Project 1: Real-Time Services on Linux GPOS

Introduction

The aim of this project is to apply the Linux Real-Time Services and the Real-Time Model to the the development of a real-life inspired real-time application.

Following the typical structure of embedded software, the project encompasses a set of cooperating tasks, involving synchronization, shared resources, access to a real-time database, etc.

Preparation

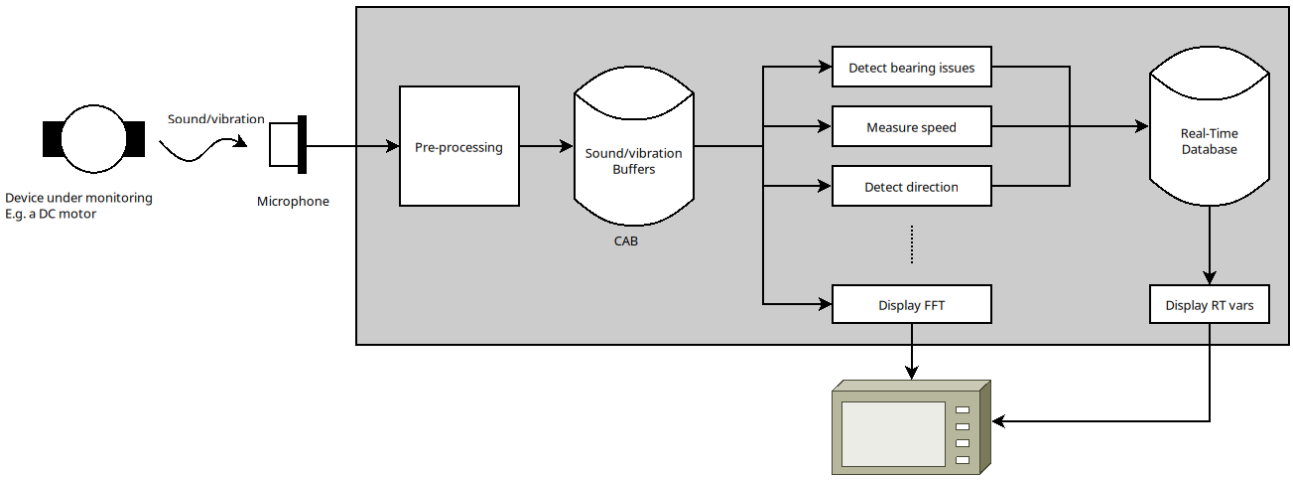
Students should first get comfortable with Linux real-time services addressed in the lab classes, including implementation of tasks as Linux threads, real-time scheduling policies, synchronization primitives, access to shared resources, etc.

Furthermore the students are required to acquire some domain-specific knowledge, in the case related to sound acquisition and processing. The source code provided includes information and links that should be studied in detail.

General description

The application to be implemented is representative of a real-time industrial monitoring system, with substantial simplifications made to make it compatible with the effort set for the project: two students per group, 6 ECTS, duration of three weeks.

The system architecture is as follows:



Sound is captured by a PC (internal sound-card, webcam, ...) by a dedicated sound capture task. The activation of this task is autonomous (sporadic task), as it originates from the sound recording hardware. After some pre-processing (e.g. low-pass filtering) sound samples are placed in a buffer (CAB type) to be processed by dedicated tasks.

These dedicated tasks perform, as the name implies, specific functions. Examples include:

- Detection of direction of movement
- Detection of structural issues, such as bearing faults
- Detection of speed
- Others

The actual algorithms for detecting these features can be quite complex, and are out of the scope of this course. So students can take simple (or even simplistic) approaches such as:

- Speed corresponds to the more powerful frequency and the motor is supposed to work in the range 2 kHz to 5 kHz
- Bearing issues manifest as low-frequency (e.g. below 200 Hz) with an amplitude greater than 20% of the amplitude of the peak frequency (that should correspond to the speed)
- etc.

Each task should detect one particular feature/issue and have reasonable timeliness attributes. E.g. speed can var very fast, so its detection can be made e.g. every 100 ms. On the other hand, bearing or housing issues develop slowly and so can be detected e.g. only once every second. In reality these values con be quite different (higher/lower), but to facilitate testing and validation, convenient values in the range of hundreds of ms can be used, trying at least to reflect that some tasks are quicker than the other ones. The attributes of these tasks, such as priorities and periods/mits, must be explicit and configurable.

Data generated by theses tasks is placed in an Real-Time DataBase (RTDB). A task prints the contents of the RTDB with a user-defined periodicity.

Moreover, data can also be graphically displayed. For example, it can be displayed the spectral contents of the acquired signal.

(Some) Requirements

- It must be implemented a CAB buffer system, or equivalent. Note that in this kind of applications there are tasks that can take much more computation time (e.g. computing the FFT) than others (e.g. low-pass filter). Moreover, some task require shorter activation periods/higher activation frequencies than others (e.g. speed can vary quickly while bearing

issues emerge slowly over time). For this reason it is important that tasks have suitable priorities and can keep a buffer for a potentially long time, so CABs are a good solution.

- Task priorities are left to you, but must be properly justified
- You can choose the frequency of activation of the tasks, noting that the system must be schedulable. This, of course, depends on the capacity of your computer and on the complexity of the algorithms you implement. It is not required to attain a given rate for any task, but the rate selected by you should be properly justified in the report.
- Note that sampling frequency and sample size/type can be adjusted to tune the workload. Note, however, that the sample code supplied to you may need to be adapted if you change the default configurations.
- Sound processing for system diagnosis and fault detection is a challenging topic with ongoing research. It is not expected that students implement complex algorithms. Simple (or even simplistic) approaches are fine, as this topic is out of the scope of the course. Use of complex signal processing libraries should be avoided. It is important that you understand the task code to reason about the execution time of tasks, and the use of e.g. ML/AI libraries (that implement very powerful but complex algorithms, hidden under a lot of abstractions and modules) makes the analysis of the code very (or better, extremely) difficult.

Note that this specification is incomplete and open, allowing for different approaches (as usual in real life). You are welcome to discuss with me any aspects of the work that raise doubts.

You only need to implement 3 or 4 processing tasks. It is suggested e.g.:

- Low-pass filter for noise attenuation
- Measuring speed
- Detecting bearing issues

It is suggested to use SDL to sound acquisition and graphical display, but you are free to choose any other library, at your will.

Deliverables

- A report, up to 8 pages, pdf format, submitted via eLearning, with:
  - Cover page: identification of the course and assignment, identification of the group members (first name, last name and ID number)
  - Description of the software architecture:
    - Which tasks were implemented and their role
    - Which data structures were used and how they are accessed
    - Which synchronization methods have been used

- Discussion of the task priorities, activation rates and system schedulability
  - A brief description (diagrams and text) of the **task execution sequence and relevant events** when the system is working normally is welcome.
  - The diagram should contain the start/finish times of tasks, operations on the buffers, signals exchanged, etc. A reader should be able to understand which task does what and when.
- Tests, results and analysis

- Zip file with the full project folder and files.
  - **The project code MUST have a Makefile and be buildable and executed just by typing make followed by make run.**
  - The use of any libraries other than SDL must be specified in the report.

Bibliography:

- Materials presented in the theoretical classes (class handouts, Buttazzo's book)
- For CABs, chapter "10.6.1 CYCLIC ASYNCHRONOUS BUFFERS" of the reference book
  - Giorgio Buttazzo (2011). HARD REAL-TIME COMPUTING SYSTEMS: Predictable Scheduling Algorithms and Applications, Third Edition, Springer, 2011
- SDL2 (online)
  - Main SDL 2 doc.
    - https://wiki.libsdl.org/SDL2/FrontPage
  - Using SDL for sound recording/playback
    - https://gist.github.com/cpicanco/12147c60f0b62611edb1377922f443cd
- Monitoring through vibration/sound
  - Kim, H.; Kim, J.; Han, K.; Won, D. 1D Modeling Considering Noise and Vibration of Vehicle Window Brushed DC Motor. Appl. Sci. 2022, 12, 11405
    - https://doi.org/10.3390/app122211405
  - Marcelo, Castelli & Fossatti, Juan & Terra, Jose. (2012). Fault Diagnosis of Induction Motors Based on FFT. 10.5772/37419.

- https://cdn.intechopen.com/pdfs/35219/InTech-Fault_diagnosis_of_induction_motors_based_on_fft.pdf