**Real Time Systems**

**Introduction part**

**What is Real-Time?**

Correctness and Response Time of the results are guaranteed. An RTS will also guarantee that a certain deadline is met (*dass eine bestimmte Frist eingehalten wird*).

**Three types of RTS:**

**-Hard RTS:** missing a deadline is a total system failure (e.g. airbag in car).

(when you miss some deadline its dead )

**-Soft RTS:** The usefulness of a result degrades after its deadline, thereby degrading the system’s quality of service. (e.g. warning systems, distance warner).

(*Die Nützlichkeit eines Ergebnisses verschlechtert sich nach Ablauf der Frist, wodurch die Servicequalität des Systems beeinträchtigt wird. (z.B. Warnsysteme, Abstandswarner*)).

(you have to meet deadline, deadline is like warningsystem, if you get warned, you hava to react, later its worst)

**-Firm RTS(***firm= feste/bestimmte****?*): periodic systems:** Infrequent(*Häufige*) deadline misses(*Deadline-Fehler*) are tolerable but may degrade (*beeinträchtigt*) the system’s quality of service. The usefulness of a result is zero after its deadline. (e.g. ignition-point-optimizer for motor (*z. B. Zündpunkt-Optimierer für Motor*))*.*

(either you have met the deadline, if you miss one you miss a single one, its tolorable)

You can **distinguish or classify real time systems** by:

- the consequences of **missing deadlines**

-**reliability** (*Zuverlässligkeit*) and **fault tolerance** (*Fehlertoleranz*)

-**centralized** and **distributed realtime systems**

(Arduino in the exercises is a centralized system, because its only one piece. The rador system or the airport are distributed systems because you have a bunch (*Bündel*) of componets that have to act with each other to meet the time constraints.

-**hierarchical** or **independent systems**

(that means you may have to have a controller that controls all subsystems than you have hierarchical system ((e.g. baggage handling system (*Gepäckförderanlage*) and the independent system is programming fire and forget)

**Different between independent and autonomes**: systems are not organized hierachical (independent) and autonomes means no interaction at all.

-**time driven and event driven** but we may also have combinations:

For example the approach light (in the exercises). The light flutter (*das Lichtflattern*) is time driven. But if we push the button it is event driven. So this system is event and time driven.

**Time driven Systems** (exp.): mp3 player, that has to convert something

**Event driven System**(exp.): computer game

**Embedded System:**

Real time systems are sometimes or often embedded systems.

An Embedded System is a computer that is a part of a overall system (*der Teil eines Gesamtsystems ist*) and controls physical processes.

Computer is embedded in a technical system

You don’t see a computer in an embedded system, for example pocket calculator, or a washing machine.

Repair costs are often higher than a new system, for example when your pocket calculator is defect you throw it away and buy a new one.

**Systems:**

A System is a set of Elements, Objects, Components or Modules that have a relations and may interact with each other. It has a well defined purpose(*Zweck*): it fulfills this purpose mostly by processing (*durch Verarbeitung*) incoming and outgoing Material, Energy or Data.

The interaction between the components are also performed by exchanging Material, Energy or Data.

**Process (DIN ISO 8402):**

A process is a set of relations or actions that converts a predefined input into a required outcome. (*Ein Prozess ist ein Satz von Wechselbeziehungen oder Tätigkeiten, die Eingaben in Ergebnisse wandeln*). So any Process has a well defined starting point, than we have an interaction, reaction or a task (something to do), and then we have a completion or an output.

**Physical process and competitional process:**

both interact with different actors and sensors.

Time constraints: competitional time (delta t) execution of the computing process is limited by the time difference between starting (request of a competitional task) and the latest completion.

**Time Standards**

**What is time (for computer scientists?):**

**We assume**

- that time exists

- In real world there exists one and only one coordinated Universal Time

- it is a continuum

**We define**

- that time is increasing all the time

- that every clock/computer has its own computer time (it depends when we have started the computer)

**Time** is based on **astronomical phenomena:**

**- Year:** Time distance between the recurrence of a distinct position of the sun (*Zeitabstand zwischen dem Wiederauftreten einer bestimmten position der Sonne*).

**- Month**: Time distance between 2 times new moon.

**- Day:** Time distance between two maximum altitudes of the sun

**- Hour:** A day has 24 hours

**- Minute:** An hour has 60 minute

**- Second:** A minute has 60 seconds

🡪 Second: (24hours \* 60min \* 60sec = 86400sec), commonly understood and historically defined as 1/86400 of a day.

- **UT** = Universal Time

**- UTC** = Universal Time Coordinated should be used for the german living.

Increases continuously.

Bases for legal time. //*(Koordinierte) Weltzeit*

**- GPS** = global positioning system (for navigating systems), starts at 1980

**- Internation Atomic Time** (TAI: Temps Atomique International)

TAI is based on the SI-second. It runs continuously.

**- Physical Quantity comprises of** (*physikalische Größe besteht aus*): absolute measure (*Maßzahl*) and measurement unit (*Maßeinheit*)

Example: Time

SI is the „**System** **International System of Units**“

**SI Base Units:**

**Symbol Name Quantity**

A ampere electrical current

(if we take to much current for Arduino it will burn down)

K kelvin temperature (273.15°K=0°C; 1K== 1C)

(Kelvin starts at the melting point of ice, where as 0 Kelvin is the lowest temperature they found.

s second time

m meter(AE)/ length  
 metre(BE)

kg kilogram mass

cd candela luminous intensity (*Leuchtstärke*)

mol mole amount of substance

**Beispiel:**

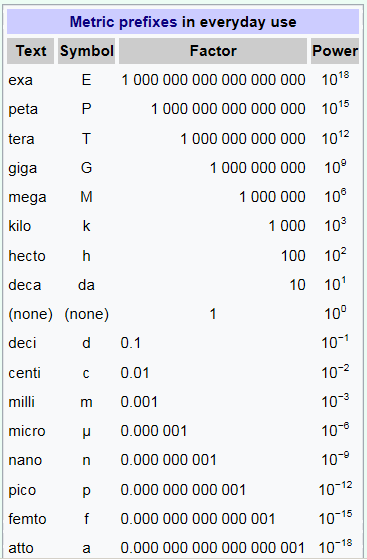
500k/w(Wärmeabgabe) \* 60mW(Energy) (500 kelvin per watt mal 60 milliWatt ) =

500k/W \* 60W \* 0,001

500k\*60\*0,001 = 30,000k

When we multiply this we get 30 kelvin and when we have a room temperature of 22°Celsius, than we know the additional (*zusätzliche*) participation at the end is 30 Kelvin

22°C + 30k (we remember Kelvin and Celsius is the same) 🡪 LED becomes 52°Celsius Degree



**we need:** Giga (because our computer has giga hearts), Mega (because Arduino has some mega hearts), kilo, milli, micro(μ,is millions (reverse of million)), nano(one nano second 1n=0,001 μs (microsecond) = 0,000001ms (milisecond) = 0,000000001s (second)).

1min =60s, 1h=60min=3600s.

1s= 1 000 000 000ns

**Beispiel:**

Quantity: (*physikalische Größe*)*:* Length, L

Comprises of (*besteht aus*):absolute measure (*Maßzahl*): 1785.53 and measurement unit (*Maßeinheit*): 1m.

🡪 l = 1785.53 \* 1m = 1785.53m = 1.78553\*10^3m=1.78553km

All calculation rules for numbers are applicable for all measurement units!

(*Alle Gesetzte der Mathematik kann man bei den Maßeinheiten auch verwenden*)

If L(ength) is the quantity then l(enght) is the variable for it)

**Movement**

**- Different between rowing and movement:**

Rowing is periodically and the **movement is continuously.**

Three major types of movement:

- **linear movement**, force-less- (*Kraftlose*), un-accelerated (*unbeschleunigte*) movement, we can measure the distance

- **periodic movement** you can count

- by a force **accelerated(***beschleunigt***) movement**

you can do counts and measure the lengths.

**Interpolation between counts:**

l(t) = const \* t // l in depends of the time is a constant times the time

we combined the linear movement and the periodic movement and the counting of the periodic movement we call time.

**What is constant**?

We make a triangle and use the delta length (l2-l1) and the same thing with the time: delta t (t2-t1) and than the constant is delta l divide by delta t

const = (l2[m]-l1[m])/(t[s]2-t[s]1) = 𝑣 [m/s] 🡨 Steigung (average velocity (*Durchschnittsgeschwindigkeit*))

Definition: **Time**

Time is a mathematical construct to describe movement in a formal way.

**What is time in our computer system?**

We develop applications and sometimes we need a time in the application: if I set an order, I need a timestamp and we get the time from the operating system.

We have some functions that gives us the actual world time and we are able to set the computer time.

We need times based execution for example jobs (every night a back up, or the upgrade on the smartphone..). we need some delay functions. Sometimes we need some sequences so we need some timer and timestamps.. this is not suitable (geeignet) for real time systems.

Timer in a computer: we need high resolution and high accuracy (*Richtigkeit*) and we need some software that is able to handle all the real time stuff, strategies, tasks (avoid system tasks that are interrupting our real time tasks), libraries and we must be able to handly concurrency (*Nebenläufigkeit*), because it does not make sense to have a real time system for every task, but one system that handle different tasks

**So:**

We have to deal with concurrency and we have to deal with scheduling algorithem

**Concurrency:**

- computer become faster and faster.. your client says you have to be ready in x or at x time.

Well, you can do some stuff in parallel. Maybe you have two CPUs.

Example: binomial function: you can do this on one core but you can do this in 2 processes.

**Sequential:** (first you do this and than this..) and when it runs on the computer it is a process.

**Parallel:** you can do it in parallel. This is probably faster. But we can get some issues, because the next calculation has to wait, until those has completed. So we get logical dependencies. So we make a block design. We divide our program into logical blocks.

Staring time is the possibility of direction or a synchronization.

**Issues with concurrency**:

- **Deadlocks:** one waits for another

- **livelocks:** one triggers the other and vice versa (*und umgekehrt*) (distributed endless loop)

- **race conditions:** the output is dependent on the sequence or timing of other uncontrollable events

**What is Scheduling**?

Scheduling is the decision how the process are assigned to run (*wie Prozesse aufgeführt warden sollen*) on available CPU(s) or core(s)

**Goals of scheduling:**

- minimizing the average response time: (e.g. interactive systems)

- maximizing throughput (*Datenrate, Verarbeitungsmenge*): e.g. server

-maximizing processor load: e.g. super computer

-fairness: fair utilization (*Nutzung, Verwendung*) of resources

- compliance (*Beachtung*) with deadlines

**Constraints:**

- Deadline have to be met

- tasks have different importance (priorities) 🡪 no fairness!

- priorities may change over time

- low response time is not enough, deadline has to be guaranteed

**Real-Time Clock:**

-high resolution (*hohe Auflösung*)

-high accuracy (*hohe Genauigkeit*)

-smooth time adjustment (*reibungslose Zeiteinstellung*)

**Real-Time Timer**

-high resolution

-high accuracy

**Software:**

-able to handle concurrency (*in der Lage, Nebenläufigkeit zu behandeln*)

-able to perform Real-Time Scheduling-Strategies (*kann Echtzeit-Scheduling durchführen*)

-able to control (avoid) system tasks

-libraries to handle Real-Time Hadware

Formularbeginn

Formularende

**Clocks and Timers**

**World Clock** (*Weltzeit Uhr*) give us just the actual UTC (Universal Time Coordinated) in our local time zone. This time is a **point in time** (*Zeitpunkt*). This point in time is an **absolute time!**

**Alarm Clock** (*Wecker*): you can set an absolute alarm time; the alarm time is a point in time.

There is a time-based trigger: The alarm clock compares the actual time with the alarm time.

When the alarm time is lower than or equal the actual time the alarm clock trigger a sound.

**Stop Watch** (*Stoppuhr*): There is nothing to set. (The reference point in time is set to 00:00,00s). There is a manual trigger to start the stop watch. The stop watch is counting upwards with the SI-second. It displays the relative time difference based on the trigger time point. There is a manual trigger to stop the stop watch 🡪 it stops the clock.

🡪 The display shows the time difference (duration (*Dauer*)) between:

**start event and stop event.**

**Stop watch with lap times** (*Rundenzeiten*):

Additional functionality: between start time and stop time you can have additional (*zusätzliche*) events 🡪 lap times.

When a lap is terminated the event initiates an additional time stamp. (*Wenn eine Runde beendet wird, löst das Ereignis einen zusätzlichen Zeitstempel aus*). This time stamp is a relative point in time. The calculation of the time difference between two absolute subsequent lap time-stamps (*zwei absolute Folgezeitstempel*) is the time difference (duration) of the preceding lap (*vorige Runde*). This duration is called lap time.

**Timer**you can set an relative alarm time. The alarm time is a time difference (duration).

There is a trigger: The trigger starts the time. The alarm is decreasing. When the alarm time is lower or equal zero the timer triggers a sound.

**Video Lecture:**

**What about Timekeeping?** (Zeitverwaltung)

It comprises: having a **clock** (Zeitgeber)

* absolute time Synchronization
* relative time (using Tics after systemstart)
* time measurement (Zeitmessung, Stoppuhr)

**Time Control** for services (Zeitsteuerung)

* to be used for frequent system Tasks (e.g. backups, update, virus scans, etc.)

**Time Monitoring** (Zeitüberwachung)

**Watchdog** (Laufzeit-Überwachung, abs, Wecker)

Timer (often hardware, relative time )

**Process Control**

(quasi-)parallel information processing of multi computing tasks (mehrere Rechenaufgaben) on a single- or multi-core-processors .

What are Tasks? There are two different Types of tasks: Threads and Processes

The difference between them is that the Processes have their own data segments.

**Video Lecture Multimedia Model**

**API**: application programming interface

The Main Duties (*Hauptaufgaben*) of **the Operating System** is

- Interface between hardware and user-programs

- Interface between hardware and partner systems using a network

- **Internal Resource Management:** provides a Process-/Processor-Managementm

(mainly for the CPU), it provides a Memory-Management (mainly for the RAM),

and it is responsible for the I/O management (it means mainly device- and file management).

**Timekeeping:**

(Not limited to the OS)

It comprises:

- having a **clock** (*Zeitgeber*)

🡪 absolute time **Synchronization**

🡪 relative time (using **Tics** after systemstart)

🡪 time measurement (*Zeitmessung, Stoppuhr (stopwatch)*)

- **Time Control** for servies (*Zeitsteuerung*)

to be used for frequent system Tasks (e.g. backups, updates, virus-scans, etc).

- **Time Monitoring (***Zeitüberwachung***)**

- **Watchdog (***Laufzeit-Überwachung, abs. Wecker***)**

- **Timer (**oftern hardware, relative time**)**

**Process Control**

There are lot or many tasks to perform by a computer.

**- What are tasks?**

There two different typs of task:

- one are **Threads** and the other are **Processes**

Both can perform competitional tasks (*beide können konkurrierende Aufgaben ausführen*)

But the main difference is, that processes have their own data segments.

**Different scheduling strategies:**

On the monocore you have **FCFS/FIFO**, which means: **first-come-first-serve/first-in-first-out.**

We have priorities time slices.

The other strategy is **EDF (earliest deadline first)**

**Requirements for a Realtime Operating System:**

🡪 time services : *absolute and relative clocks, timer, timeouts*

🡪 defined reaction times

🡪 realtime- compliant (*kompatible*) scheduling (*Terminplanung*)

🡪 realtime- compliant inter process communication (IPC) (*Echtzeit-kompatible Kommunikation zwischen Prozessoren*)

**Video: Real-Time Model 1: Points in Time, Time Duration**

**Time beam** represents everything what we need for real time systems. It’s a model for the time.

We start with an event and call it A. We need a zero point (reference point). We call it t0, it should mean that the time is zero in this case. But what is A? we have to define a Unit, we say [t] for time and its unit is seconds [s]. t0=0, t1=1,…, A=ta,…, B=tb,…,tn.

*„Eine Physikalische größe setzt sich immer zusammen aus einer Maßzahl und einer Maßeinheit. Dabei ist die Maßzahl eine dimensionslose Zahl, und die Maßeinheit sowas wie Sekunden, Meter, Kilogramm, Kilometer etc.. “.*

Than we can define a Time interval between the event A and B;

I[A,B] = [ta,tb]. 🡨 this is a section of the time beam. This is a time interval. How long is this time? when we have numbers, we can calculate.

We define the difference and called it delta (triangle) = [A,B]= delta [ta,tb] = delta tA,B

🡪positive difference : |tA-tB| = tb-ta, (ta≤tb)

*Maßeinheiten werden genauso gerechnet wie Zahlen, das heißt, Sekunden können gekürzt werden, Brüche können erweitert werden etc..*

**Execution Times**

**Job:** a Job J is a single CPU-time requirement to perform a computational sequence

**Execution time** (*Ausführungszeit*):

Execution time is the duration (*Dauer*) of a specific Job between request and Job completion (*Fertigstellung*)

- Relevant for **deadlines** is the maximum execution time of a program on a certain hadware.

- the Worst Case Execution Time (WCET) is the maximum limit of the variable execution time.

**Video: Real-Time Model1: JOBS**

We start with a time beam. When we have a single CPU we write on the y-axis 1.

Than we have a start time (ts) and a completion time (tc). The request for the calculation we call request Req = (delta texec,), exec = execution time, this is the time, which the CPU need to accomplish (*erreichen/vollenden*) the required calculation.

When we have just one job, than the start time is the request time. But when the CPU is already working for another calculation, the new requested task has to wait and than the start time and the request time are different. In this case we call it a job, it has a request time and an execution time. The time between the request and the start time is called waiting time (delta twait). the time between the request and the completion time is called response time (delta tresp). The response time has to fulfil time constraints. The time constraints are defined by a deadline. In this case it’s a relative deadline; it is the time between the request and the time, where the calculation has to be accomplished.

When we add a the request time and the relative deadline, we find the absolute deadline.

The time between the completion time and the absolute deadline is called slack time(*Schlafzeit*) (delta tslack).

**Tasks**

**- Periodic task:** when the time of reappearing is constant, the task is called a periodic task.

**- Sporadic task:** when the recurrence (*Wiederholung*) is random but **there is a minimum difference between two job**s of the same task, the task is called a sporadic task.

**-aperiodic task:** when the recurrence is random but there **is no minimum time difference between two** jobs of the same task, means they may even appear at the same time, the task is called an aperiodic task.

**- Real Time scheduling:**

**- without Priority**: FIFO (First in first out), Round Robin, SJF (Shortest Job First), SRT (Shortest Remaining (Execution-)Time)

**- static Priority**: - EDD (Earliest Due Date First)

- RMS (Rate Monotonic Scheduling)

- DMS (Deadline Monotonic Scheduling)

**- Dynamic Priority:** - EDF (Earliest Deadline First)

- LRT (Latest Release (*Freigabe*)Time)

**Scheduling planning**

**Schedule:**

- A schedule of a set of jobs is called **feasible** (*zuverlässig/machbar*) when each job can be completed with its individual Deadline.

- **to schedule** means to decide, which process will be processed in which time frame.

- a scheduling algorithm is called **optimal** if it is able to create a schedule in those cases in which feasible schedule exists.

**EDF Earliest Deadline First:**

- has dynamic priorities

- each tasks priority is calculated depending on deadline:

- an executable (*ausführbarer*) task with shorter deadline has always a higher priority than the other tasks

- an executable task with higher priority will always interrupt a task with lower priority

- a task with the same priority is not interrupted.

**RMS Rate Monotonic Scheduling:**

- is a priority based preemptive scheduling, in which the threads with the lower periods should be given higher priorities and vice versa.

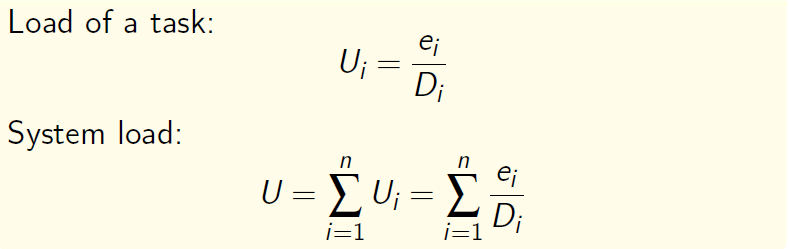
- the priority is solely (*einzig und allein*) determined by its period only with the rule

**“lower the period higher the priority and vice versa”**

**🡪** if a system can not be schedulable by RMS, the system can not be scheduled with any other static priority assignment (*Zuordnung*).

**- Load test:**

ei = execution time, Di = expiration time (*Ablaufzeit*) of **dead**, Pi = Di



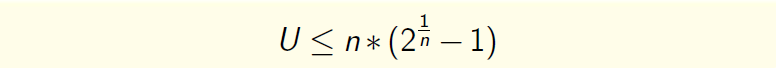
Load U = 1 means the processor never idles (*nie im Leerlauf*)

Load U > 1 there is no feasible schedule

Load U < 1 means: this test does not exclude that a feasible schedule may exists (*dieser Test schließt nicht aus, dass ein ausführbarer Zeitplan existiert* ).

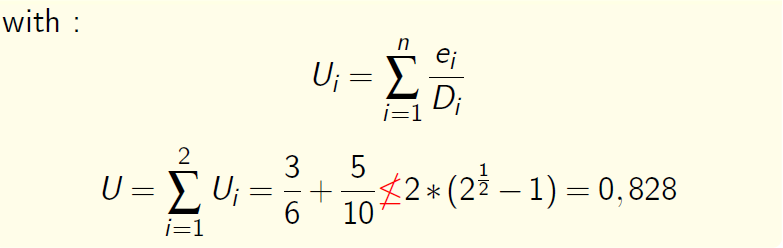
**- Test for Rate Monotonic schedule (RMS):**

*Schedulability Test:*



Where n ist he number of periodic tasks.

**Example**: (n=2: p1=(6,3) and p2 (10,5) [(Di, e)])



**Technical**

If a task is non-interruptible it can have only three states:

- **Ready:** (waiting to start) ready tasks are those that are able to execute (they are not in the blocked suspended state) but are not currently executing because a different task of equal or higher priority is already in the running state.

- **Running:** when a task is actually executing it is said to be in the running state.

- **Completed**

**Process:**

process is that executing unit of a computation, which is controlled by some processes of the OS. A running application program can be said to consist of a number of processes.

A process is a sequence of instructions. It is a dynamic entity that is a program in execution. It is an active part. During execution it gives the result. A process is stored in memory.

**Latency Time:**

Is the time that is needed for computer internal processing (operating system and processor internal) and leads to couple of time delay (*Zeitverzögerung*) between an initiation and the start of the requested task.

*(Ist die Zeit, die für die computerinterne Verarbeitung benötigt wird (Betriebssystem und Prozessor intern) und führt zu einer gewissen Zeitverzögerung zwischen einer Initiierung und dem Start der angeforderten Aufgabe.)*

There are: **- Interrupt-Latency**

**- Task-Latency**

**- Kernal-Latency**

**- Preemption Day (***Verdrängungszeit***)**

**Thread:**

Two processes can run at the same time, but they do not share memory. Suppose we provided software entities that could run at the same time but also share memory.

Such entities exist in most of the actual OS. They are called **threads**.

A thread has some of the characteristics of a process, but it is possible to have threads sharing the same memory space.

**Difference Process / Thread**

**Process Thread**

- Process is considered heavy (*gilt als schwerer*) - Thread is considered light weight

- Unit of Resource Allocation and of Protection - Unit of CPU utilization

- Process creation is very costly in terms - thread creation is very economical

of resources

- Program execution as process is relatively slow - Programs executing using threads are

- Process cannot access the memory belonging to comparatively faster

another process - thread can access the memory area

- Process switching is time consuming belonging to another thread within the

One Process can contain several threads same process.

- thread switching is faster

- one thread can belong exactly to one

Process.