Introducing critical real-time software design and programming

Frank Singhoff

University of Brest, Lab-STICC/CNRS UMR 6285, France

Projekt je financirala Europska unija iz Europskog Socijalnog fonda

Today agenda

1. Introduction to safety critical real-time software
2. Scheduling analysis
3. RTEMS Real-time operating systems

4. Labs on Cheddar and RTEMS, real-time scheduling analysis and programming in C

- To get lecture/lab material: http://beru.univ-brest.fr/split2022

Summary

1. Safety critical systems and software.

2. Critical real-time software.

3. Real-time operating systems and real-time scheduling analysis

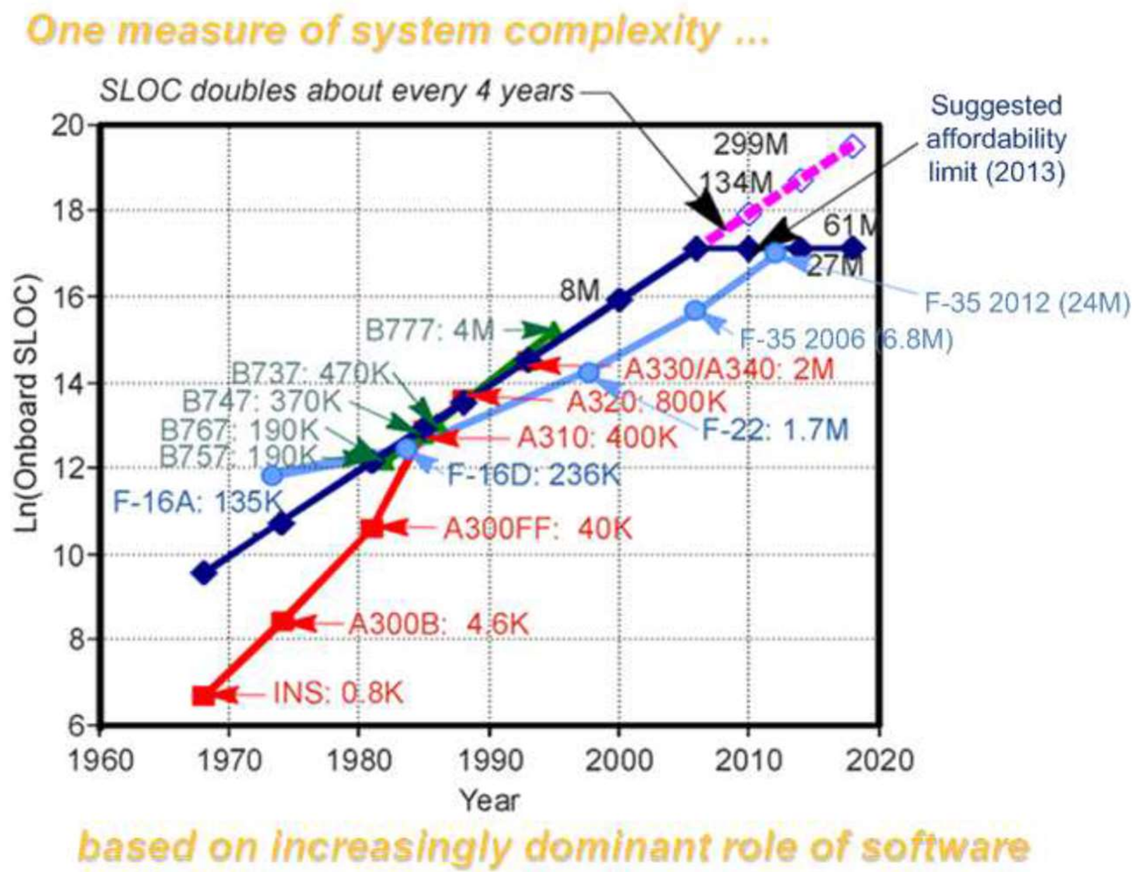
Safety critical systems

- **"A safety-critical system is a system whose failure or malfunction may result in death or serious injury to people, loss or severe damage to equipment/property, ... "**

- Examples: railway, aircraft, automotive, underground.
- Software contributes to the safety of the system.
- How to be sure that a software is safe? Bug free?
- Required by regulation (e.g. avionic systems).
- Today software embedded in critical systems is complex, large.

Avionic systems (1)

- From SAVI program (US research program) who investigated about software in avionic (Peter Feiler)
- SLOC, for Source Line of Code.



Avionic systems (2)

- F35 has approximately 175 times the number of SLOC as the F16.
- But, it is estimated to have required 300 times the development effort.
- Software development effort, which increases exponentially with SLOC, is increasing at an alarming rate

- Doubled every 4 years

Summary

1. Safety critical systems and software.

2. **Critical real-time software.**

3. Real-time operating systems and real-time scheduling analysis

Real-Time critical software (1)

- *« The correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced »* Stankovic, 1988.

- Properties we look for:
  - Functions must be predictable: the same data input will produce the same data output.
  - Timing behavior must be predictable: must meet temporal constraints (e.g. deadline).

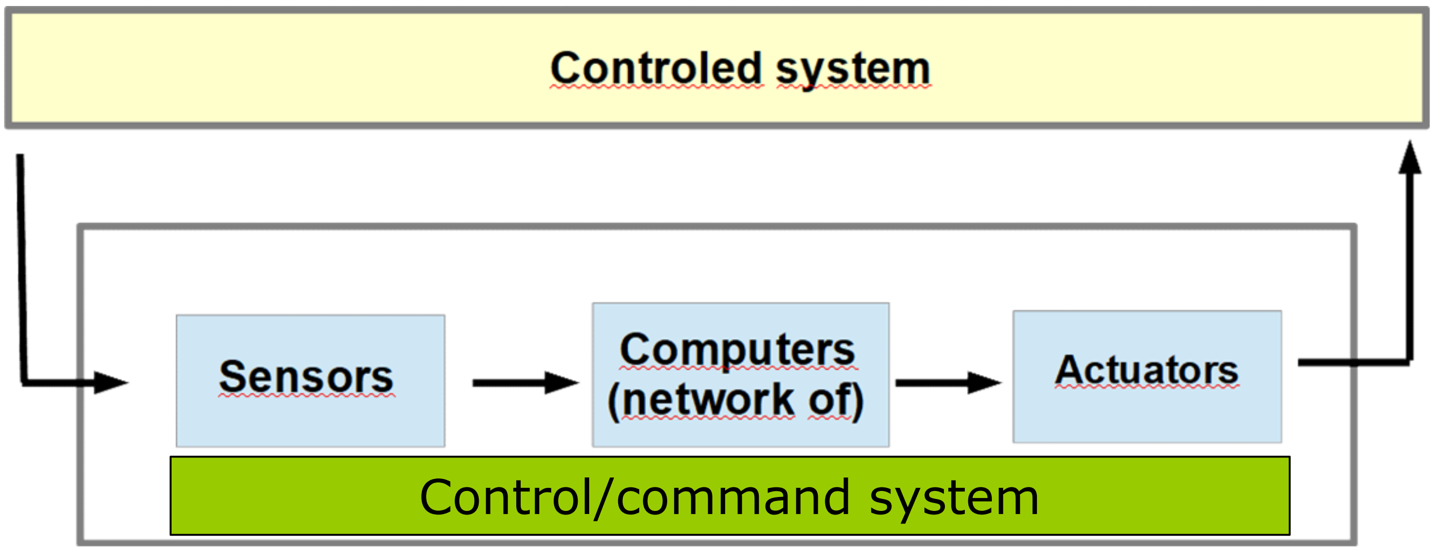
- Predictable means ... we can **compute** the program temporal behavior **before** execution time.

Real-Time critical software (2)

- **Critical real-time systems:** temporal constraints MUST be met, otherwise defects could have a dramatic impact on human life, on the environment, on the system,

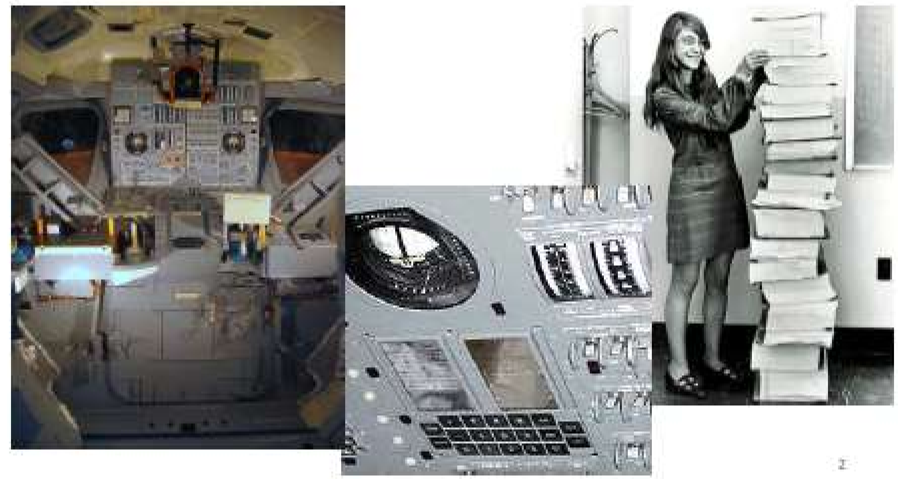
- **Examples of temporal constraints:**
  - **Few milliseconds** for radar systems.
  - **One second** for machine-man interfaces (in an aircraft for example).
  - Up to **several months or years** for spacecrafts (Mars Express, Voyager, ...).

Real-Time critical software (3)



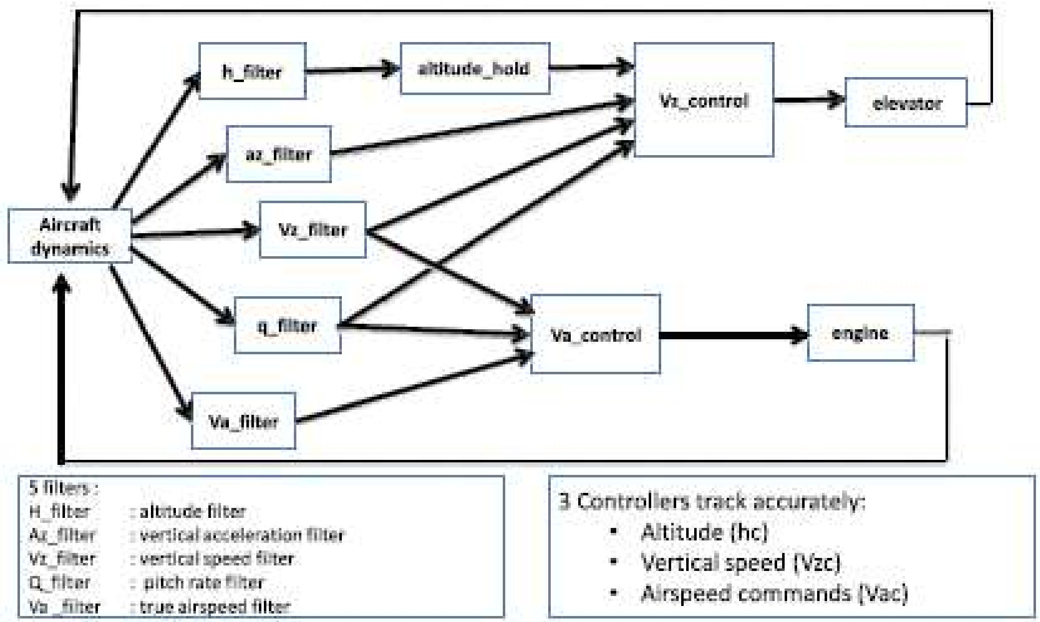
- *Real-time control and command software:* computing system/programs which reacts in a given time 1) from sensor inputs 2) to send commands to actuators.
- How to prove that the software will react in a given time/duration? deadline?

Space software



- Apollo Guidance Computer (AGC).
- One of the first critical real-time system. 65000 SLOC in assembly language.
- Quality project manager: Margaret Hamilton.
- Probably the first fixed priority operating system => alarm handling during Apollo 11 landing on moon.

Avionic real-time software (1)



- ROSACE Aircraft flight control-command software (Pagetti 2014).
- Objectives: control aircraft take off.
- Inputs/sensors: airspeed, elevation, ...
- Outputs/actuators: engine, ...

Avionic real-time software (2)

Task	WCET us	Period us
aircraft_dynamics	200	5000
Va_c, h_c	500	20000
H_filter, Az_filter, Va_filter, q_filter, az_filter	100	10000
delta_e_c delta_th_c	500	20000
Altitude_hold, va_control Vz_control	100	20000
Engine, elevator	100	5000

- Period = fixed delay between each work ; WCET = worst case execution time
- Implemented as a set of 14 tasks. 2300 SLOC in C language.
- Fully open-source, i.e. POSIX C source code available.

Summary

1. Safety critical systems and software.
2. Critical real-time software.
3. **Real-time operating systems and real-time scheduling analysis**

Scheduling analysis, what is it ?

- **Real-time software** has temporal constraints to meet (e.g. deadline).
- Many systems are built with operating systems providing multitasking facilities … Tasks may have deadline.
- Take the task scheduling into account in order to check task temporal constraints.
- How the OS must schedule? How to predict?

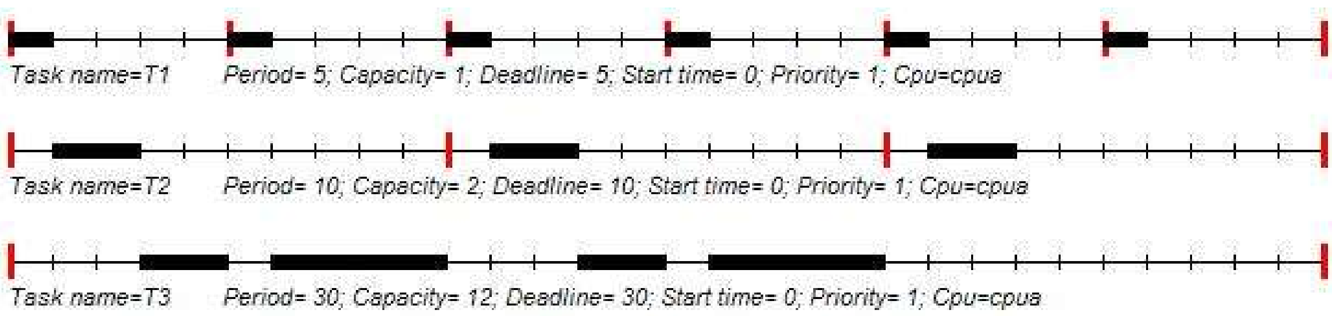
Real-Time scheduling

1. **Simplified tasks models** (to model functions of the system)

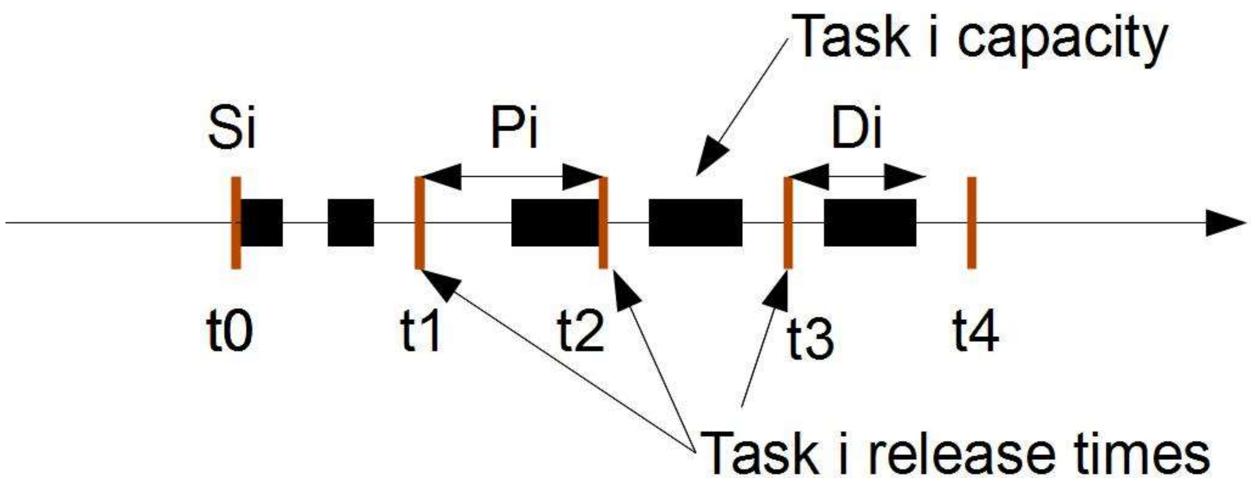
2. **Analytical methods** (called feasibility tests)
   - **Example:**

$$R_i \leq Deadline \qquad R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j$$

3. **Scheduling algorithms**: build the full scheduling/GANTT diagram



Real-time scheduling: models of task



- **Usual parameters of a periodic task i:**
  - **Period:** Pi (duration between two release times). A task starts a job for each release time.
  - **Deadline to meet:** Di, timing constraint to meet.
  - **First task release time (first job):** Si.
  - **Worst case execution time of each job:** Ci (or capacity or WCET).
  - **Priority:** allows the scheduler to choose the task to run

Uniprocessor fixed priority scheduling

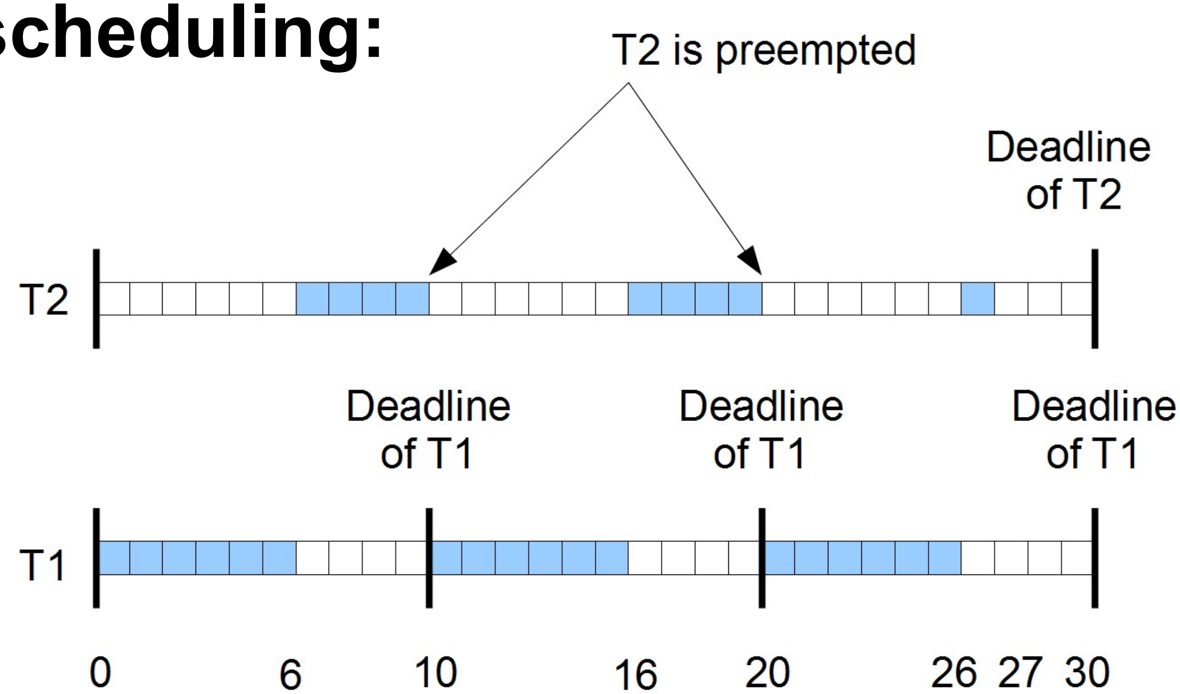
- **Fixed priority scheduling :**
  - Scheduling based on fixed priority => priorities do not change during execution time.
  - Priorities are assigned at design time (off-line).
  - Scheduler easy to implement into real-time operating systems.
- **Rate Monotonic priority assignment :**
  - Optimal assignment in the case of fixed priority scheduling and uniprocessor.
  - Periodic tasks only.

Uniprocessor fixed priority scheduling

- **Two steps:**
  1. **Rate monotonic priority assignment:** the highest priority tasks have the smallest periods. Priorities are assigned off-line (e.g. at design time, before execution).
  2. **Fixed priority scheduling**: at any time, run the ready task which has the highest priority level.

Uniprocessor fixed priority scheduling

- **Rate Monotonic assignment and preemptive fixed priority scheduling:**



- Assuming VxWorks priority levels (high=0 ; low=255)
- T1 : C1=6, P1=10, Prio1=0
- T2 : C2=9, P2=30, Prio2=1

Uniprocessor fixed priority scheduling

- **Schedulability tests <u>to predict on design-time if deadline will be met</u>:**

1. **Run simulations on feasibility interval** = [0,LCM(Pi)].
   Sufficient and necessary condition.

2. **Processor utilization factor test:**
   $U = \sum_{i=1}^{n} Ci/Pi \leq n.(2^{\frac{1}{n}}\text{-}1)$ (about 69%)
   Rate Monotonic assignment and preemptive scheduling.
   Sufficient but not necessary condition.

3. **Task worst case response time, noted Ri** : delay between task release time and task completion time. Any priority assignment, preemptive scheduling.

Uniprocessor fixed priority scheduling

- **Compute Ri, task i worst case response time:**
  - Task i response time = task i capacity + delay the task i has to wait for higher priority task j. Or:

$$R_i = C_i + \sum_{j \in hp(i)} waiting\ time\ due\ to\ j \qquad or\ R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j$$

  - hp(i) is the set of tasks which have a higher priority than task i.
  - $\lceil x \rceil$ returns the smallest integer not smaller than x.

Uniprocessor fixed priority scheduling

- To compute task response time: compute wi^k with:

$$wi^n = Ci + \sum_{j \in hp(i)} \lceil wi^{n-1}/Pj \rceil . Cj$$

- Start with wi^0=Ci.
- Compute wi^1, wi^2, wi^3, ... wi^k upto:
  - If wi^k >Pi. No task response time can be computed for task i. Deadlines will be missed !
  - If wi^k = wi^{k-1}. wi^k is the task i response time. Deadlines will be met.

Uniprocessor fixed priority scheduling

- **Example:** T1(P1=7, C1=3), T2 (P2=12, C2=2), T3 (P3=20, C3=5)

$$w1^0 = C1 = 3 \Rightarrow R1 = 3$$

$$w2^0 = C2 = 2$$

$$w2^1 = \mathrm{C2} + \left\lceil \frac{w2^0}{P1} \right\rceil . C1 = 2 + \left\lceil \frac{2}{7} \right\rceil . 3 = 5$$

$$w2^2 = \mathrm{C2} + \left\lceil \frac{w2^1}{P1} \right\rceil . C1 = 2 + \left\lceil \frac{5}{7} \right\rceil . 3 = 5 \Rightarrow R2 = 5$$

$$w3^0 = C3 = 5$$

$$w3^1 = \mathrm{C3} + \left\lceil \frac{w3^0}{P1} \right\rceil . C1 + \left\lceil \frac{w3^0}{P2} \right\rceil . C2 = 10$$

$$w3^2 = \mathrm{C3} + \left\lceil \frac{w3^1}{P1} \right\rceil . C1 + \left\lceil \frac{w3^1}{P2} \right\rceil . C2 = 13$$

$$w3^3 = \mathrm{C3} + \left\lceil \frac{w3^2}{P1} \right\rceil . C1 + \left\lceil \frac{w3^2}{P2} \right\rceil . C2 = 15$$

$$w3^4 = \mathrm{C3} + \left\lceil \frac{w3^3}{P1} \right\rceil . C1 + \left\lceil \frac{w3^3}{P2} \right\rceil . C2 = 18$$

$$w3^5 = \mathrm{C3} + \left\lceil \frac{w3^4}{P1} \right\rceil . C1 + \left\lceil \frac{w3^4}{P2} \right\rceil . C2 = 18 \Rightarrow R3 = 18$$

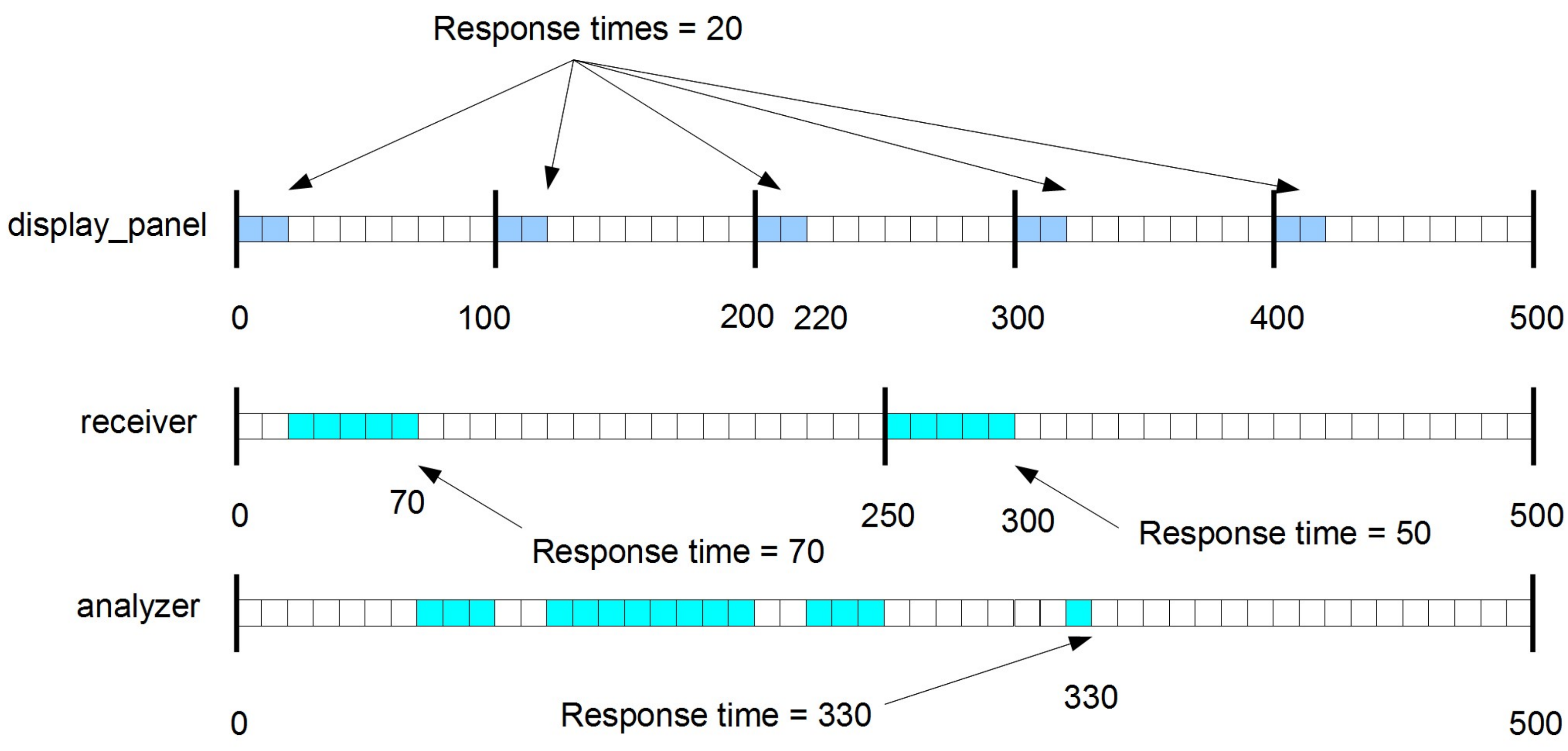
Uniprocessor fixed priority scheduling

- **Example:**
  - "display_panel" thread which displays data. P=100, C=20.
  - "receiver" thread which sends data. P=250, C=50.
  - "analyser" thread which analyzes data. P=500, C=150.

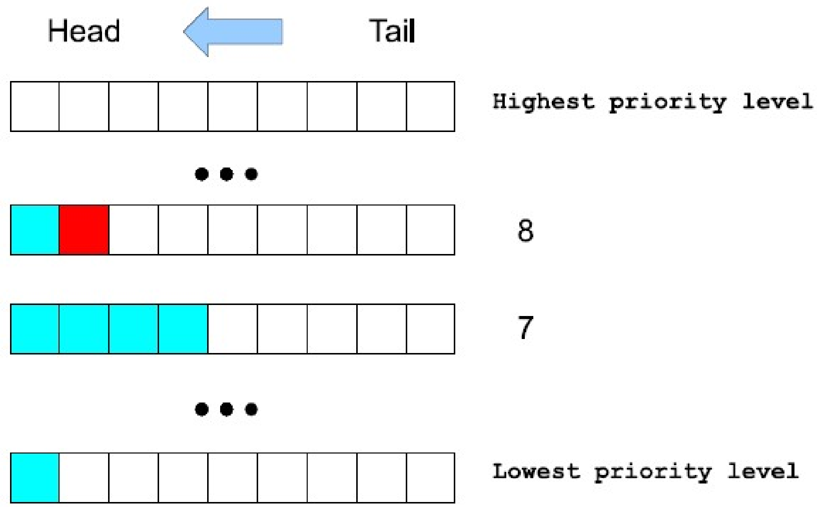
- **Processor utilization factor test:**
  - U=20/100+150/500+50/250=0.7
  - Bound=3.$(2^{\frac{1}{3}} - 1)$=0.779
  - U≤Bound => deadlines will be met.

- **Worst case task response time:** $R_{analyser}$=330, $R_{display\_panel}$=20, $R_{receiver}$ =70.
- **Run simulations on feasibility interval:** [0,LCM(Pi)] = [0,500].

Uniprocessor fixed priority scheduling



RTEMS operating system



- **Compliant with the POSIX real-time scheduling model**
  - Several threads inside one address space
  - Preemptive fixed priority scheduling. At least 32 priority levels.
  - Two-levels scheduling, choose:

    1. The queue with the highest priority level ready thread.

    2. The thread from the queue selected in (1) according to a policy (e.g. SCHED_FIFO or SCHED_RR).

Conlusion/Summary

- **Software is now of a major concern for safety of critical systems**
- **Real-time critical software:** software with timing constraints to meet (deadline). Concurrent software (i.e. tasks and synchronization).

- **Specific development technologies (design, verification, programming):**
  1. Scheduling/schedulability analysis.
  2. Real-time operating systems.