Problem solutions 2

Alberto Defendi - alberto.defendi@helsinki.fi

February 14, 2022

1 Solutions

Exercise 1:

We consider the analysis of a self-driving car of autonomy level-1, where the driver is assisted with an assistance system for steering or acceleration/deceleration [Int14].

1. **Event based** Level-1 requires intervention if the autonomous system fails to respond properly, so it must be possible to take control of the driving process by acting on the speed level if the system fails to adjust the car to a suggested speed. This is a sensible aspect of the system, since in a possible failure, the car could reach speeds that may make it difficult to control, or may break the law.
   Reading data from the sensors and evaluate car's condition can be done using event-based, since it would increase the accuracy of the measurements' timing.

   Thread-based Check of data from GPS, street maps and police database should run asynchronously without the intervention of the pilot, because the system may consult internet or access the main memory to check the data, which can be a slow operation.

2. Preemptive scheduling, because we can have processes of higher priority, like the action of the pilot on the level, that would switch from the assisted driving thread, to the manual. This would require a more complex scheduler, but this is balanced by the fact that the car is equipped with a powerful computer that is also used for infotainment.

3. In this car we can have a on-board computer that acts a a gateway, manages infotainment and does computation from sensor's data, and multiple low-end devices that collect data from sensors and do small

computations to check if the measurements are in the normal range. The computational power of this device may be similar to a high-range ARM-based tablet that is power-efficient in fall 2021. The gateway OS should be:

- Multi-threading.
- Offer a GUI compatibility.
- Support drivers for interacting with internal peripherals (e.g touchscreen) interacting with external peripherals (low-end devices) and radio/WiFi connectivity.
- Security features to avoid the auto to get hacked.
- A suitable OS could be Android.
- Dynamic memory, since there can be many application. The fragmentation is not problematic, since the main memory is a fast flash memory.

The self-driving module OS should support:

- Real time constraint (see next point).
- Events.

4. **Hard** If the driver cannot take control of the car, it results in a system failure and the car should stop gracefully, without becoming dangerous for other street users. Also if the sensors that collect data from the car devices fail,

   Firm If the autonomous system fails to collect speed data to assist the speed change, the driver should be informed and the system should become fully manual.

   Soft A possible soft constraint in the car, but not related to the driving experience, can be the live audio streaming of the infotainment, which down-samples to a lower bit-rate if there is low signal or if the system is at computational capacity level.

5. **Always on** The car has the advantage of having a discrete amount of energy autonomy, which can be used to power computers. Elements that are involved in the car driving must always be kept on to avoid incidents. Also the car condition sensors should continuously monitor possible issues.

   Core sleep The devices core can be splitted in high-power and low-power core, the firsts used where the demand of computations is higher, the seconds where is lower.

Screen dim The infotainment screen can be dimmed to reduce energy consumption.

Exercise 2:

1.1 Consumer: Smart body scale

(a) Weights people body mass and provides data analysis trough a mobile application. This helps the user to keep track of the weekly/monthly average and of any weight change.

(b) **Peripherals** The device should support peripherals such as a weight sensor, a LCD, a set of buttons and a WiFi antenna.

Connection The data is visualized trough a display, and is possible to select the current user from a set of buttons on the device. The device can upload measurements on a centralized trough the WiFi connection, but if this results to be too expensive to produce, the scale can connect to a phone trough Bluetooth and use the phone as a gateway.

Deep sleep The device operation is fully event-driven, since all the operations are executed during the usage of the balance, and when not used, the device goes to deep sleep. An example flow could be: the user selects its profile, steps on the balance and waits until the result, the data is sent to the server and the device goes to sleep.

Non-preemptive Since the operations can happen sequentially, there is no need to have higher priority processes, as long as the scale firsts measures the person, and then sends the data (which may take longer).

Event-based The scale could wake up when a change on the weight sensor is detected. After that, the data is visualized and sent.

Static allocation The allocation can be static since the data of the program is the measurement of the person, that is later sent and deleted.

(c) Tiny OS would be the best choice in this situation where the scale integrates a low-end device with little power (such as an array of 3/4 AA batteries), little memory and a limited number of actions.

1.2 Enterprise: Room management

(a) We develop an IoT system to offer a smart monitor to view and track the availability of a room, showing the event calendar and the possibility to reserve a room. A possible application could be adding one of these device outside a classroom at the University of Helsinki, where the daily lecture calendar is replaced manually everyday. The final system, will be integrated with the already existent info-system and will make possible to view the lessons calendar in that room and add the possibility to reserve a room if needed trough an online portal hosted by a gateway.

(b) **Peripherals** Should support drivers for epaper display and an Ethernet connection.

Price The device has to be low-end, since the computations are minimal and simple.

Low power The control unit energy requirements can be reduced at minimum, since the operations are infrequent over time (the device syncs only when the gateway (a server) updates the timetable). It would be efficient to have the device in deep-sleep when no events are happening, since if we are using a e-paper display, it can hold static text and images indefinitely without electricity. The on trigger can be done when any signal is received from the gateway from the LAN cable.

Memory Small memory is needed, enough to host the system and to temporarily download the data before printing it on the display. It is important that the program deletes the data, to avoid to run out of memory.

Non-preemptive Since the limited power, some tasks can take long, it is important to give time to each task to finish. This design is easier to implement and requires a simpler mechanism to context-switch.

Static allocation The data size is limited to what can fit on a display, and the system does only the simple task of fetching data.

Security The device must avoid the connection with unwanted host, because it could be used to breach into the network.

(c) Tiny OS would be perfect, since the device is low-end, receive the power trough the LAN cable and should do only simple operations.

Exercise 3:

1.3 RIOT OS

The OS of my choice is RIOT [Jav+18], specifically designed for IoT devices and lightweight in size, without sacrificing features like multi-threading.

Architecture Unlike Tiny OS which is monolithic, RIOT implements a micro-kernel design, which supports multi-threading and networking communications. The kernel supports real-time preemptive tickless scheduling, where the scheduler is run only when a thread is unlocked (saving additional power).
The kernel architecture is adapted from Fire kernel, which provides primary features such as scheduling, process communication and synchronization.

Advantages: 6LoWPAN The OS supports 6LoWPAN, which enables low-power radio communication between multiple devices.

Low-memory usage Thanks to its micro-kernel, the OS has a low impact on the memory usage, using the size of just 1.5KB RAM and 5KB of ROM.

Developer friendly Implements multi-threading, is effective in memory, in terms of APIs (partial POSIX compliance) and is robust against bugs.

POSIX The partial POSIX compliance makes possible to have an uniform API that is independent of underlying hardware, this makes possible to use RIOT with boards using different types of instruction sets. The documentation says that RIOT supports 100+ boards [Rio21].

Micro-kernel This kernel architecture can help reduce the kernel size in small OSs like RIOT.

Linear time fast execution As reported by the article, the task execution has an efficiency of $O(1)$.

Open source and relatively small code-base, thanks to the micro-kernel architecture, making it easier to contribute for aspiring OS developers, compared to monolithic OS.

Limitations: Language support Due to its tiny size, the OS SDK only supports the languages it was programmed in, which are C and C++. Despite those language being popular nowadays, new appealing alternatives for low-level systems are growing, such as GO and Rust, which offer low-level access and can be easier to use.

POSIX At the moment of writing, the OS only partially supports POSIX standard API, which can be problematic for some application and may require maintenance problems in future.

Micro-kernel The micro-kernel architecture can have security

No concurrency

No security features

No database

No documentation

Applications RIOT OS is applicable anywhere where low-end devices are used (where the memory, battery and power of devices is a constraint). The OS can be used in real-time applications.

References

[Int14] SAE International. *Summary of SAE International's Levels of Driving Automation for On-Road Vehicle.* 2014. URL: `https://web.archive.org/web/20180701034327/https://cdn.oemoffhighway.com/files/base/acbm/ooh/document/2016/03/automated_driving.pdf`.

[Jav+18] Farhana Javed et al. "Internet of Things (IoT) Operating Systems Support, Networking Technologies, Applications, and Challenges: A Comparative Review". In: *IEEE Communications Surveys Tutorials* 20.3 (2018), pp. 2062–2100. DOI: `10.1109/COMST.2018.2817685`.

[Rio21] RIOT Riot-Os. *Riot-os/riot: Riot - the friendly OS for IOT.* 2021. URL: `https://github.com/RIOT-OS/RIOT`.