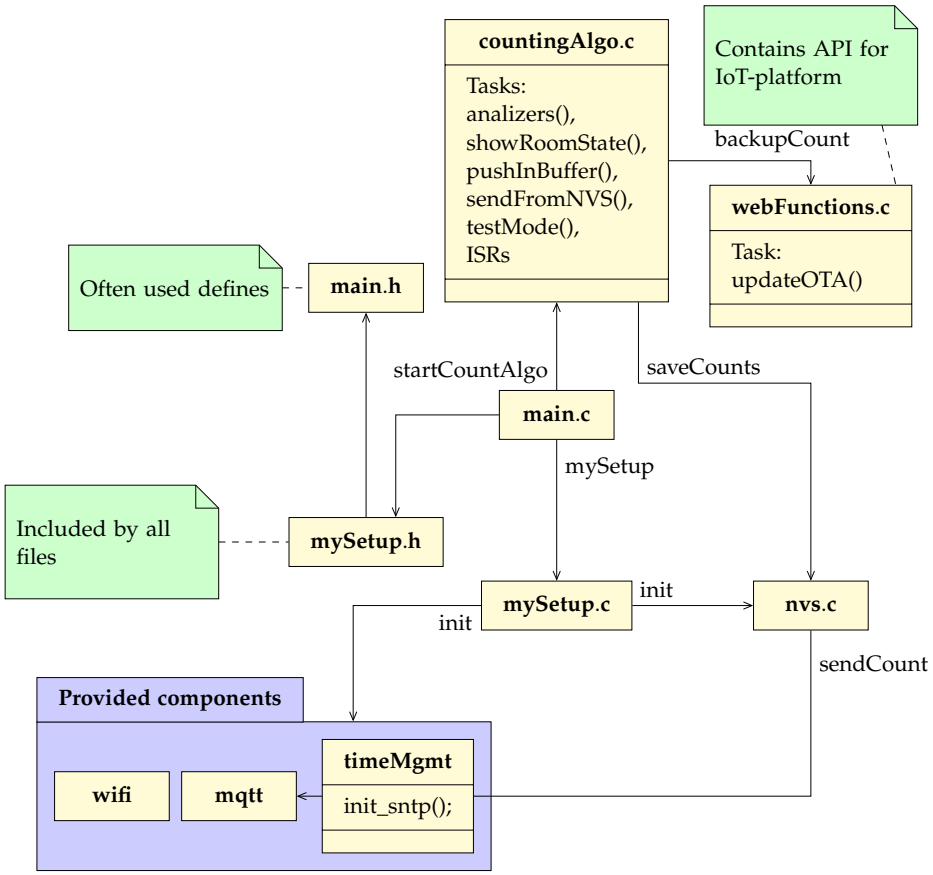

Figure 1.1: An overview of the software architecture for the counting algorithm on the ESP32.

In detail, the programmer used the cJSON library. This is used in `initNVS_json()`, which creates a new json file with the required schema for the current key `keyWords[nvs_index]`. Here, `keyWords[]` contains all 7 keys and `nvs_index` is the current, selected key with free space. It will be increased if the current key is full. Because this function is called with an open or closed NVS handle, there is the option to open the NVS handle.
This option occurs in `sendDataFromJSON_toDB()` too, which is periodically called every 5 minutes. This function deletes every key in the NVS until `nvs_index`, after sending its content with MQTT to the database. In the end, it creates a json file with `initNVS_json()` at `nvs_index = 0`.
The most important function `writeToNVM(sensorName,peopleCount,state,time)` appends to the array `sensorName`, a json dictionary with `peopleCount` at time `time`. `State` was used in a discarded idea. Before `addEventToStorage()` stores the event to the NVS, there are checks whether we can store data. For example if we expand the limit of `NUM_KEY_WORDS 7`, the ESP sends the data to MQTT. Otherwise, it creates a new index with `initNVS_json()`.
All these functions use heap allocation.

1.1.4 webFunctions.c

One can see in figure 1.1 that this files spawns one of 5 processes. `updateOTA()`'s function is self describing. In addition, it contains a memory leak detection which is necessary, but it never got activated. And there is the option to restart the device over the IoT-platform. This feature was never used due to a discarded idea (sending the log via the network with [2]).
More over, this file contains the API for the IoT-platform with a function: to fetch number with a given key, to fetch the count's backup, and to send a system report. The programmer saves an array as a string (with contains the system reports) such that there are converting operation with the cJSON library while adding a new string to that array.

1.1.5 countingAlgo.c

The heart of the program is the counting algorithm which is started in the `main.c`. We start with the local variables, that are only accessible in `countingAlgo.c`. A `QueueHandle` is used to get sequentially the sensor's events in the right temporal order. `SemaphoreHandles` ensure read and write operations for multiple tasks for the count variable, the buffer, and whether the test mode button was pressed. The buffer has many flags and variables, the author mentions later. Next to these are the `Barrier_data` that is a struct containing the data for one sensor/barrier event. Of course, `count`, `prediction`, and `state_counter` (state of the state machine) are local variables too. The ISRs have debouncing so that there are timestamps for each ISR. The task `pushInBuffer()` waits for an event in the `QueueHandle`, and it pushes it in the buffer. For this action we need the Semaphore to block the buffer to avoid inconsistencies and to increase `fillsize`. This task and the task `anlyzer()` share the same high priority. Before executing the state machine, `checkBuffer()` The `anlayzer()` uses a state machine (see figure 1.2). A previous counting approach can be found in section 1.2. The programmer found out that if the state is even an OUT event increases and an IN event decreases the

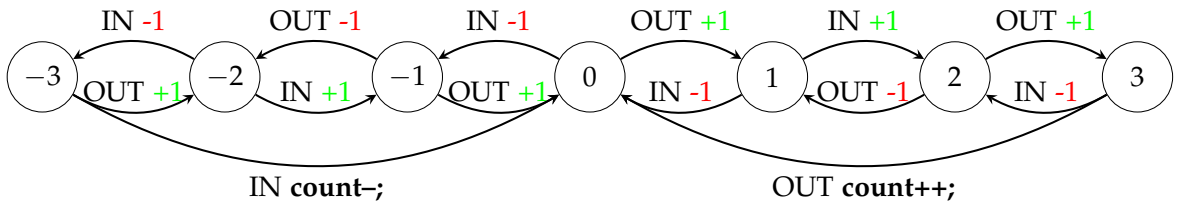

Figure 1.2: The state machine. The numbers in the circles represent the current state of the state machine. The colored number is the operation for the state, the bold text is the operation for the count of people. “IN” stands for an event for the indoor sensor and “OUT” for an event of the outdoor barrier.

current state. Visa vies for an odd state. This reduced the code

1.2 Difficulties and Discarded Ideas

timer, extern c-diffculties, backup->nvs->mqtt now directly to IoT-platform

2 Algorithm Comparison

Bibliography

[1] *Git Hub Issue about NVS pages*. `https://github.com/espressif/esp-idf/issues/5977`. [Online; accessed 3-July-2023]. 2023.

[2] *Git Hub repo for sending the log over the network*. `https://github.com/nopnop2002/esp-idf-net-logging`. [Online; accessed 4-July-2023]. 2023.