Real-time Control Systems:

A Tutorial

Based on the paper of the same name by **A. Gambier**

Diego Mendes, nº42875

Index

- Real-Time Systems
- Real-Time Operating Systems
  - RTOS and non-RTOS
  - Scheduling
    - Static Priorities
    - Dynamic Priorities
  - What not to do
- Digital Control Systems
  - Design considerations
  - Misconceptions about real-time
  - Implementation
  - What not to do
- Real-Time Platform
  - Choosing a Real-Time Platform
- Questions…

Real-Time Systems

- "A real-time systems is one in which the correctness of a result not only depends on the logical correctness of the calculation but also upon the time at which the result is made available."

Real-Time Operating Systems

- When looking at a RTOS, one should start by analyzing such features as:
  - It’s parallelism support (multi-tasking and multi-threading)
  - It’s predictability.
  - And it’s responsiveness to an external event.
  - Predictability and responsiveness make up for another feature named *system latency*.

Real-Time Operating Systems

- From a more technical perspective, one should look for:
  - Fast context switching.
  - Small sized OS.
  - Support for preemptive Scheduling based on priorities.
  - Inter-task communication and synchronization mechanisms.
  - Real-Time timers.

RTOS and non-RTOS

- The main difference between them is the task manager, which is composed by the Dispatcher and the Scheduler.
- Like a non-RTOS, a RTOS provides support for multi-tasking with multiple threads and inter-task communication and synchronization mechanisms such as semaphores, shared memory, pipes, mail boxes, etc…
- In RTOS synchronization is of an even greater importance in order to avoid blocking of shared resources and to guarantee that the tasks are preformed in the correct order when necessary.

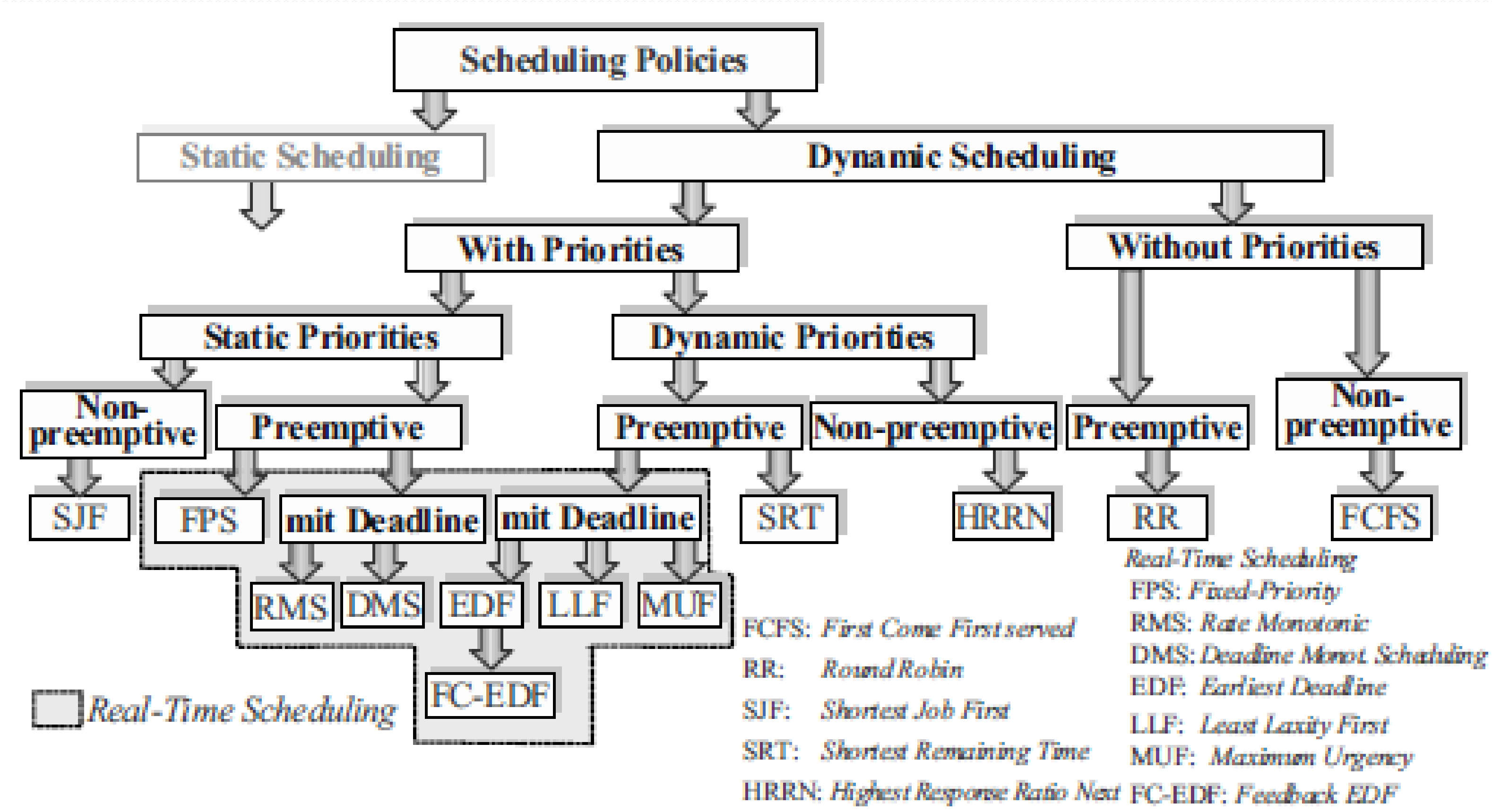
Real-Time Operating Systems: Scheduling

- In 1973, Liu and Layland showed that the total processor utilization U is given by

$$U = \sum_{i=1}^{n} \frac{C_i}{\min(D_i, T_i)}$$

- Today this equation is used as schedulability test where:
  - n – number of tasks
  - C – task execution time
  - D – task deadline
  - T – task period

Real-Time Operating Systems: Scheduling



Real-Time Operating Systems: Scheduling (Static Priorities)

- **FPS** (Fixed Priority Scheduling) – The order the tasks are executed is defined by their priority.
- **RMS** (Rate Monotonic Scheduling) – *"The shorter the period, the higher the priority."*
  - The utilization upper bound is given by $U \le n(2^{1/n} - 1)$
  - Optimal algorithm among fixed priority policies.
  - It's possible to know which deadline will be missed.
  - Low utilization(<70%) .
  - Fixed priorities lead to starvation and deadlocks.
  - Deadlines should be equal to periods.
- **DMS** (Deadline Monotonic Scheduling) – Modified RMS allowing tasks to have deadlines different from their periods.
  - Priority is inversely proportional to its deadline.

Real-Time Operating Systems: Scheduling (Dynamic Priorities)

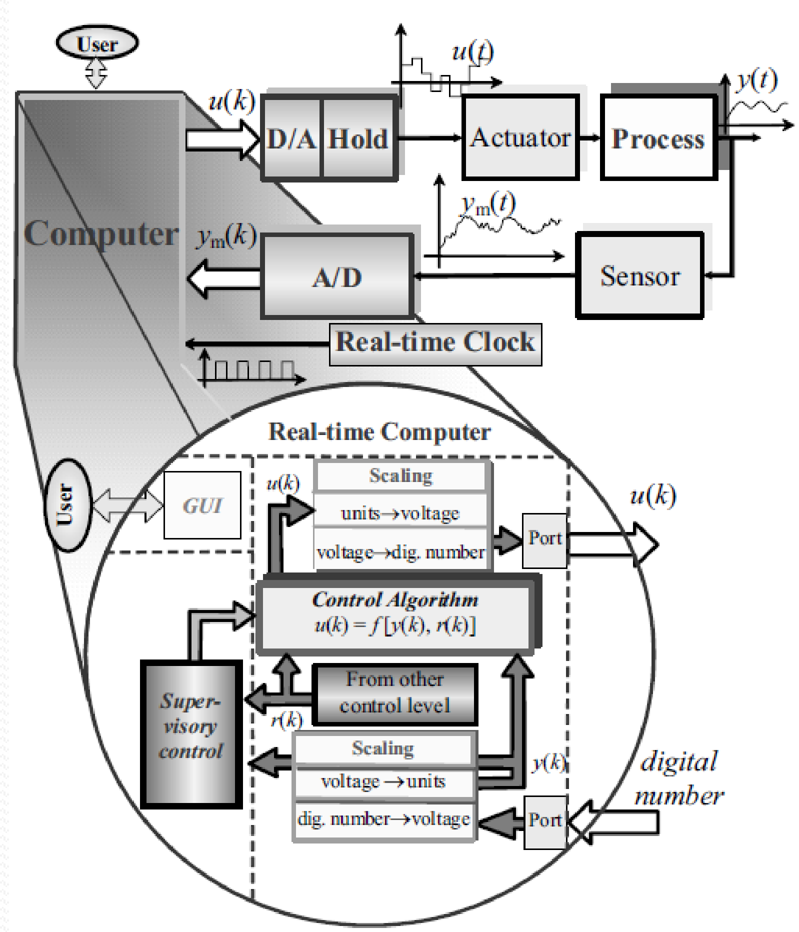
- **EDF** (Earliest Deadline First) – The closer a deadline is the greater a task priority becomes.
  - If all tasks are periodic and preemptive, this algorithm is optimal and has a utilization $U \leq 1$
  - The execution time of the task is not taken into account.
- **LLF** (Least Laxity First) – "The smaller the laxity, the higher the priority."
  - ***Laxity = deadline - remaining execution time***
  - It takes into consideration the execution time.
  - But it uses an estimation as the execution time might not be know *a priori,* so the scheduling might be incorrect.
  - There's also no way of knowing which task will fail in case of overload.
- **MUF** (Maximum Urgency First) – Priority is granted according to a task urgency, not allowing for more important tasks to fail their deadlines in because of less important tasks.
  - Urgency is defined by a combination of two fixed priorities and a dynamic one.
  - The dynamic priority is inversely proportional to the task laxity.
  - The fixed priorities are called *task criticality* and user *priority*.
  - *Task criticality > dynamic priority > user priority*

Real-Time Operating Systems: What not to do

- Don’t use large or many conditional statements.
- Don’t use empty/dummy loops as delays.
- Don’t use interrupts indiscriminately.
- Don’t use fixed configuration information(e.g. #define).
- Don’t use big single loop for implementation.
- Don’t use message passing as primary communication method.
- Carefully debug the code.
- Analyze memory usage during the design.
- Don’t design without execution-time measurement.

Digital Controlled Systems

- Today most of the implemented control systems are based on digital hardware.



Digital Control Systems: Design considerations

- Errors due to A/D and D/A conversions and limited length words calculations.
- Errors in the software development are common.
- Sampling is not uniform, periodic or synchronous("no zero-time-execution").
- There may be variations in the control algorithm execution time(control jitter).

Digital Control Systems: Misconceptions about real-time

- "Having D/A and A/D to interface between the controller and the real world is enough to obtain a real-time system."
- "If the physical process is slow there's no need for real-time."
- "Guarantying real-time performance is meaningless or that even though it wasn't taken into account, the control systems works."
- "Real-time programming is exclusively assembly coding, priority interrupt programming and device driver writing."

Digital Control Systems: Implementation(1)

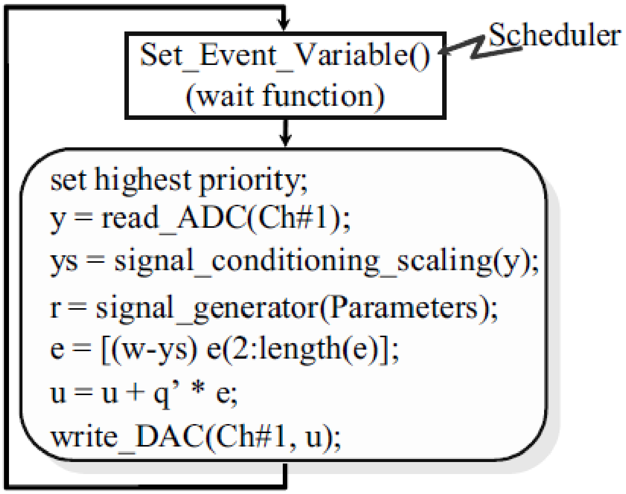
- A controller can be represented by the general polynomial equation.

$$R(q^{-1})\cdot u(k) = T(q^{-1})\cdot r(k) - S(q^{-1})\cdot y(k)$$

- The controller algorithm is executed once in every sampling period h.
- The sampling period is a compromise between Nyquist-Shannon Law *(fs>2B)*, the computation time delay τ with it's possible jitter and the limits of the hardware.
- When in a system we have $0<\tau<h$ we're facing a delay, as for when $\tau \geq h$ we have a loss.

Digital Control Systems: Implementation(2)

- A simple real-time implementation is conceivable with a single periodic task.
- The monolithic approach tends to lead to a larger feedback-delay.
- In order to diminish this delay a predictive controller may be implemented. A linear predictor is given by the following:



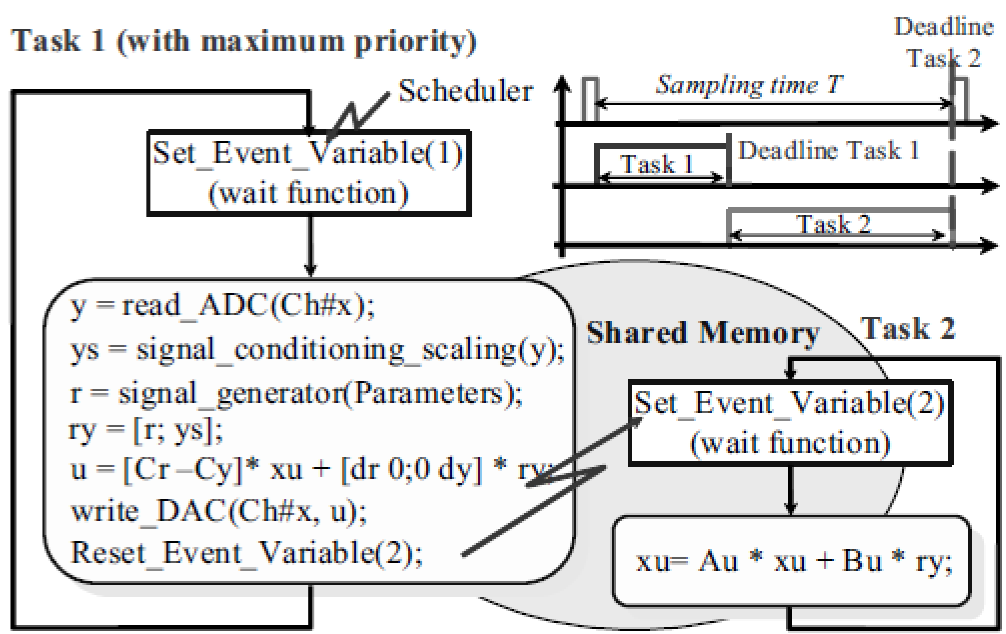
$$\frac{\hat{y}(k+1)-y(k)}{(k+1)-k}=\frac{y(k)-y(k-1)}{k-(k-1)}\equiv \hat{y}(k+1)=2y(k)-y(k-1)$$

Digital Control Systems: Implementation(3)

- Another approach is possible by dividing the process in more than one real-time task.
- Such an approach can be defined by the following equations:

$$u(k) = \left[\begin{array}{c|c} \mathbf{C}_r & -\mathbf{C}_y \end{array}\right] \left[\begin{array}{c} \mathbf{x}_r(k) \\ \hline \mathbf{x}_y(k) \end{array}\right] + \left[\begin{array}{c|c} d_r & \mathbf{0} \\ \hline \mathbf{0} & -d_y \end{array}\right] \left[\begin{array}{c} r(k) \\ \hline y(k) \end{array}\right]$$

$$\left[\begin{array}{c} \mathbf{x}_r(k+1) \\ \hline \mathbf{x}_y(k+1) \end{array}\right] = \left[\begin{array}{c|c} \mathbf{A} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{A} \end{array}\right] \left[\begin{array}{c} \mathbf{x}_r(k) \\ \hline \mathbf{x}_y(k) \end{array}\right] + \left[\begin{array}{c|c} \mathbf{B}_r & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{B}_y \end{array}\right] \left[\begin{array}{c} r(k) \\ \hline y(k) \end{array}\right]$$



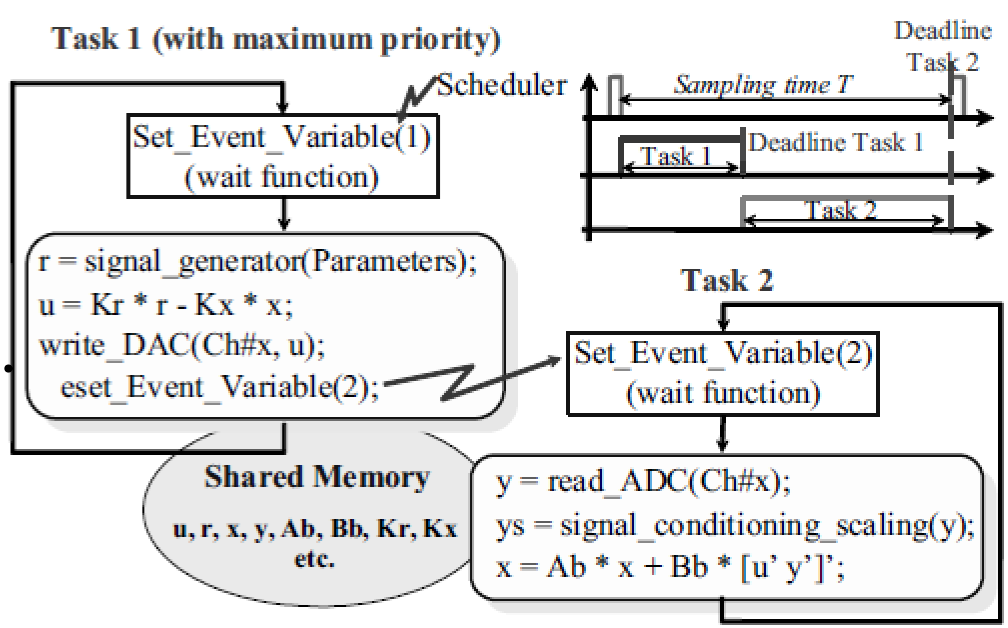
Digital Control Systems: Implementation(4)

- Such a structure is ideal for a state-space controller.

$$\mathbf{u}(k) = \mathbf{K}_r\,\mathbf{r}(k) - \mathbf{K}_x\,\hat{\mathbf{x}}(k)$$

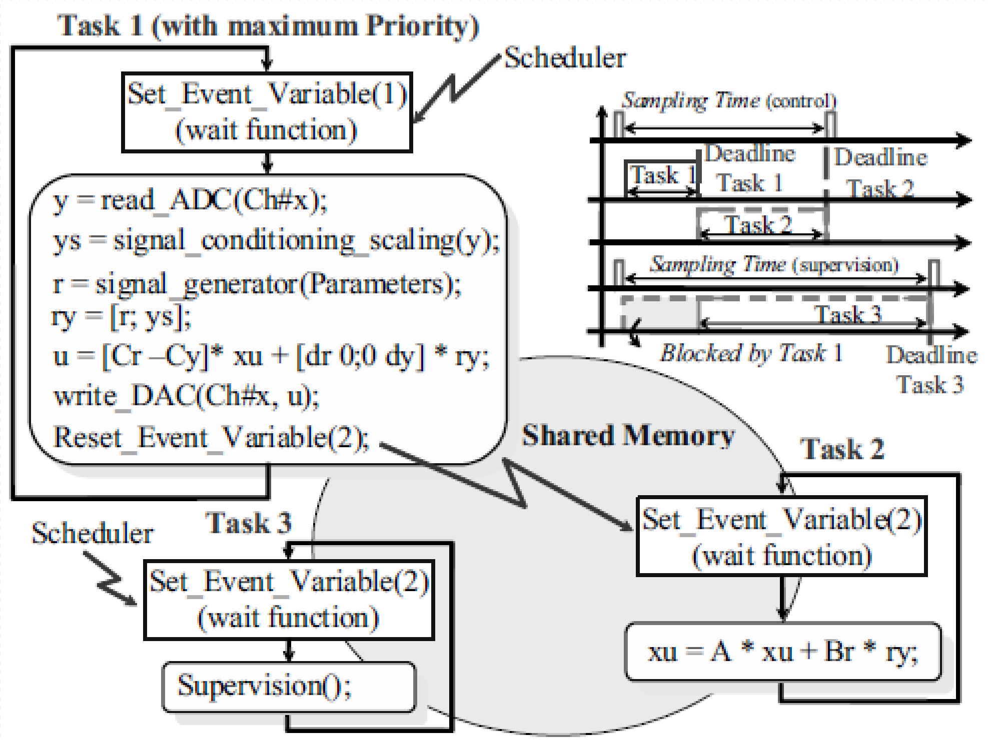
$$\hat{\mathbf{x}}(k+1) = [\mathbf{A} - \mathbf{K}_o\mathbf{C}]\,\hat{\mathbf{x}}(k) + [\mathbf{B} - \mathbf{K}_o\mathbf{D}]\,\mathbf{u}(k) + \mathbf{K}_o\mathbf{y}(k)$$

- The main task calculates the new control signal.
- Followed by Task 2 where the new state variable are calculated.



Digital Control Systems: Implementation(5)

- As an extra is also possible to add a system supervisor as a new task.

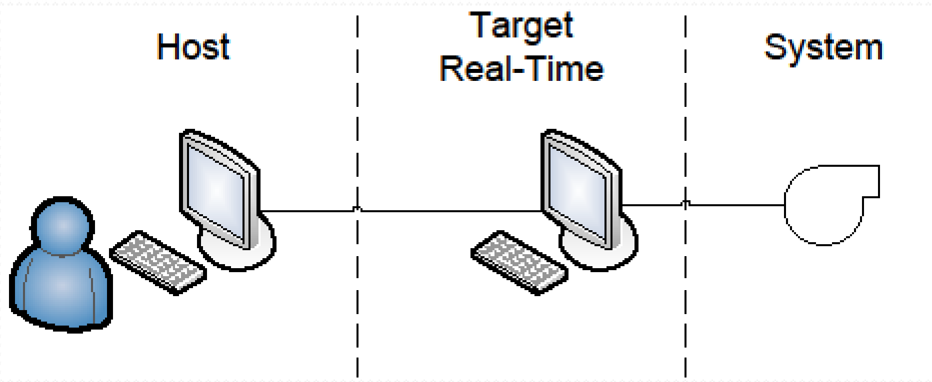


Digital Control Systems: What not to do

- Don't overlook the anti-aliasing filter.
- Don't implement the anti-aliasing filter in software.
- Don't overlook the signal scaling.
- Don't implement continuous-time controllers, when not necessary.

Real-Time Platform

- It’s very common to find a real-time platform composed by two computers, a host and a target.



Choosing a Real-Time Platform

- Preemptive Multitasking for hard real-time requirements.
- POSIX compliant.
- Support for real-time scheduling.
- Small latency.
- Integration with Labview.
- Existence of a development environment.

Questions…