Embedded and Real-Time Systems

Avramidis Panagiotis

March 6, 2022

Assignment 2

Personal Assignment 2 ERTS

Requirements The assignment concerns the development of a COVID-19 contact tracing application. The intention is for the student to practice on real-time programming using POSIX pthreads. The program "scans" for nearby devices (i.e. MAC addresses) every 10 seconds using a fake bluetooth routine. They are kept for 20 minutes before being discarded. If an address is found in between 4 and 20 minutes, it is added in the close contacts queue. The close contacts are kept for 14 days. Every four hours we conduct a COVID test. In the case of a positive test the close contacts are immediately "uploaded" to a fictional server.

Program structure The application creates 4 threads in total. Each thread is tasked with satisfying some timing requirements. The threads work concurrently and synchronize using mutex and condition variables. A single structure `global_object` is being passed to the threads in order to share access. The program stores relevant timestamps and MAC addresses for further analysis with a python script. A Makefile is used to automated the building process and automatically upload to the remote target when cross-compiling.

1. Bluetooth search worker The first thread created is the `bt_search_worker`. This thread is responsible with performing the scanning every 10 seconds and subsequently adding the the MAC found in the `MacSearches` queue.

2. Close contacts worker Every time a new MAC is found, this thread wakes up and checks wether to add the MAC to the `CloseContacts` queue.

3. Close contacts timer Here, the queue holding the close contacts is being accessed is ascending order. That is, the oldest contact is accesses first by querying the head. The thread routine reads the scan time, and computes the time the contact should be removed (14 days). Then it sleeps for until the time is reached. The contact is removed and the cycle repeats again. As long as the queue is empty, the thread wait on the condition variable `CloseContacts->notEmpty`. The code was adapted from the prod-cons.c by Andrae Muys.

4. Covid testing thread Every 4 hours a COVID test is performed. The `testCOVID` routine randomly returns a boolean value that signifies the test result. If the test is positive, the `uploadContacts` routine is called immediately, which "uploads" the contacts to a fictional server. In reality, the MAC addresses along with the upload timestamps are stored in the binary files "uploaded_contacts_counter.bin" and "uploaded_contacts_counter.bin"

Testing routine

log data In order to check the program behavior and verify it, a python script was employed. For that purpose the binary data emitted by the application are used. The files are stored in the 4 .bin files: `./bt_searches.bin ./close_contact`
As the names suggest, the first two files used for storing every scanned MAC and every MAC added to the close contacts respectively. The file`uploaded_contacts.bin` file hold every batch of uploaded MACs, while the file `uploaded_contacts_counter.bin` stores information of the number of uploaded MAC contacts and the corresponding timestamp of the event. The two files are used together to obtain the list of MAC batches with the python script.

Processing the data The first plots the timeseries of the scanned MACs' error in us (microseconds, deviation from the 10 second target). In addition to the two graph, statistics are computed for the two timeseries using numpy routines. (minimum, maximum, median and standard deviation).

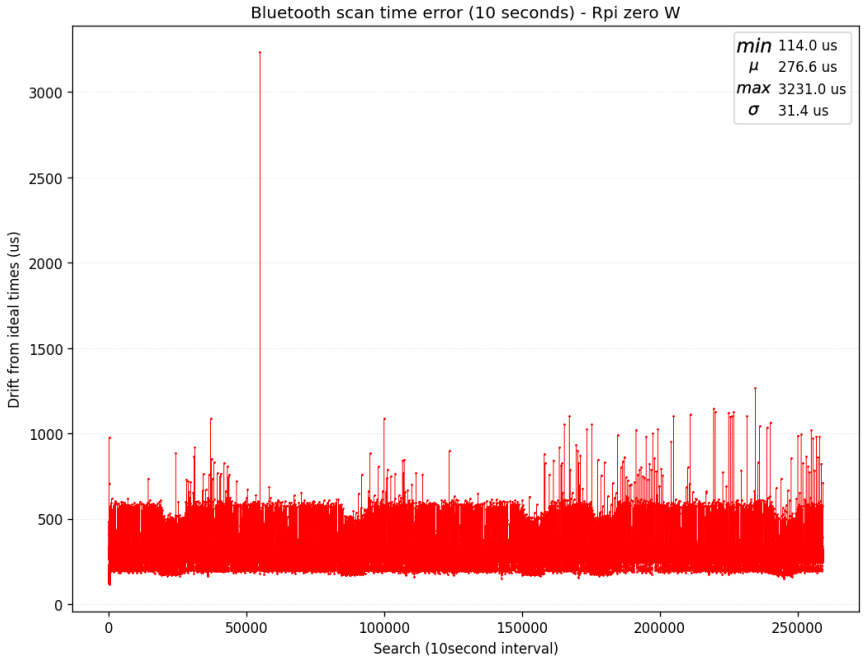

Figure 1: Scan time error

Setup The os used is the ESPX image provided by the course instructor. The connection goes through usb.

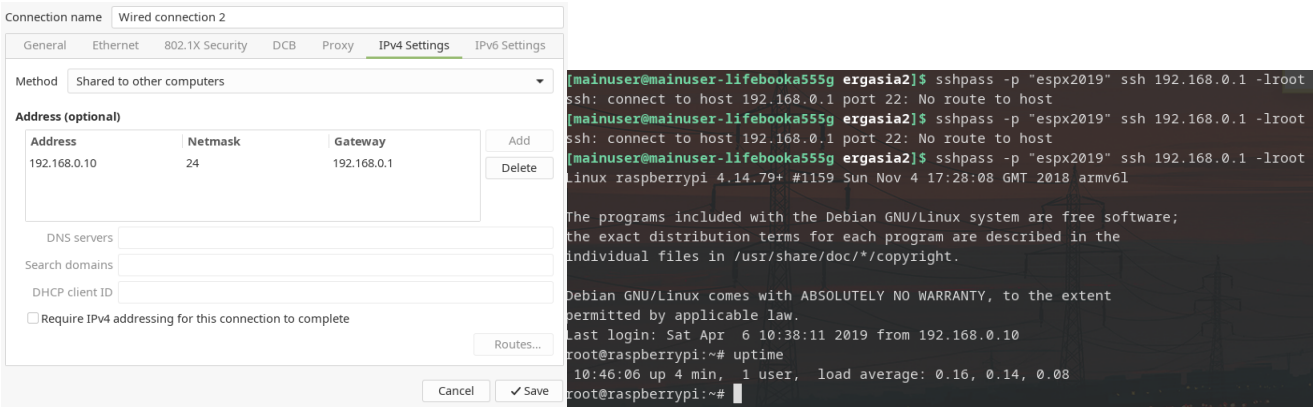

Figure 2: Network settings

Bonus: CPU usage Using the command `nohup ./covidTrace_ARM32 & disown` we disassociate the daemon with the shell so that it doesn't get terminated on ssh exit. The ps linux command is very powerful, and it was used in order to monitor CPU and memory: `watch -n 1 ps -C covidTrace_ARM32 -o %cpu,%mem,cmd` The watch is commonly used to continuously apply a single command and watch the output for changes. Here we monitor every 1 second.

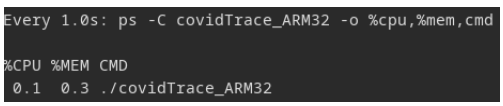

Figure 3: CPU - Memory monitoring

As can be seen, the CPU usage is fairly low!

Conclusion With a mean delay of 276 us, the application satisfies the requirements (soft real-time), although there are some latency spikes.

References

[1] https://github.com/grpan/ERTS-AUTH