Real-Time and Embedded Systems @ SIT

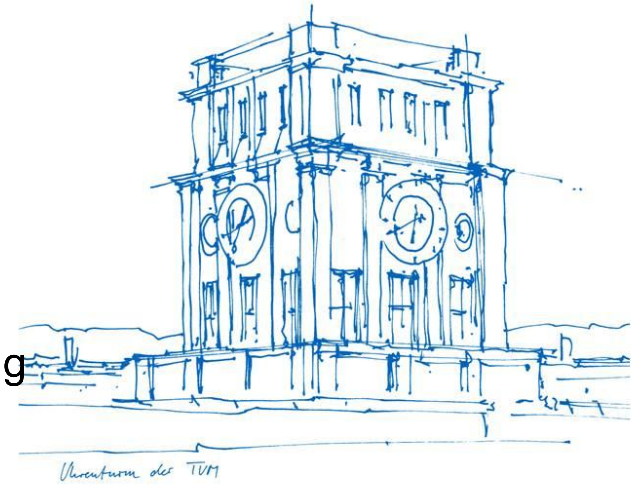
Scheduling

Alexander Hoffman

Technical University of Munich

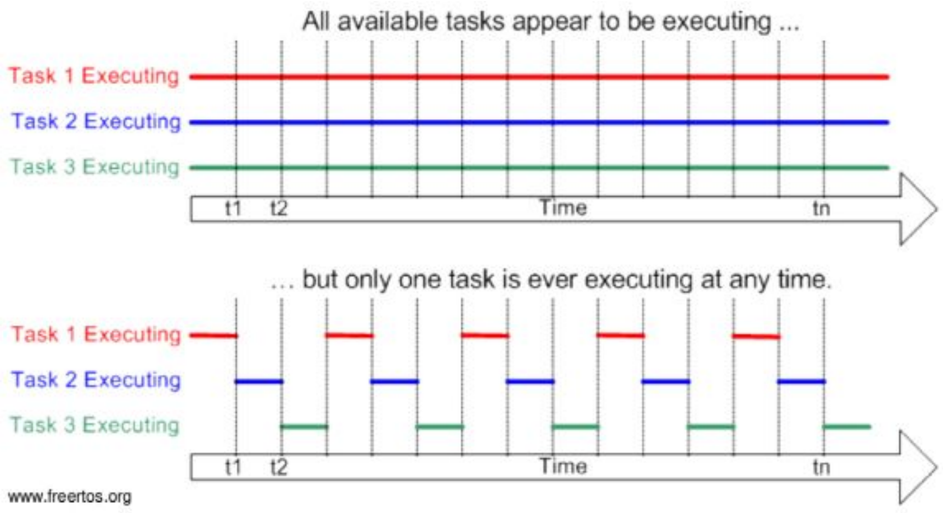
Department of Electrical and Computer Engineering

Chair of Real-Time Computer Systems



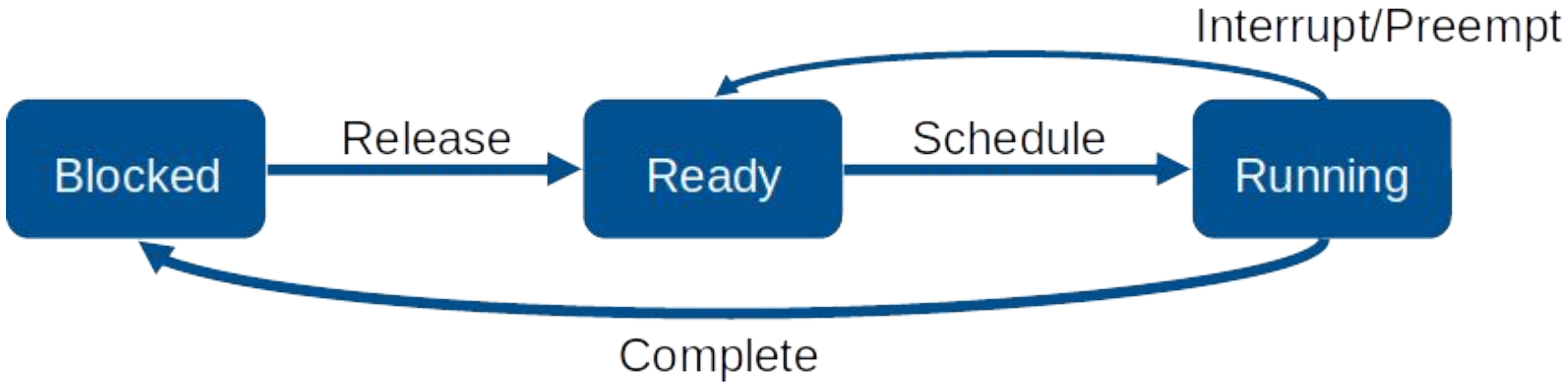
Overview

- We want to run multiple tasks/processes on a single CPU core
- Tasks must either:
    - Run one after the other
    - Switch between execution of different tasks



Release Mechanisms

- The scheduler can only execute tasks that are **ready**, ie. not blocked or suspended



Release Mechanisms

Time Triggered:

- Tasks are released at specific points in time
- Usually periodic, but also more complex schemes possible
  - Eg. releasing a frame render task in a game to achieve a certain FPS

Event Triggered:

- Tasks are released in response to some event happening
  - Eg. NW-packet arrives (fires interrupt), timer, sensor value changes, semaphore becomes available
- Generally based on interrupts

Static vs Dynamic Scheduling

Static Schedule:

- There is a predetermined time table that specifies which task is allowed to run and when
- Can also include fixed scheduling of message transmissions
- Best fits time triggered task releases
- Common on bus systems
  - Eg. Time-division multiple access (TDMA)

Dynamic Schedule:

- The scheduler decides at runtime which task should be run next
- Only tasks that are ready/released/unblocked can be picked
- Scheduling decisions are made based on order (Eg. FCFS) or priority

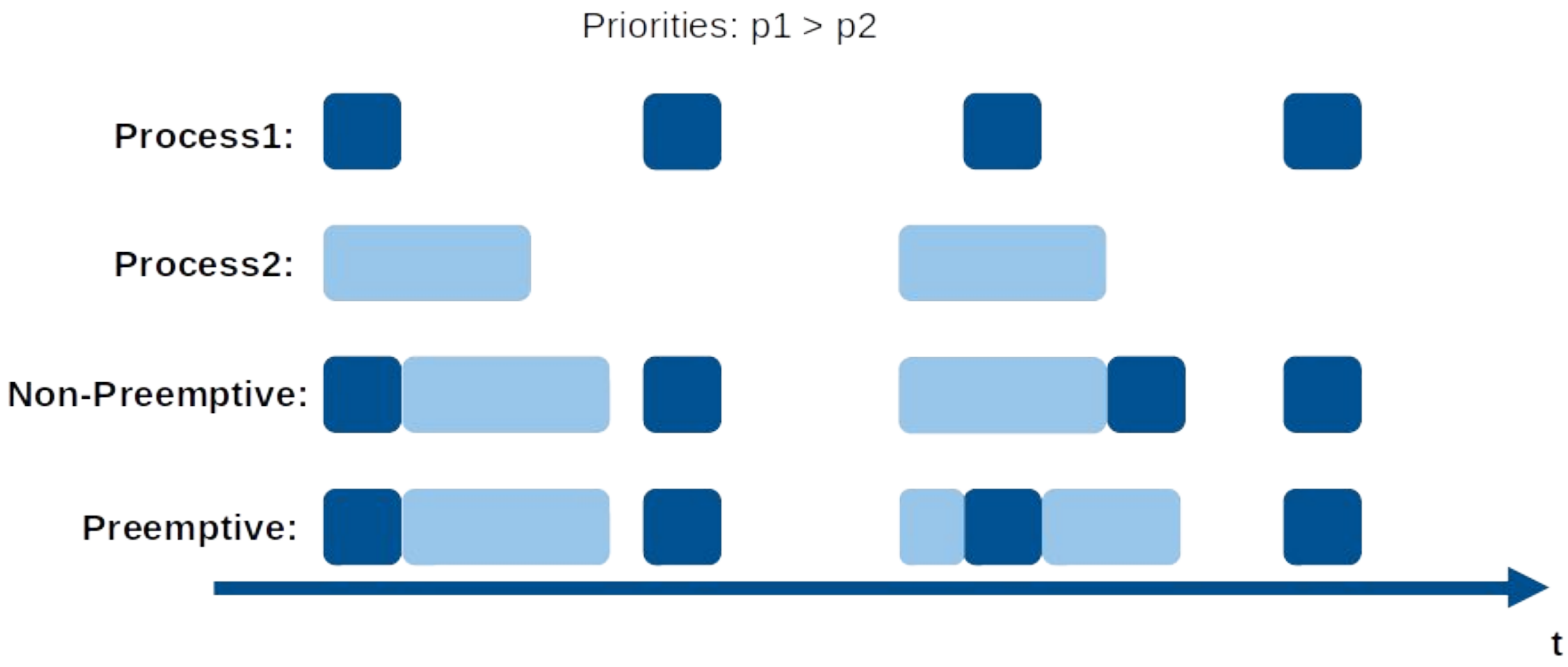
Preemption

- The temporary interruption of an executing task with the intention of finishing its execution at a later time point
- Does not require the task's cooperation
- Decision to preempt a task is made by the scheduler

Non-Preemptive

- Once a task is running, it runs until completion
- Advantages:
    - Easy to implement
    - Low overhead
- Disadvantages:
    - Low priority task can significantly delay a high priority task
    - More difficult to analyze globally
    - Generally less responsive systems
- Common on communication protocols
- Unsuitable when execution times of one task are longer than deadlines from another task

Preemptive vs. Non-Preemptive



Priorities

No Priority:

- Tasks are not scheduled according to priority
- Usually a static schedule or ordered by first-come-first-served (FCFS)
- Scheduler will often try to be fair (Round Robin)

Fixed Priority:

- Each task is assigned a fixed priority at design time
  - Eg. rate monotonic (task with shortest period gets the highest priority)

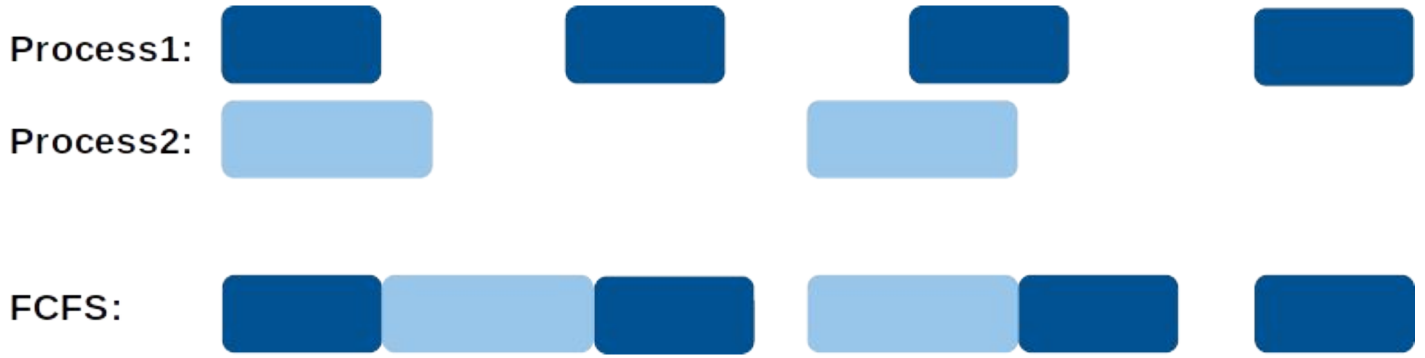
Dynamic Priority:

- Priority of a task can change during runtime depending on factors such as:
  - Remaining time to deadline
  - What resources are currently being held (priority inheritance)

Dynamic Scheduling Strategies - FCFS

First Come First Served (FCFS)

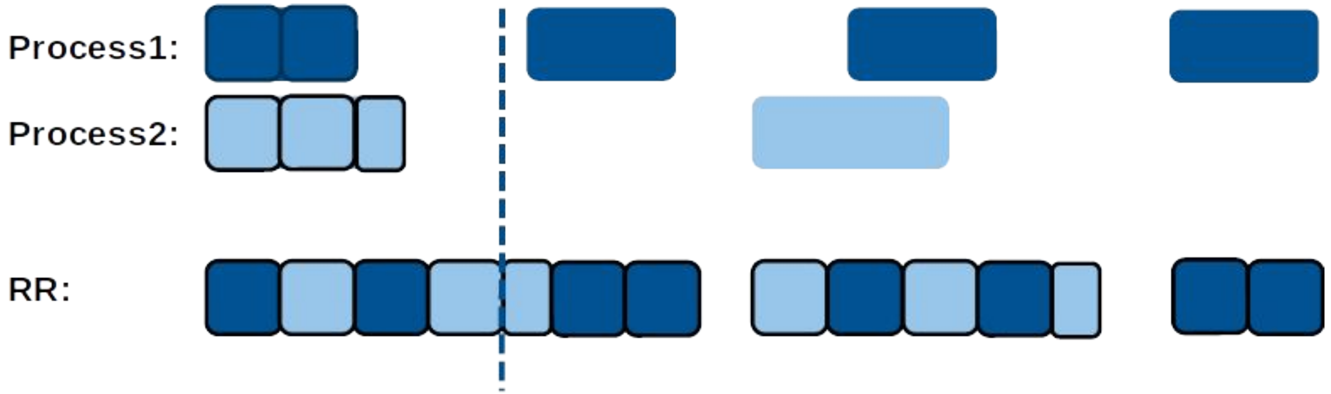
- Released tasks are put into a FIFO queue
- When a task completes the scheduler picks the first task from this queue and lets it run to completion
- Non-preemptive
- No priorities, although multiple queues with different priorities might exist
- Used by FreeRTOS to schedule tasks of same priority



Dynamic Scheduling Strategies - RR

Round Robin (RR)

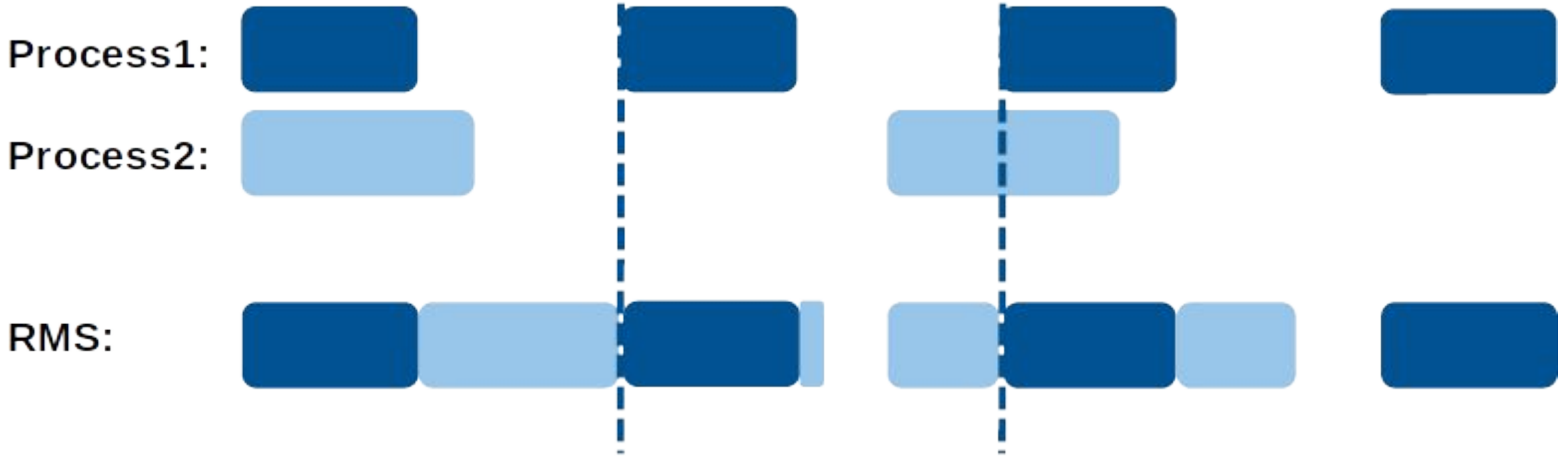
- Released tasks are put into a FIFO queue
- Scheduler picks first task in the queue
- After a fixed amount of time (a quantum) the current task gets blocked and put to the back of the queue, at which point the scheduler chooses the next task to execute from the front of the queue



Dynamic Scheduling Strategies - RMS

Rate Monotonic Schedule (RMS)

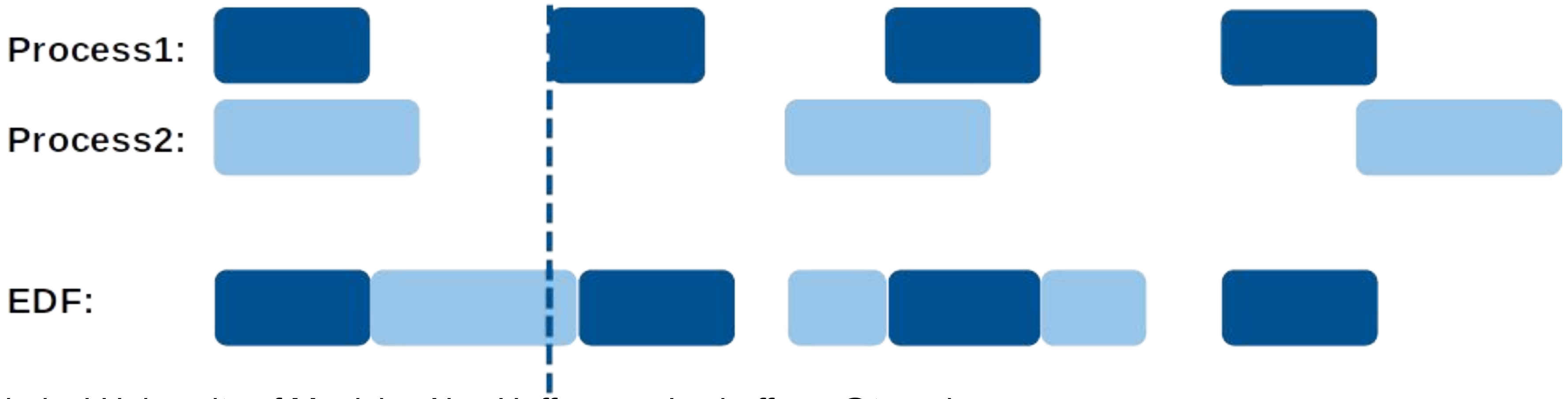
- Fixed priority schedule
- Highest priority assigned to task with shortest period
- Preemptive scheme (low priority task gets interrupted when high prio task is released)



Dynamic Scheduling Strategies - EDF

Earliest Deadline First (EDF)

- Dynamic priority schedule
- Highest priority assigned to task with closest deadline
- Preemptive
- Optimal schedule: If a task set is schedulable at all, it is schedulable with EDF



Schedulability Analysis

Is a given task set schedulable?

Each set of tasks is defined by:

- T_i : Period/Minimal inter arrival time
- e_i : Execution time
- p_i : Priority
- d_i : Deadline
- r_i : Response time
- Our assumption: Deadline = Period / Minimal inter arrival time

- Necessary, but not sufficient condition: Utilization must be <= 1
- Necessary and sufficient condition: $r_i <= d_i$ for all tasks

Schedulability Analysis

Is a given task set schedulable?

First Test: Is processor utilization <= 1?

$$U = \sum \frac{e_i}{T_i} \leq 1$$

- No: Task set **is NOT** schedulable
- Yes: Task set **MAY** be schedulable, further steps required to determine schedulability

Quick first test to see if you should continue with more involved tests

Schedulability Analysis - RMS

Second Test: Liu & Layland (1973) proved that for a RMS task set **IS** schedulable if

$$U = \sum \frac{e_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)$$

Where *n* is the number of periodic tasks

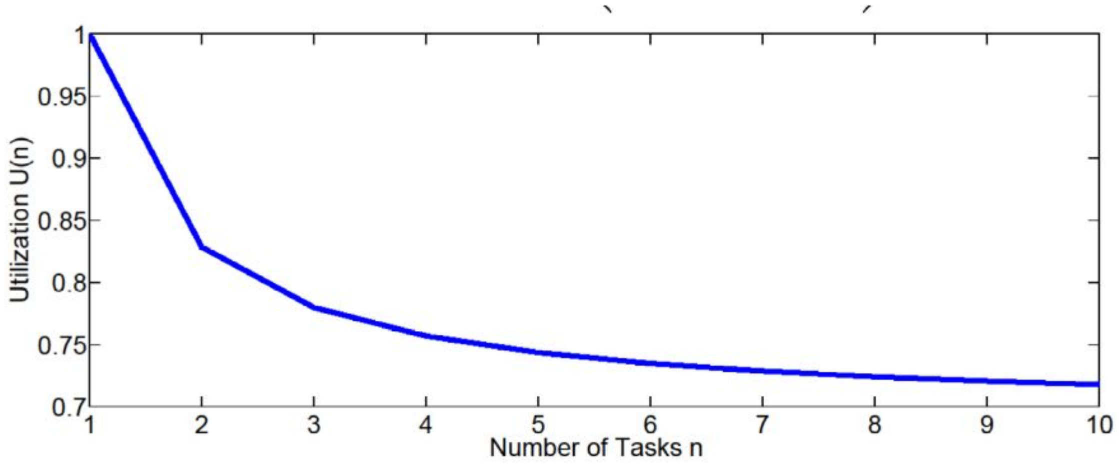
- Yes : Task set **IS** schedulable
- No : Task set **MAY** be schedulable, more investigation required

Schedulability Analysis - RMS

Taking the limit of the Liu & Layland method

Therefore you can estimate that any set of RMS tasks (infinite limit) can meet their deadlines if CPU util is less than 69.32%

$$\lim_{n\to\infty} n(\sqrt[n]{2} - 1) = ln2 \approx 0.6931$$



Schedulability Analysis - RMS

Is a given task set schedulable?

Third Test: Compute the actual response times via fixpoint computation

- Fixpoint computation: a value in a function's input domain is mapped to itself via the function
  - Ie. *f(c)* = *c*, will reach a steady state when the result (c) is fed back into the function *f(c),* meaning (*c,f(c)*) lies on line *x* = *y*
- Problem:
  - Response time of task depends on the number of interruptions
  - The number of interruptions depends on the response time of task (non-linear)

Schedulability Analysis - RMS

Response time given by

$$r_i = e_i + \sum_{k=hp(i)} \lceil \frac{r_i}{T_k} \rceil e_k \leq d_i$$

Where:

e_i is the execution time of the task

hp(i) is the set of higher priority tasks

T_k is the period of the task

d_i is the deadline of the task

Schedulability Analysis - RMS

Iteratively solving to find the fixpoint for each task using iterative formula:

$$r_i^n = e_i + \sum_{k=hp(i)} \lceil \frac{r_i^{n-1}}{T_k} \rceil e_k \leq d_i$$

Where:

r_i^{n-1} the response time from the previous iteration

$r_i^0 = e_i$

If all tasks' response time found to be <= d_i then system is schedulable

Other Important Factors

- Synchronization (Mutex, Semaphore):
    - We only considered tasks that run independent of each other, but
        - They might need to communicate (share date)
        - They might want to use the same resources (I/O, memory)
    - Need a way to communicate and/or get exclusive access to shared resources
- Priority Inheritance:
    - Low priority tasks the have exclusive access to share resources temporarily inherits priority from higher priority task that wants to access the same resource
- Deadlocks:
    - Two tasks need to get exclusive access to two shared resources before they can progress
    - They both manage to obtain one and are waiting for the other to be released

Other Important Factors

- Arbitration/decentralized scheduling (eg. CAN bus, Ethernet)
  - For event based communication over a shared medium we need a distributed scheduling protocol otherwise, two senders might corrupt each other
  - Ethernet: CDMA/CD(Carrier Sense, Multiple Access/Collision Detection):
    - Participants listen if there is a collision on the bus, stop transmission, if there is a collision and retransmit after a random backoff period
  - CAN Bus (Controller Area Network): CDMA/CA (Carrier Sense, Multiple Access/Collision Avoidance):
    - Messages have a preamble. If two nodes start to send at the same time, one preamble will be uncorrupted and that node is allowed to continue transmission