Ravensburg-Weingarten University

Master Thesis

Implementierung eines Earliest First Deadline Scheduler in FreeRTOS auf einem STM32

Fabian Wicker 32235

12. Januar 2021

HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit „Implementierung eines Earliest First Deadline Scheduler in FreeRTOS“ selbständig angefertigt und mich nicht fremder Hilfe bedient habe. Alle Stellen, die wörtlich oder sinngemäß veröffentlichtem und unveröffentlichtem Schriftgut entnommen sind, habe ich als solche kenntlich gemacht.

Unterwaldhausen, den 12. Januar 2021

Fabian Wicker

HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Aufgabenbeschreibung

Ziel dieser Masterarbeit ist es, einen Earliest Deadline First (EDF)-Scheduler in Free Real-Time Operating System (FreeRTOS) auf einem STM32-Mikrocontroller zu integrieren. Die korrekte Funktion des EDF-Schedulers soll anhand zwei Testanwendungen garantiert werden. Folgende Aufgaben gehören zu dieser Masterarbeit:

- Implementierung des EDF-Schedulers in FreeRTOS
- Erstellung einer vom Benutzer steuerbare visuellen Testanwendung
- Integration einer Fast Fourier Transformation (FFT)-Berechnung

HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Abstract

Lückenfüller...

HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Inhaltsverzeichnis

HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Abkürzungen

z.B.	zum Beispiel
WLAN	Wireless Local Area Network
FPGA	Field Programmable Gate Array
DSP	Digital Signal Processor
EDF	Earliest Deadline First
FreeRTOS	Free Real-Time Operating System
FFT	Fast Fourier Transformation
RMS	Rate Monotonic Scheduling
DMS	Deadline Monotonic Scheduling
CPU	Central Processing Unit
WCET	Worst Execution Time
MQTT	Message Queuing Telemetry Transport
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
API	Application Programming Interface
CLI	Command Line Interface
RAM	Random-Access Memory

ROM	Read-Only Memory
IRQ	Interrupt Request
ISR	Interrupt Service Routine
i.A.a	in Anlehnung an
FIFO	First Input First Output

HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Formelverzeichnis

1 Grundlagen

Lückenfüller

```
    createEDFTask(              // Task Creation
        testFunc,               // Pointer to task entry function
        "Test Function",        // A descriptive name for the task
        300,                    // The number of words to allocate
        NULL,                   // A value that will be passed into the task
        1,                      // WCET in ms
        5,                      // Period of Task in ms
        4 );                    // Deadline of Task in ms
```

Programmcode 1: FreeRTOS Task creation and deletion

1.1 Echtzeitsysteme

Unter Echtzeit wird die Anforderung bezeichnet, dass innerhalb einer kürzesten definierten Zeitspanne die geforderte Aufgabe korrekt ausgeführt wird. Echtzeitsysteme sind Systeme, welche diese Anforderungen erfüllen und kommen in diversen Technikgebieten zur Anwendung, wie zum Beispiel (z.B.) in Signalstellanlagen, Robotik und Motorsteuerungen. Da nicht bei allen Systemen eine Einhaltung der vorher definierten Zeitspanne von Nöten ist, wird die Echtzeitanforderungen in drei Kategorien unterteilt. Dabei wird bei Echtzeit unter drei verschiedenen Kategorien unterschieden:

- **Feste Echtzeitanforderung** führt bei einer Überschreitung der vorher definierten Zeitspanne zum Abbruch der Aufgabe.
  - **Beispiel:** Verbindungsaufbau eines Smartphones in das lokale Wireless Local Area Network (WLAN).
- **Weiche Echtzeitanforderung** kann die vorher definierte Zeitspanne überschreiten, arbeitet die Aufgabe normalerweise aber schnell genug ab. Die definierte Zeitspanne kann auch als Richtlinie bezeichnet werden, die aber nicht eingehalten werden muss.
  - **Beispiel:** Druckgeschwindigkeit eines Druckers.
- **Harte Echtzeitanforderung** garantiert die Erfüllung der Aufgabe zu der vorher definierten Zeitspanne.
  - **Beispiel:** Bremspedal eines Autos.

Echtzeit beschreibt somit das zeitliche Ein- bzw. Ausgangsverhalten eines Systems, allerdings nichts über dessen Realisierung. Dies kann je nach Anforderung das Echtzeit-

system rein auf Software auf einem normalen Computer implementiert sein, jedoch wird bei harter Echtzeitanforderung oftmals eine reine Hardware-Lösung mit speziellen Architekturen in Hard- und Software wie z.B. Mikrocontroller-, Field Programmable Gate Array (FPGA)- oder Digital Signal Processor (DSP)-basierte Lösungen verwendet. Bei der Implementierung eines Echtzeitsystems wird unter drei verschiedenen Umsetzungen unterschieden:

- **Feste periodische Abarbeitung:** Es wird nur eine Aufgabe, wie z.B. bei der Umwandlung von analogen Signalen zu digitalen Signalen, mit einer fixen Frequenz f abgearbeitet, welche die Reaktionszeit $f = \frac{1}{Reaktionszeit}$ erfüllt.

- **Synchrone Abarbeitung:** Im Gegensatz zur festen periodischen Abarbeitung können bei diesem Ansatz mehrere Aufgaben, wie z.B. das Abfragen mehrere Sensoren und einer unterschiedlichen Reaktion darauf, erfüllt werden. Allerdings muss dabei die kleinste geforderte Reaktionszeit unter den Aufgaben die hälfte der maximalen Laufzeit für den Gesamtdurchlauf aller Aufgaben betragen. Dieser Ansatz wird vor allem für weiche Echtzeitsysteme verwendet, weil je nach Komplexität des Systems, das Vorhandensein mehrere Codepfade oder das Warten auf Ein- oder Ausgangssignalen mit unterschiedlicher Ausführungszeit ein Nichtdeterminismus besteht.

- **Prozessbasierte Abarbeitung:** Bei diesem Ansatz können, wie auch bei der Synchronen Abarbeitung mehrere Aufgaben erledigt werden, jedoch mit viel höheren Komplexität. Dabei laufen in der Regel verschieden Prozesse gleichzeitig und mit unterschiedlicher Priorität ab, geregelt durch das Echtzeitbetriebssystem. Die minimale Reaktionszeit definiert sich durch die Zeitdauer für einen Wechsel von einem Prozess niederer Priorität zu einem Prozess höherer Priorität, anschließend beginnt erst die Abarbeitung des Prozesses mit der höheren Priorität. Durch die Unterbrechung eines Prozesses niedriger Priorität zu einem Prozess höherer Priorität wird für diesen die Erfüllung einer harten Echtzeitanforderung erzwungen, dies wird auch als Präemptives Multitasking bezeichnet. Der Wechsel wird dann eingeleitet, wenn ein definiertes Ereignis eintritt, z.B. durch den internen Timer des Prozessors oder einem externen Trigger (Interrupt) wie z.B. ein anliegendes Signal.

(Siemers, 2017; Wikipedia, 2020a)

1.2 Scheduler

Als Scheduler (zu Deutsch Steuerprogramm) wird eine Logik bezeichnet, welche die zeitliche Ausführung von mehreren Prozessen steuert und wird als präemptiver oder kooperativer Scheduler in Betriebssystemen eingesetzt. Bei einem kooperativen Scheduling wird einem Prozess die benötigten Ressourcen übergeben und wartet, bis der Prozess vollständig abgearbeitet wurde. Im Gegensatz zu einem kooperativen Scheduler kann ein

präemptiven Scheduler ein Prozess die Ressourcen vor der Fertigstellung entziehen, um zwischenzeitlich diese anderen Prozessen zuzuweisen (Wikipedia, 2020d).

Allgemein lassen sich die unterschiedliche Scheduler-Systeme in drei Rubriken unterteilen (Wikipedia, 2020b):

- **Stapelverarbeitungssysteme**: Ankommende Prozesse werden nach Eingangszeit in eine Reihe (Queue) angeordnet, die dann nacheinander abgearbeitet wird.
- **Interaktive Systeme**: Eingaben von Benutzen sollten schnellstmöglich abgearbeitet werden, weniger wichtige Aufgaben wie z.B. die Aktualisierung der Uhrzeit werden unterbrochen oder erst im Nachhinein abgearbeitet.
- **Echtzeitsysteme**: Das Echtzeitsystem garantiert die Fertigstellung des Prozesses innerhalb einer definierten Zeitspanne, sofern die CPU Auslastung nicht über 100% beträgt.

Für die Anordnung selbst, wann welcher Prozess ausgeführt wird, gibt es verschiedene Arten von Scheduler.

1.2.1 RMS

RMS ist ein präemptives Scheduling-Verfahren, welches die Prioritäten einzelner Prozesse nach deren Periodenlänge statisch anordnet. Je niedriger die Periode eines Prozesses ist, je höher ist dessen Priorität. Mit Hilfe der Gleichung 1 kann die maximale Menge an Jobs, die mit RMS verarbeitet werden können, berechnet werden. Durch eine zunehmende Anzahl von Prozessen nähert sich die Grenze dem Wert von $ln(2) \approx 0.693$, welches eine maximale Auslastung von 69,3% bedeutet. Die maximale Auslastung bezieht sich dabei auf die CPU-Auslastung, alle anderen Ressourcen wie z.B. Arbeitsspeicher werden als unbegrenzt angesehen. Sofern die maximale Auslastung nicht übertreten wird, verpasst kein Prozess die dazugehörige Deadline (Wikipedia, 2020c).

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \cdot \left(\sqrt[n]{2} - 1 \right) \tag{1}$$

U	CPU-Auslastung
C_i	Ausführungszeiten
T_i	Periodenlänge
n	Anzahl der Jobs

Formel 1: RMS Berechnung der CPU-Auslastung (Wikipedia, 2020c)

1.2.2 DMS

Analog zu RMS arbeitet DMS präemptiv prioritätsbasiert, wobei bei DMS der Prozess mit der kürzesten Deadline die höchste Priorität bekommt. Falls bei der Abarbeitung des aktuellen Prozesses ein neuer Prozess mit einer höheren Priorität eintrifft, wird der aktuelle Prozess unterbrochen und der Prozess mit der höheren Priorität bekommt die Rechenzeit. Die Berechnung der maximalen Auslastung erfolgt mit der gleichen Formel 1, dies führt zur gleichen maximalen Auslastung von 69,3%, bei der keine Deadline überschritten werden (Wikipedia contributors, 2021).

1.2.3 EDF

Im Unterschied zu DMS wird bei EDF beim Eintreffen eines Prozesses mit höherer Priorität der aktuelle Prozess mit niedriger Priorität nicht unterbrochen, auch sind die Prioritäten nicht statisch, sondern werden bei jedem Scheduling-Vorgang neu vergeben. Ein Context Switch, das heißt ein Wechsel zwischen den Prozessen, findet nur vor Beginn eines Prozesses oder am Ende eines Prozesses statt. In Zuge dieser Masterarbeit wird ein Context Switch nur nach Beendigung eines Prozesses ausgelöst, sofern ein anderer Prozess eine höhere Priorität erhalten hat. Im Gegensatz zu RMS und DMS kann bei EDF der Scheduler die CPU bis auf 100% betragen, dabei darf die Zeitspanne der einzelnen Prozesse bis zur Deadline nur jeweils größer oder gleich der Periode des Prozesses sein (Wikipedia contributors, 2021).

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2}$$

U	CPU-Auslastung
C_i	Ausführungszeiten
T_i	Periodenlänge
n	Anzahl der Jobs

Formel 2: EDF Berechnung CPU-Auslastung (Wikipedia contributors, 2021)

1.3 FreeRTOS

FreeRTOS ist ein Echtzeitbetriebssystem für eingebettete Systeme, welches unter der freizügigen Open-Source Lizenz MIT steht. Für eine leichte Wartbarkeit wurde FreeRTOS weitestgehend in C entwickelt, außerdem ist der Scheduler des Betriebsystems zwischen präemptiver und kooperativer Betrieb konfigurierbar um verschiedene Einsatzzwecke abzudecken (Wikipedia, 2019). Des Weiteren wurde FreeRTOS auf über 40 Mikrocontroller-Architekturen portiert („FreeRTOS“, n. d.), um eine größere Bandbreite zu erreichen.

FreeRTOS hat wenig Overhead und der Kernel unterstützt Multithreading, Warteschlangen, Semaphore, Software-Timer, Mutexes und Eventgruppen („FreeRTOS Features“, n. d.). Des Weiteren stellt FreeRTOS unter der Bezeichnung ”FreeRTOS Plus”verschiedene Erweiterungen zur Verfügung, welche erweiterte Funktionen bereitstellen (FreeRTOS, 2020b).

- **FreeRTOS + TCP**: Socketbasiertes TCP/UDP/IP Interface.
- **Application Protocols**: MQTT und HTTP Anwendungsprotokolle für IoT.
- **coreJson**: Effizienter Parser für JSON
- **corePKCS#11**: Verschlüsselungs API-Layer
- **FreeRTOS + IO**: Erweiterung für Peripheriegeräte
- **FreeRTOS + CLI**: Effiziente CLI-Eingaben

Im Zuge dieser Masterarbeit wurde als einziges der Stack ’FreeRTOS + TCP’ für die Verbindung zwischen Computer und Mikrocontroller.

1.3.1 FreeRTOS Scheduling Richtlinie

Der FreeRTOS Scheduler arbeitet prioritätsbasiert und beinhaltet standartmäßig hauptsächlich zwei Eigenschaften:

- **Time Slicing Scheduling Policy**: Gleich priorisierte Tasks erhalten die gleiche Anteile an CPU-Zeit, dieses Verfahren ist auch als ’Round-Robin Algorithmus’ bekannt.
- **Fixed Priority Preemptive Scheduling**: Es wird immer die Task ausgeführt, mit der höchsten Priorität. Das bedeutet eine niedriger priorisierte Task bekommt nur CPU-Zeit, wenn die höher priorisierten Tasks nicht den ’ready’-Status besitzen. Teilen sich zwei Tasks gleichzeitig die höchste Priorität, tritt die Time Slicing Scheduling Policy in Kraft.

Um einen oder beide dieser Eigenschaften abzuschalten, kann dies unter der Datei ’FreeRTOSConfig.h’, wie in Programmcode 2 dargestellt, abgeändert werden (Microcontrollerslab, 2021).

```
1    #define configUSE_PREEMPTION                    ( 1 )
2    #define configUSE_TIME_SLICING                  ( 1 )
```

Programmcode 2: FreeRTOS Scheduling Policy Properties

1.3.2 Overhead und Footprint

Die Dauer eines Context Switches, auch Taskwechsel genannt, zwischen zwei tasks ist abhängig von dem Port, dem Compiler und der Konfiguration von FreeRTOS. FreeRTOS selbst gibt ein Beispiel anhand einer Cortex-M3 CPU eine Taskwechsel Zeit von 84 CPU Zyklen an (FreeRTOS, 2020c). Der minimale Random-Access Memory (RAM)- und Read-Only Memory (ROM)-Footprint wird zwischen 6kByte und 12kByte angegeben (FreeRTOS, 2020a).

1.3.3 Task States

In FreeRTOS erzeugte Tasks können vier verschiedene Zustände haben (FreeRTOS, 2020d):

- **Running**: Wird gerade ausgeführt.
- **Ready**: Ist bereit zur Ausführung.
- **Blocked**: Wartet auf ein Event, kann nicht ausgeführt werden.
- **Suspended**: Wurde temporär deaktiviert.

Ein Ablaufdiagramm der einzelnen Zustände und deren Wechsel mit Hilfe von FreeRTOS Funktionen wird in Abbildung 1 dargestellt.

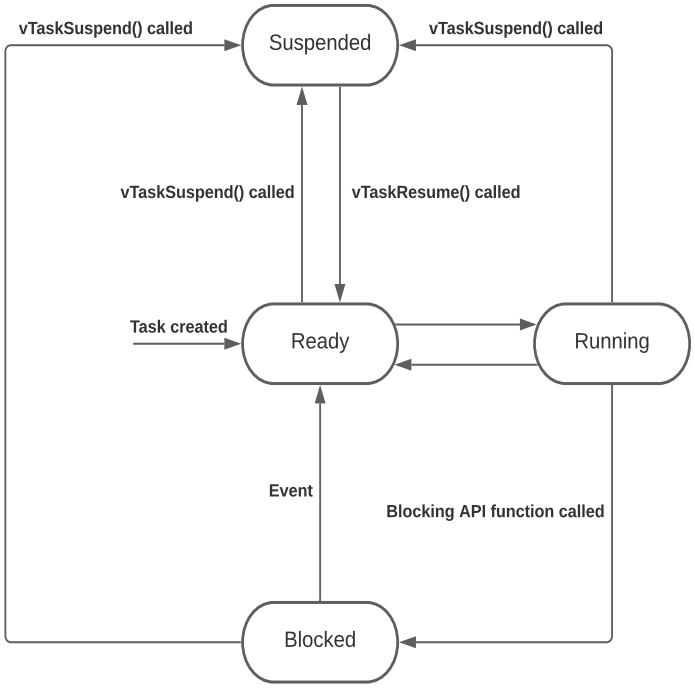

Figure 1: FreeRTOS Task States in Anlehnung an (i.A.a): (FreeRTOS, 2020d)

1.3.4 Interrupts

Interrupts werden durch ein Ereignis ausgelöst, z.B. von einem bestimmten Wert eines Timers und führt zur einer Unterbrechung der aktuellen Programmausführung, um in der Regel kurzen, aber zeitlich kritischen, Vorgang abzuarbeiten. Nach der Unterbrechungsanforderung (Interrupt Request (IRQ)) führt der Prozessor eine Unterbrechungsroutine (Interrupt Service Routine (ISR)) mit erweiterten Privilegien ausgeführt. Im Anschluss wird der vorherige Zustand des Prozessors wiederhergestellt und die vorherige Ausführung an der unterbrochenen Stelle fortgesetzt (Wikipedia, 2021).

1.3.5 FreeRTOS Sys-Tick

Der FreeRTOS Sys-Tick, auch System Tick genannt, ist fundemental für die Taskwechsel in FreeRTOS, dieser Interrupt wird standartmäßig jede Millisekunde aufgerufen. Dieser Wert bietet eine gute Balance zwischen Taskgeschwindigkeit und Overhead von Taskwechsel. Bei jedem Sys-Tick wird die FreeRTOS Funktion 'vTaskSwitchContext()' aufgerufen, diese überprüft ob eine höher priorisierte Task unblocked, also ausführbar

geworden ist und falls dies zutrifft wird ein Taskwechsel auf die höher priorisierte Task durchgeführt.

1.3.6 FreeRTOS Queues

Queues, zu Deutsch Warteschlange, werden oft, auch außerhalb von FreeRTOS für die sichere Kommunikation zwischen verschiedenen Tasks und/oder Interrupts verwendet. Diese Implementierung von FreeRTOS vermeidet Fehler wie ein Deadlock, z.B. eine höher priorisierte Task wartet auf einen Wert einer niederen priorisierten Task, wobei die niedere priorisierte Task nie ausgeführt wird. Dabei wird oft das Prinzip First Input First Output (FIFO) verwendet, um eine zeitliche Abarbeitung der Daten abzuarbeiten.

1.4 Periodische Tasks

Anwendungen auch Prozesse oder Tasks genannt, werden in Echtzeitsystemen oft periodisch, z.B. das Abrufen eines Sensorwertes, ausgeführt. Periodische Tasks setzen sich aus folgenden Parametern zusammen:

$$T_i(\Phi_i, P_i, e_i, D_i) \tag{3}$$

T_i	Task Nummer
Φ_i	Phase
e_i	Ausführzeit (Worst Execution Time (WCET))
D_i	Deadline

Formel 3: Parameter von periodischen Tasks

Mit diesen vier Parameter jeder einzelnen periodischen Tasks können die verschiedenen Scheduler-Verfahren die Ausführung der einzelnen Tasks planen und umsetzen. Im Zuge der Aufgabenstellung bei dieser Masterarbeit wurden keine Tasks mit einer Phasenverschiebung verlangt, daher kann der Parameter $\Phi_i = 0$ angesehen werden.

1.5 UDP

Für die Erfüllung der Aufgabe dieser Masterarbeit musste ein Kommunikationsmittel zwischen Computer und Mikrocontroller ausgesucht werden, welches verschiedene Eigenschaften erfüllt:

- **verbindungslos**: Der Sender sendet Daten ohne eine Bestätigung des Eingangs der Daten zu bekommen. Dies hat den Vorteil einer latenzärmeren Datenübertragung.

- **wenig Overhead**: Das Empfangen und Senden sollte so wenig wie möglich Ressourcen, wie CPU-Zeit, blockieren.

- **hohe Übertragungsrate**: Es sollten so viele Daten wie möglich, innerhalb einer bestimmten Zeit übertragen.

Aus diesen Gründen wurde das Netzwerkprotokoll UDP, welches heutzutage in vielen Anwendungen, wie z.B. **IP!** (**IP!**)-Telefonie und Videostreams, verwendet. Für den Verbindungsaufbau verwendet UDP Ports, welche Teil einer Netzwerk-Adresse darstellen, um versendete Daten dem gewünschten Programm an der Gegenstelle zukommen zu lassen (Wikipedia, 2020e).

2 Plattform

...

Figures

Tables

Literatur

FreeRTOS [[Online; Stand 9. November 2020]]. (n. d.). %5Curl%7Bfreertos.org%7D

FreeRTOS. (2020a). FreeRTOS Footprint [[Online; Stand 11. Januar 2021]]. https://www.freertos.org/RTOS.html

FreeRTOS. (2020b). FreeRTOS Libraries [[Online; Stand 11. Januar 2021]]. https://www.freertos.org/FreeRTOS-Plus/index.html

FreeRTOS. (2020c). FreeRTOS Overhead [[Online; Stand 11. Januar 2021]]. https://www.freertos.org/FAQMem.html#ContextSwitchTime

FreeRTOS. (2020d). FreeRTOS Task States [[Online; Stand 11. Januar 2021]]. https://www.freertos.org/RTOS-task-states.html

FreeRTOS Features [[Online; Stand 9. November 2020]]. (n. d.). %5Curl%7Bfreertos.org/features%7D

Microcontrollerslab. (2021). FreeRTOS Scheduler: Learn to Configure Scheduling Algorithm [[Online; Stand 11. Januar 2021]]. https://microcontrollerslab.com/freertos-scheduler-learn-to-configure-scheduling-algorithm/#:~:text=FreeRTOS%20Scheduling%20Policy,-FreeRTOS%20kernel%20supports&text=In%20this%20algorithm%2C%20all%20equal,before%20a%20low%20priority%20task.

Siemers, P. D. C. (2017). Echtzeit: Grundlagen von Echtzeitsystemen [[Online; Stand 9. November 2020]]. %5Curl%7Bhttps://www.embedded-software-engineering.de/echtzeit-grundlagen-von-echtzeitsystemen-a-669520/%7D

Wikipedia. (2019). FreeRTOS — Wikipedia, Die freie Enzyklopädie [[Online; Stand 9. November 2020]]. %5Curl%7Bhttps://de.wikipedia.org/w/index.php?title=FreeRTOS&oldid=192678143%7D

Wikipedia. (2020a). Echtzeitsystem — Wikipedia, Die freie Enzyklopädie [[Online; Stand 9. November 2020]]. %5Curl%7Bhttps://de.wikipedia.org/w/index.php?title=Echtzeitsystem&oldid=200724790%7D

Wikipedia. (2020b). Prozess-Scheduler — Wikipedia, Die freie Enzyklopädie [[Online; Stand 11. Januar 2021]]. https://de.wikipedia.org/w/index.php?title=Prozess-Scheduler&oldid=205225114

Wikipedia. (2020c). Rate Monotonic Scheduling — Wikipedia, Die freie Enzyklopädie [[Online; Stand 11. Januar 2021]]. https://de.wikipedia.org/w/index.php?title=Rate_Monotonic_Scheduling&oldid=196180419

Wikipedia. (2020d). Scheduling — Wikipedia, Die freie Enzyklopädie [[Online; Stand 9. November 2020]]. %5Curl%7Bhttps://de.wikipedia.org/w/index.php?title=Scheduling&oldid=202856507%7D

Wikipedia. (2020e). User Datagram Protocol — Wikipedia, Die freie Enzyklopädie [[Online; Stand 12. Januar 2021]]. https://de.wikipedia.org/w/index.php?title=User_Datagram_Protocol&oldid=204278447

Wikipedia. (2021). Interrupt — Wikipedia, Die freie Enzyklopädie [[Online; Stand 11. Januar 2021]]. https://de.wikipedia.org/w/index.php?title=Interrupt&oldid=207161465

Wikipedia contributors. (2021). Earliest deadline first scheduling — Wikipedia, The Free Encyclopedia [[Online; accessed 11-January-2021]]. https://en.wikipedia.org/w/index.php?title=Earliest_deadline_first_scheduling&oldid=999660324