DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Programmentwurf

Entwicklung eines rudimentären Zeitscheibensystems in C

Studiengang: Embedded Systems

Studienrichtung: Automotive Engineering

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

von

Fabian Klotz

Abgabedatum:	09.04.2023
Kurs:	TSA22
Studiengangsleiter:	Prof. Dr. Ing. Florian Leitner-Fischer

Erklärung

gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017 in der Fassung vom 25.07.2018.

Ich versichere hiermit, dass ich meinen Programmentwurf mit dem Thema:

Entwicklung eines rudimentären Zeitscheibensystems in C

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Weingarten, den 28. März 2023

_____

Fabian Klotz

Kurzfassung

Im Rahmen des Programmentwurfs der Vorlesung Programmieren soll ein rudimentäres Zeitscheibensystem entworfen werden. Hierbei sind einige Anforderungen zu erfüllen, wie zum Beispiel eine sehr große Variabilität des Programms, sodass mit möglichst wenig Aufwand neue Tasks angelegt, oder zu anderen Zykluszeiten umgesetzt werden können.

Das vorliegende Programm basiert hierbei auf dem Konzept des Threadings um die zeitliche Abfolge zu organisieren. In den einzelnen Tasks können Funktionen mit einer bestimmten Priorität, also Vorgabe der auszuführenden Reihenfolge, konfiguriert werden. Außerdem können sich die einzelnen Funktionen selbst für die Ausführung in einem Task mit einer gewissen Priorität registrieren. Braucht eine Funktion länger für die Ausführung wie die angegebene Zykluszeit, wird der Task erst dann neu gestartet, wenn er regulär wieder an der Reihe ist.

Abschließend wird der Worst-Case Jitter der einzelnen Tasks berechnet.

Inhaltsverzeichnis

1 Konzeptentwurf

Um ein passendes Konzept für die Umsetzung des Programms zu finden, wurden im Voraus einige Überlegungen, vor allem zur Steuerung der zeitlichen Abfolge angestellt. Mit dem Konzept des Threadings kann sowohl die zeitliche Abfolge, als auch der Aufruf der einzelnen Funktionen in den Tasks effizient gesteuert werden.

Die Abbildung soll Threading bildhaft darstellen[1] .

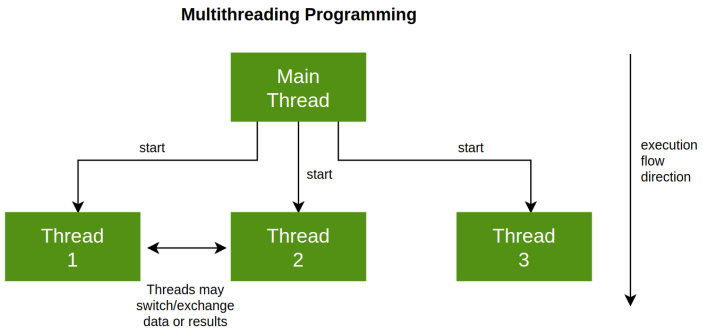

Abbildung 1.1: Threading

Der „Main Thread“ startet mehrere untergeordnete Threads. Alle Threads greifen auf die gleichen Ressourcen und globalen Variablen zu.

Am Anfang des Programms wird definiert, wie viele Tasks (bzw. Threads) es geben soll. Die genannte Anzahl Threads wird draufhin gestartet. Jeder Thread bekommt eine eigene Verkettete Liste, in der die einzelnen ausführbaren Funktionen mit einer Priorität gespeichert werden. Wird eine Funktion zur Liste hinzugefügt, sortiert ein Algorithmus die Funktionen nach Priorität. Die Threads durchlaufen alle die gleiche Funktion, in

[1] https://www.baeldung.com/wp-content/uploads/sites/4/2020/07/multithreading.png

der durch die Listen iteriert wird und die Funktionen abgearbeitet werden. Anschließend wird der Thread so lange schlafen gelegt, bis der nächste reguläre Zyklus ansteht. Durch die Verwendung von Mutex werden Race-Conditions verhindert. Die Threads laufen jeweils in einer Endlosschleife, damit das Programm erste terminiert wird, wenn es der User abbricht. Wird das Programm von außen terminiert, wird der von den Listen allozierte Speicher wieder freigegeben.

2 Umsetzung

2.1 pthread

Um erste Versuche mit Threading zu starten, wird ein Programm erstellt, das zwei Threads startet. Dabei wurde die Bibliothek 'pthread.h' verwendet.

`pthread` steht für ' POSIX Threads' und ist eine API (Application Programming Interface), die es ermöglicht, in C und C++ Threads zu erstellen und zu verwalten. POSIX steht dabei für "Portable Operating System Interface", was bedeutet, dass diese API auf vielen verschiedenen Betriebssystemen verfügbar ist und somit portabel ist. Pthreads werden unter UNIX-basierten Betriebssystemen wie MacOS oder Linux unterstützt.

Die `pthread`-API stellt eine Reihe von Funktionen bereit, mit denen Threads erstellt, gesteuert und synchronisiert werden können. Einige wichtige Funktionen der `pthread`-API sind:

- `pthread_create`: Diese Funktion erstellt einen neuen Thread und führt eine bestimmte Funktion aus, die dem Thread als Argument übergeben wird.

- `pthread_join`: Diese Funktion blockiert den aufrufenden Thread, bis der angegebene Thread beendet wurde.

Die `pthread`-API ist sehr leistungsfähig und wird häufig in Anwendungen eingesetzt, die viele gleichzeitig ausgeführte Threads erfordern, wie beispielsweise Serveranwendungen

oder Multimedia-Anwendungen[2].

Als Threadfunktion wird eine Funktion erstellt, die eine Liste der vorgegebenen Struktur erstellt und dann eine Endlosschleife bis zur Terminierung von außen durchläuft. In dieser Endlosschleife wird zuerst überprüft, ob in der Liste Funktionen zur Ausführung hinterlegt sind. Sind alle Funktionen abgearbeitet wird der Thread so lange schlafen gelegt, bis der nächste periodische Zyklus der angegebenen Zykluszeit ansteht, um Systemressourcen zu sparen und den Jitter zu minimieren. Die Knoten der jeweiligen Listen haben folgende Parameter:

- Einen Funktionspointer
- Eine Priorität der Funktion
- Einen Pointer auf den nächsten Knoten

Wird eine der vier Beispielfunktionen zu einer Liste hinzugefügt, wird die Liste anschließend unverzüglich neu sortiert. Hier kommt ein Bubblesort Algorithmus zum Einsatz.

Funktionen können auf zwei unterschiedlichen Wegen zu einer Liste hinzugefügt werden. Es kann zum einen eine "reguläre"Registrierung der Funktion durchgeführt werden. Hierzu wird die Funktion `add_function(task[], function, priority);` . Die Priorität legt die Reihenfolge der Ausführung fest. Die höchste Zahl stellt hierbei auch die höchste Priorität um.

Außerdem kann eine Selbstregistrierung innerhalb der Funktion stattfinden. Hierfür wird die Funktion `selfRegister(function, priority, thread);`. Wichtig ist, dass der erste Thread der Null entspricht.

[2]https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html

3 Bedienung

3.1 Kompilierung

Zur Kompilierung des vorliegenden Programms muss beachtet werden, dass pthreads nur auf UNIX-basierten Betriebssystemen verfügbar ist. Getestet wurde die Kompilierung ausschließlich in folgendem Docker Container:
`leitnerfischerdhbw/es-ubuntu-x86`

Pthread muss manuell für das Kompilieren hinzugefügt werden. Der Befehl für die Kompilierung lautet als:
`gcc -pthread 'Dateiname'`

3.2 Änderung der Anzahl auszuführender Tasks

Um die Anzahl der Tasks zu ändern muss zuerst folgende Zeile geändert werden und auf die gewünschte Anzahl erhöht oder erniedrigt werden:
`define NUM_TASKS 4`
Danach muss die Zykluszeit festgelegt werden. Hierfür wird in diesem Array ein Element mit der gewünschten Zeit hinzugefügt werden (Zeit in ms):
`int cycleTimes[NUM_TASKS] = {1000, 5000, 10000, 100000};`
Zusätzlich sollte im Konfigurationsarray eine Spalte mit den gewünschten Funktionen und Priorität zur Ausführung in diesem Task hinzugefügt werden. Sollte das Array nicht

erweitert werden, kann es zu unschönen Ergebnissen führen, da die nicht initialisierten Elemente nicht automatisch '0' sind.

3.3 Änderung der Zykluszeiten der Tasks

Um die Die Zykluszeiten zu ändern muss lediglich in diesem Array die gewünschte Zeit geändert werden (Zeit in ms).
`int cycleTimes[NUM_TASKS] = {1000, 5000, 10000, 100000};`

3.4 Hinzufügen von Funktionen

Um eine Funktion hinzuzufügen muss im ersten Schritt folgender Wert erhöht bzw. erniedrigt werden:
`define NUM_FUNC 5`
Anschließend wird die neue Funktion analog zu den bestehenden Funktionen angelegt. Die Übergabewerte sind hierbei die double Werte a und b und eine int Variable um eine potentielle Selbstregistrierung zu ermöglichen. Im Array `double (*fktPointer[NUM_FUNC])(double, double, double)` muss außerdem ein neues Element angelegt werden, das gleich zu benennen ist, wie die erstellte Funktion Abschließend muss das Konifgurationsarray um eine (oder mehr) Zeilen erweitert und initialisiert werden.

4 Jitter

Obwohl Multithreading ein sehr Effizientes Konzept der Informatik ist, kann es sein, dass die Systemauslastung so hoch ist, dass die Threads nicht gleichzeitig ausgeführt werden können. Das kann verschiedene Gründe, wie zum Beispiel der Anzahl der verfügbaren Prozessorkerne, die Auslastung des Arbeitsspeichers etc., haben. Im schlimmsten Fall, sind so wenige Systemressourcen vorhanden, dass alle Threads nacheinander ausgeführt werden müssen. In diesem Fall würde sich der Jitter folgendermaßen berechnen:

$\sum Zykluszeiten$

In diesem Fall ist der Worst-Case-Jitter also 135 Sekunden. Um die Behauptung zu validieren, dass der Jitter von der Verfügbarkeit von Systemressourcen abhängt, wurde während der Ausführung des Programms ein CPU-Benchmark gestartet. Tatsächlich kannte beobachtet werden, dass die Ausführungszeit der einzelnen Threads teilweise deutlich erhöht wurde. Getestet wurde auf einem MacOS System mit Apple M1 und 16gb RAM. Die maximal zu beobachtende Abweichung ist 99ms.

5 Tests

Um das vorliegende Programm zu testen wird dieses als erstes Kompiliert, um sicherzugehen, dass keine Kompilerwahrnungen oder Errors auftreten. Ein weiterer Testfall ist das Einfügen eines `sleep` in eine Funktion, das eine aufgerufene Funktion so lange blockiert, dass die eigentliche Zykluszeit abgelaufen ist. Das Programm reagiert wie erwartet und gibt ein Fehler auf der Konsole aus, dass die Funktion zu lange für die Ausführung benötigt hat und wird erst wieder neu gestartet, wenn der nächste reguläre Zyklus ansteht. Außerdem wird die Erweiterbarkeit der Tasks und Funktionen überprüft. Die Selbstregistrierung von Funktionen wird getestet und funktioniert ordnungsgemäß. Beim Starten des Programms wird auf der Konsole eine Tabelle ausgegeben, welche genau zeigt, welche Funktionen in welchem Task konfiguriert sind.

6 Quellen und Hilfsmittel

Für die Umsetzung der Anforderungen in den Quellcode wurden unter anderem folgende Hilfsmittel verwendet: "Fit fürs Studium Informatik Rheinwek Computing - Bockmeyer|Fischbeck|Neubert

Vorlesungsskripte aus der Vorlesung 'Programmieren' von Herr Prof. Dr. Ing. Florian Leitner-Fischer

`https://hpc-tutorials.llnl.gov/posix/`

`chat.openai.com`