Automatisches Entwurfs- und Entwicklungsframework für harte Echtzeitsysteme

Jan Blumenthal, Frank Golatowski, Jens Hildebrandt
{jan.blumenthal,frank.golatowski,jens.hildebrandt}@etechnik.uni-rostock.de
Universität Rostock, Fachbereich Elektrotechnik und Informationstechnik

Abstract

Der Artikel beschreibt das YASA 2 - Framework für den Entwurf und die Entwicklung harter Echtzeitsysteme. Er beinhaltet ganzheitliche Lösungsvorschläge für Probleme, die insbesondere aus dem Verhalten zugrunde liegender Betriebssystemscheduler resultieren. Das Framework berücksichtigt die theoretischen und praktischen Aspekte gleichermaßen und stellt eine Symbiose aus einem Schedulinganalyser und der Schedulingumgebung eines Echtzeitbetriebssystems dar.
Durch die hier vorgestellte Softwarelösung wird der Entwicklungszyklus für Echtzeitsysteme grundlegend verbessert. Mit dem Framework werden auf ihr Zeitverhalten hin zu testende Zielsysteme generiert, die RT-Linux als Betriebssystem verwenden. Der Cross-Entwicklungsprozess ist sowohl unter Windows als auch unter Linux durchführbar.
Neu an diesem Framework ist die Möglichkeit, verschiedene neuartige Schedulingalgorithmen und Synchronisationsverfahren auf ihre Eignung hin zu evaluieren. Nach erfolgter Evaluierung lassen sich die Algorithmen in das Zielsystem integrieren. Dabei können sowohl durch Hardware unterstützte (Scheduling-Coprozessoren) als auch Software-Scheduler in das Zielsystem eingebettet werden.

1. Einleitung

In modernen Betriebssystemen werden Programme aus Sicht des Anwenders quasi-parallel abgearbeitet. In der Realität erfolgt die Ausführung jeweils in kurzen Abständen hintereinander. In Echtzeitsystemen besitzen Prozesse Zeitschranken, die einzuhalten sind, um die Funktionalität der gesamten Anwendung nicht zu gefährden. Das Verpassen einer Deadline könnte fatale Auswirkungen bis hin zur Gefährdung von Menschen und Maschinen zur Folge haben. Für die Entwicklung harter Echtzeitsysteme ist daher der Einsatz von Schedulern erforderlich, die das Einhalten von Zeitschranken der einzelnen Prozesse garantieren.
In der Praxis werden vielfach statische Scheduler und Synchronisationsprotokolle eingesetzt, die mit dem Betriebssystem fest verbunden sind. Der Standardscheduler ist der prioritätsbasierte Scheduler, der durch seine einfache Realisierung, aber schlechtes und nicht dynamisches Schedulingverhalten gekennzeichnet ist. Der Einsatz von dynamischen Schedulern und Synchronisationsprotokollen erfordert einen umfangreichen und fehleranfälligen Eingriff in die Betriebssysteme, da keine einheitliche Programmierschnittstelle existiert.
Die Optimierung des Schedulingverhaltens erweist sich als schwierig, da die aktuellen Betriebssysteme keine Möglichkeit der Protokollierung und späteren Auswertung zur Verfügung stellen. Ein Vergleich von verschiedenen Schedulern erweist sich häufig als schwierig.
Ebenso problematisch ist der Einsatz von Hardwareschedulern. Sie bieten aufgrund der Parallelität der Verarbeitung einen massiven Geschwindigkeitsvorteil gegenüber Softwareschedulern. Ihr Einsatz ist aber begrenzt auf Systeme mit geeigneter Hardware und Betriebssystemen mit entsprechenden Schnittstellen zur Programmierung der Coprozessoren.

2. Stand der Technik

In den zurückliegenden Jahren wurden verschiedene Frameworks zur Entwurfsunterstützung harter Echtzeitsysteme entwickelt. Dazu gehören Perts [5], RTGA (Real-time Graphic Analyzer) [7], Stress [8], Timewiz [9] und Cheddar [12]. In [6] wird ein rekonfigurierbares Schedulinganalyse-Framework für den RED-Linux Real-Time Kernel beschrieben.
EVASCAN [1][2] ist ein allgemeines Framework, das eine Schedulinganalyse von Echtzeitsystemen mittels YASA (Yet Another Scheduling Analyzer) [3] vornimmt. Zur Evaluierung realer Systeme werden dort synthetische Lasten herangezogen.
Das weiterentwickelte und komplett überarbeitete neue Framework YASA 2 (im Weiteren als YASA bezeichnet), zielt auf die Entwicklung von echten Targetsystemen, die einer Schedulinganalyse unterworfen werden. Durch die Anwendung eines solchen Frameworks erhält der Echtzeitsystementwickler ein sehr leistungsfähiges Werkzeug zur Unterstützung bei Entwurf und Implementierung harter Echtzeitsysteme.

3. Ziele, Lösungsansatz und Architektur des Frameworks

Durch die vorgestellte Softwarelösung wird der Entwicklungszyklus für Echtzeitsysteme deutlich verbessert und auf eine deterministische Grundlage gestellt.
In bisherigen Echtzeitentwicklungen konnte lediglich der Funktionsnachweis des Systems durch Testläufe erbracht werden. Die Einhaltung der Zeitschranken konnte bisher nicht garantiert werden, da vom Betriebssystem keine Informationen über den tatsächlichen Schedulingverlauf zur Verfügung gestellt wurden. Ein durch ungünstiges Scheduling ausgelöstes Fehlverhalten des Systems konnte nur durch Trial & Error Methoden erkannt und verbessert werden (Abbildung 1: Vergleich der Entwicklungszyklen).
In YASA wird ein modifizierter Ansatz angewendet. Das eingesetzte Betriebssystem wird durch *Executives* erweitert. Die Executives ermöglichen eine Protokollierung interner Ereignisse für die spätere Analyse des Schedulingverhaltens.

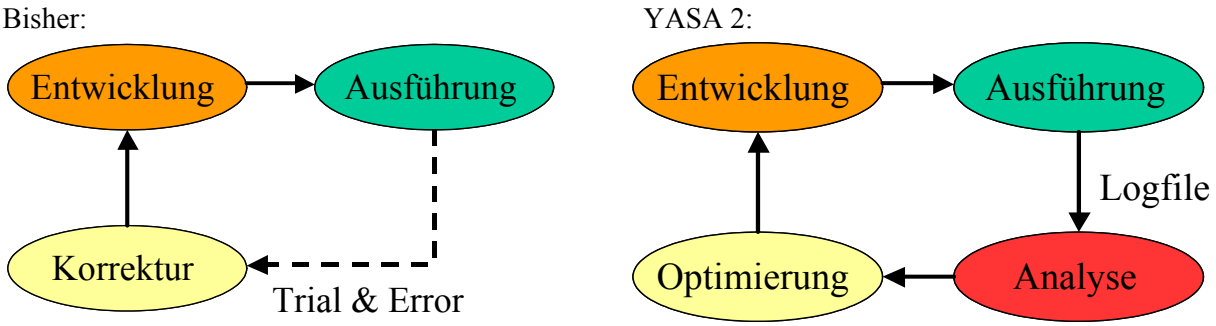

Abbildung 1: Vergleich der Entwicklungszyklen

Neben der Optimierung des zeitlichen Ablaufs wurden weitere Teilziele für das Framework definiert:

- Ausführung der Applikationen auf Echtzeitsystemen unter realen Bedingungen
- Unterstützung verschiedener Echtzeitbetriebssysteme bzw. Hardware-Coprozessoren
- Einheitliche Entwicklungsumgebung für Simulationen und Echtzeitanwendungen
- Erweiterbarkeit durch Entwicklung neuer Scheduler
- Einfacher Wechsel des Schedulers in der Zielumgebung
- Unterstützung verschiedener Synchronisationsprotokolle unabhängig vom Scheduler
- Wahl der Synchronisationsprotokolle pro Semaphore

- Unterstützung multiprozessorfähiger Anwendungen
- Protokollierung von Ereignissen
- Ausführliche Auswertung und umfangreiche grafische Darstellung der Ergebnisse
- Konsequente Trennung von Entwicklungsframework und Front-End
- Ansprechende und zeitgemäße Oberfläche

Lösungsansatz und Architektur

Das Entwicklungsframework YASA kapselt durch den Einsatz von Executives die Schedulingumgebung der Betriebssysteme und erweitert sie um zusätzliche Eigenschaften, wodurch sich folgende Vorteile ergeben:

- Definition von einheitlichen Programmierschnittstellen (API)
- Austauschbarkeit einzelner Komponenten zur Laufzeit
- Protokollierung aller Systemereignisse
- Auswertung und Optimierung durch externe Programme
- Vergleichbarkeit verschiedener Konfigurationen

Die Erweiterung der originalen Betriebssysteme beschränkt sich lediglich auf die Schedulingumgebung, so dass die Beziehungen zwischen den Prozessen, dem Betriebssystem und der Systemumgebung nicht beeinflusst werden.

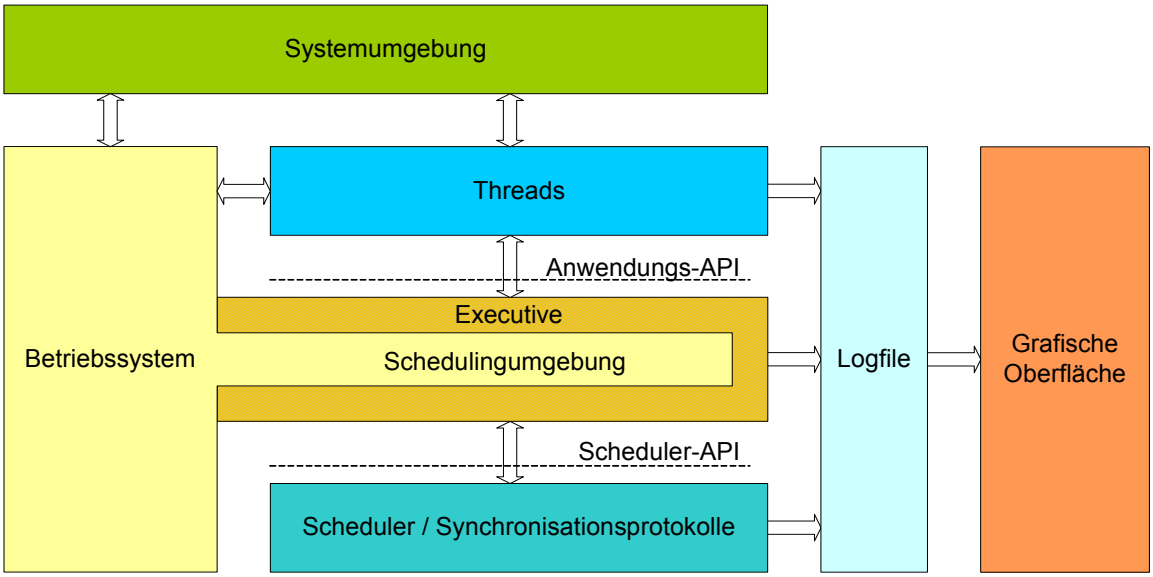

Abbildung 2: Architektur des YASA-Frameworks

Das Executive besteht im Wesentlichen aus Präprozessoranweisungen, die die internen Systemstrukturen, z.B. des Prozesskontrollblocks PCB bzw. der Schedulingstrukturen der Prozessoren, mit denen der YASA-Komponenten verbinden. Der Einsatz von Makros erweist sich als vorteilhaft, da der Programmumfang bereits auf Kompilierungsebene dem aktuellen System angepasst wird und überdimensionierte Programmteile sowie unnötige Verzweigungen verhindert werden.

Scheduler

Die herkömmlichen Implementierungen von Schedulern in Echtzeitbetriebssystemen arbeiten zumeist prioritätsbasiert, in seltenen Fällen deadlinegesteuert. In YASA sind zurzeit 12 Scheduler umgesetzt,

die in Tabelle 1 aufgeführt sind. Die Kenntnis des Prioritätstyps des verwendeten Schedulers ist wichtig für die sinnvolle Konfiguration der Tasks.

Tabelle 1: Verfügbare Schedulingverfahren in YASA

Abkürzung	Protokoll	Prioritätstyp	Prioritätsvergabe
DMS	Deadline Monotonic	Deadline	Statisch
EDF	Earliest Deadline First	Deadline	Dynamisch
ELLF	Enhanced Least Laxity First	Schlupf	Dynamisch
FCFS	First Come First Serve	FIFO	Zyklisch
LEF	Least Execution Time First	Rechenzeit	Statisch
Linux	Linux Kernel Scheduler	Integer	Dynamisch
LLF	Least Laxity First	Schlupf	Dynamisch
Prio	Prioritätsbasiert	Integer	Statisch
Red	Robust Earliest Deadline First	Deadline	Dynamisch
RMS	Rate Monotonic	Deadline	Statisch
RR	Round Robin	FIFO	Zyklisch
SJF	Shortest Job First	Rechenzeit	Dynamisch

Synchronisationsprotokolle

Neben den Schedulern werden verschiedene Synchronisationsprotokolle unterstützt. Ein Synchronisationsprotokoll hat entscheidenden Einfluss auf die Deterministik der Gesamtapplikation, da durch sie das Schedulingverhalten beim Setzen und Freigeben von Mutexen (Semaphoren) beeinflusst wird.
In YASA sind die Protokolle implementiert, die in heutigen Echtzeitbetriebssystemen umgesetzt sind bzw. im POSIX-Standard (1003.1c) und den zugehörigen POSIX-Profilen (PSE51) definiert sind. Die aktuell verwendeten Protokolle in POSIX beschränken sich allerdings auf Priority Inheritance (PTHREAD_PRIO_INHERIT) und Priority Ceiling (PTHREAD_PRIO_PROTECT).

Tabelle 2: Synchronsiationsprotokolle in YASA

Abkürzung	Protokoll	Bisherige Schedulingklasse	Schedulingklasse in YASA
FIFO	First In First Out	PRIO	Statisch
PRIO	Priority Inheritance Protocol	PRIO	Statisch
PIP	Priority Inheritance Protocol	PRIO	Statisch
PCP	Priority Ceiling Protocol	PRIO	Statisch
CSP	Ceiling Semaphore Protocol	PRIO	Statisch
DPCP	Dynamic Priority Ceiling Protocol	EDF	Alle
SRP	Stack Resource Policy	EDF	Alle
KPP	Kernel Priority Protocol	Alle	Alle

Die Zuordnung der Synchronisationsprotokolle ist bisher an den jeweiligen Scheduler gebunden (Tabelle 2). Durch die Verwendung von YASA ist es möglich, dynamische Synchronisationsverfahren [10] in jeder Zielumgebung zu verwenden. Um eine höchstmögliche Flexibilität in der Konfiguration von unterschiedlichen Schedulern und Synchronisationsprotokollen zu erreichen, wurde der Datentyp

dynamischer Prioritätstyp eingeführt. In Abhängigkeit dieses Datentyps ändern sich bei Bedarf die Vergleichsoperationen innerhalb der Scheduler. Dadurch ist es möglich, alle statischen Scheduler mit statischen Synchronisationsprotokollen zu kombinieren. Die dynamischen Synchronisationsprotokolle können durch diese Neuerung mit jeder Schedulerklasse zusammenarbeiten. So kann im Gegensatz zu der Stack Resource Policy (SRP) Implementierung von P. Balbastre in RT-Linux [13] SRP statt nur mit EDF auch mit anderen Schedulern wie LLF oder SJF benutzt werden.
Die bei der Programmierung der Scheduler und Synchronisationsprotokolle verwendeten Präprozessor-Makros bieten eine einheitliche Schnittstelle zu der gekapselten Schedulingumgebung, wodurch sie vollständig plattformunabhängig sind und auf jeder Zielplattform eingesetzt werden können.

Entwicklungsprozess

Die Entwicklung einer Echtzeitanwendung in YASA erfolgt in einem dreiteiligen Zyklus (Abbildung 3). Die erste Phase dient der Konfiguration des Projekts, der Deklaration von Prozessen und Mutexen (Semaphoren) sowie der Beschreibung der Schedulingumgebung des Systems. Das Zielsystem der Anwendung wird durch das Executive bestimmt.
In der Erzeugungsphase wird das Projekt unter Beachtung der Prozess-Parameter und der Prozess-Quelltexte kompiliert. Die Einstellungen der Schedulingumgebung bewirken eine Veränderung der Kernelquelltexte und erfordern eine Neuübersetzung des Betriebssystemkernels.
Die dritte Phase beinhaltet die autonome Ausführung des Projekts auf dem angepassten Echtzeitkernel mit anschließender Erzeugung der Systemlogfiles. Die folgende Auswertung gestattet eine Korrektur der Prozess-Parameter sowie den Start eines weiteren Zyklusdurchlaufs.

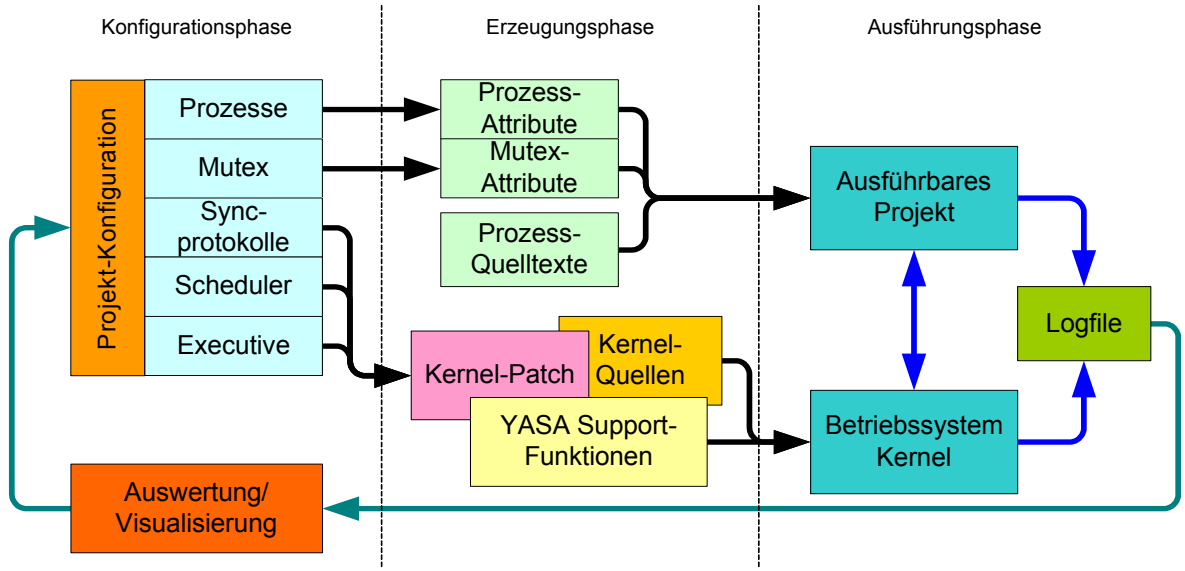

Abbildung 3: Entwicklungsphasen eines YASA-Projekts

Die Executives sind bereits für RT-Linux vorhanden und einsatzbereit, wodurch sich vielfältige Einsatzgebiete ergeben:

- Einfache Erstellung von Echtzeitapplikationen
- Analyse und Optimierung des Schedulingverhaltens
- Fehlersuche im Kernelmodus
- Lehr- und Forschungszwecke
- Entwicklung neuer Scheduler

In YASA sind alle Komponenten vollständig modularisiert. Wird ein neues Zielsystem hinzugefügt, hat dieses Zugriff auf alle verfügbaren Scheduler bzw. Synchronisationsprotokolle. Andererseits sind neuentwickelte Scheduler und Synchronisationsprotokolle für alle Zielsysteme nutzbar. Ausnahmen bilden Systeme, deren Scheduling durch spezielle Hardware-Coprozessoren unterstützt wird. Sie sind Gegenstand aktueller Forschungsarbeiten.

4. Benutzeroberfläche

Zur Konfiguration und Auswertung der Echtzeitprojekte wurde ein eigenständiges und leistungsfähiges grafisches Front-End entwickelt (Abbildung 4). Die Programmierung der grafischen Bedieneroberfläche unter Verwendung der Klassenbibliothek Qt von Trolltech ermöglicht den Einsatz des Entwicklungssystems unter allen infrage kommenden Betriebssystemen.

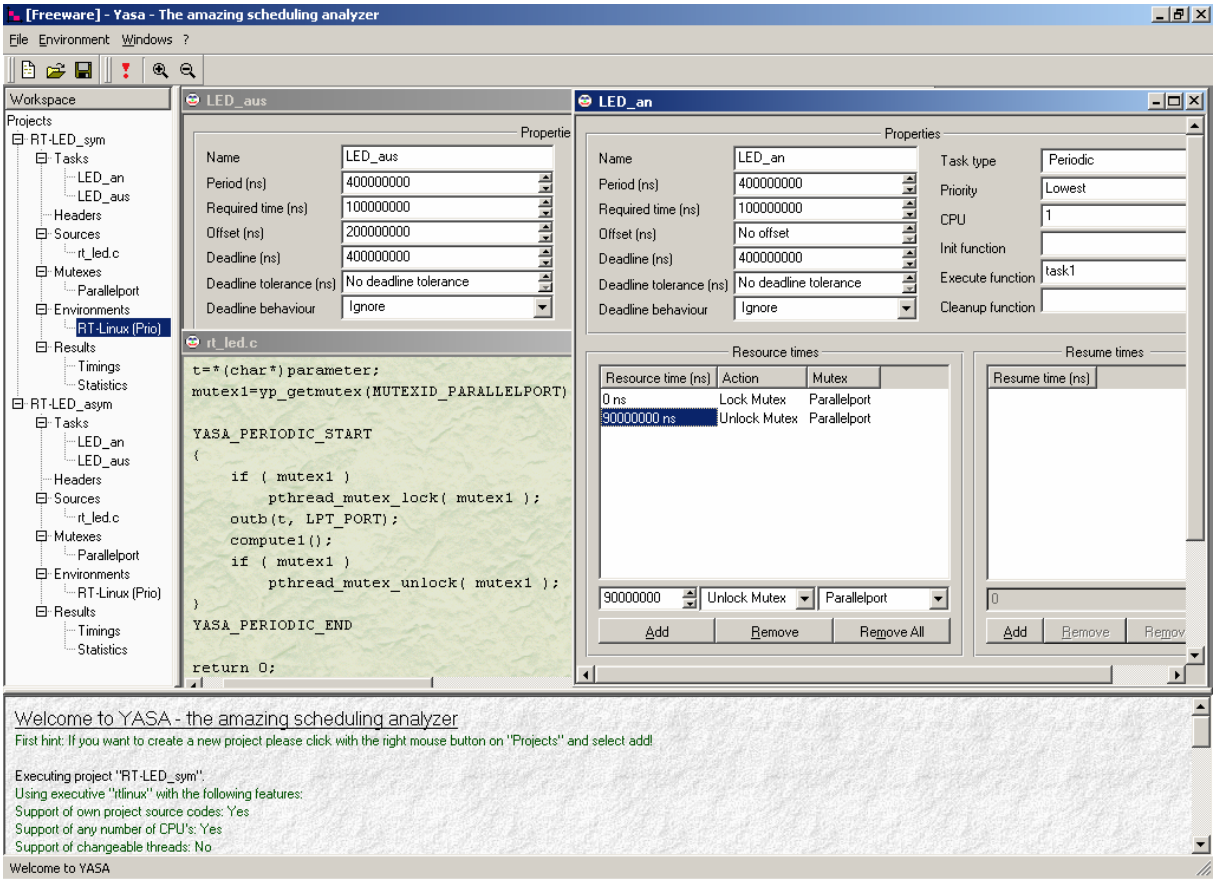

Abbildung 4: Bedienoberfläche des Frameworks

5. Anwendungsentwicklung

In diesem Abschnitt wird der prinzipielle Ablauf der Anwendungsentwicklung in YASA beschrieben. Nach dem Start wird ein Arbeitsbereich mit einem Projekt angelegt. Prinzipiell ist es möglich, mehrere Projekte gleichzeitig zu verarbeiten. Ein Projekt wird als Baumstruktur angezeigt und enthält die zugehörigen Objekte bzw. Eigenschaften wie z.B. Tasks, Headers, Sources, Mutexe und Environments.
Eine Task ist charakterisiert durch folgende Eigenschaften: Periode, Rechenzeit, Offset, Deadline, Priorität. Die spätere Ausführung der Tasks auf realen Echtzeitsystemen erfordert die Deklaration von drei Funktionen: *Init()*, *Execute()* und *Cleanup()*. Die optionale Init()-Funktion wird bei Start der Applikation zur Initialisierung für jede Task separat gestartet. Die Execute()-Funktion enthält den Pro-

grammcode der Task. Die Cleanup()-Funktion dient dem Freigeben von Ressourcen und dem Beenden einer Task.
Das Schützen von kritischen Abschnitten wird durch die Konfiguration von Mutexen erreicht. Jedem Mutex kann ein Synchronisationsprotokoll zugewiesen werden. YASA berechnet für jedes Mutex die Ceiling-Prioritäten der entsprechenden Prioritätstypen (s. Tabelle 1) und fügt die erforderlichen Daten automatisch in die Mutex-Attribute der erzeugten Quelltexte ein.
Durch die Eigenschaft *Environment* wird das Zielsystem charakterisiert. Hier werden den einzelnen Prozessoren die gewünschten Scheduler zugeordnet. Im weiteren Entwurfsverlauf müssen die notwendigen Thread-Quelltexte über die Eigenschaft *Sources* hinzugefügt werden.
Die konfigurierten Task-, Mutex- und Environmentparameter müssen an das Zielsystem übergeben werden. Diese Parameter werden in Attribut-Strukturen automatisch erzeugter C-Quelltexte abgelegt, wodurch eine plattformunabhängige Darstellung erreicht wird. Im Anschluss kann die Applikation für das Zielsystem unter Einbeziehung von Startupcodes übersetzt und ausgeführt werden. Gegebenenfalls ist eine Übersetzung des kompletten Betriebssystem-Kernels, wie unter RT-Linux, notwendig (siehe Abbildung 3).
Während der Ausführung einer Applikation werden die internen Ereignisse mitprotokolliert und später in einem Logfile gespeichert. Diese Datei wird von YASA eingelesen und interpretiert. Es besteht die Möglichkeit, die Auswertung in grafischer (Abbildung 5) oder in tabellarischer Form vorzunehmen. Im Gegensatz zu herkömmlichen Schedulinganalysern kann YASA nicht nur die Schedulingverläufe der Projekte, sondern auch die der einzelnen Tasks in grafischer oder tabellarischer Form vergleichen.

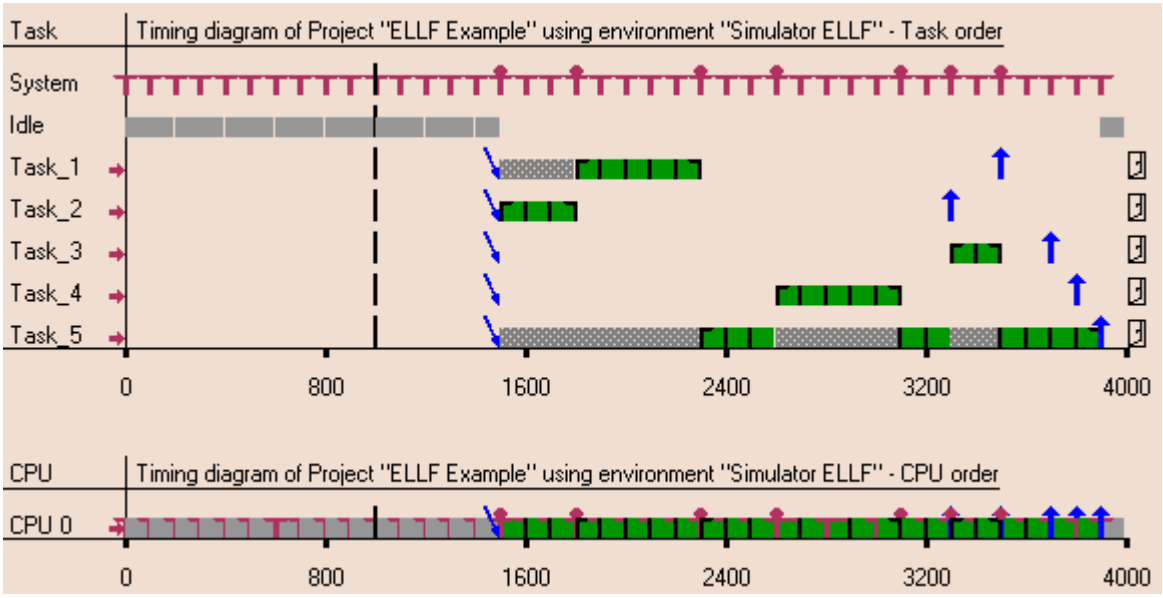

Abbildung 5: Darstellung des Zeitverhaltens (ELLF)

Dem Entwickler bieten sich dadurch vielfältige Möglichkeiten des Vergleichs und der Analyse von unterschiedlichen Schedulingverläufen. Abbildung 5 zeigt die Ausführungsreihenfolge wie sie sich bei der Simulation unter Verwendung des ELLF-Schedulers ergibt. Die zusätzliche Darstellung von Mutex- und Prozessordiagrammen verbessert die Vergleichbarkeit gegenüber herkömmlichen Verfahren deutlich. Eine weitere Darstellungsform ermöglicht den direkten visuellen Vergleich verschiedener Schedulingalgorithmen (Abbildung 6).

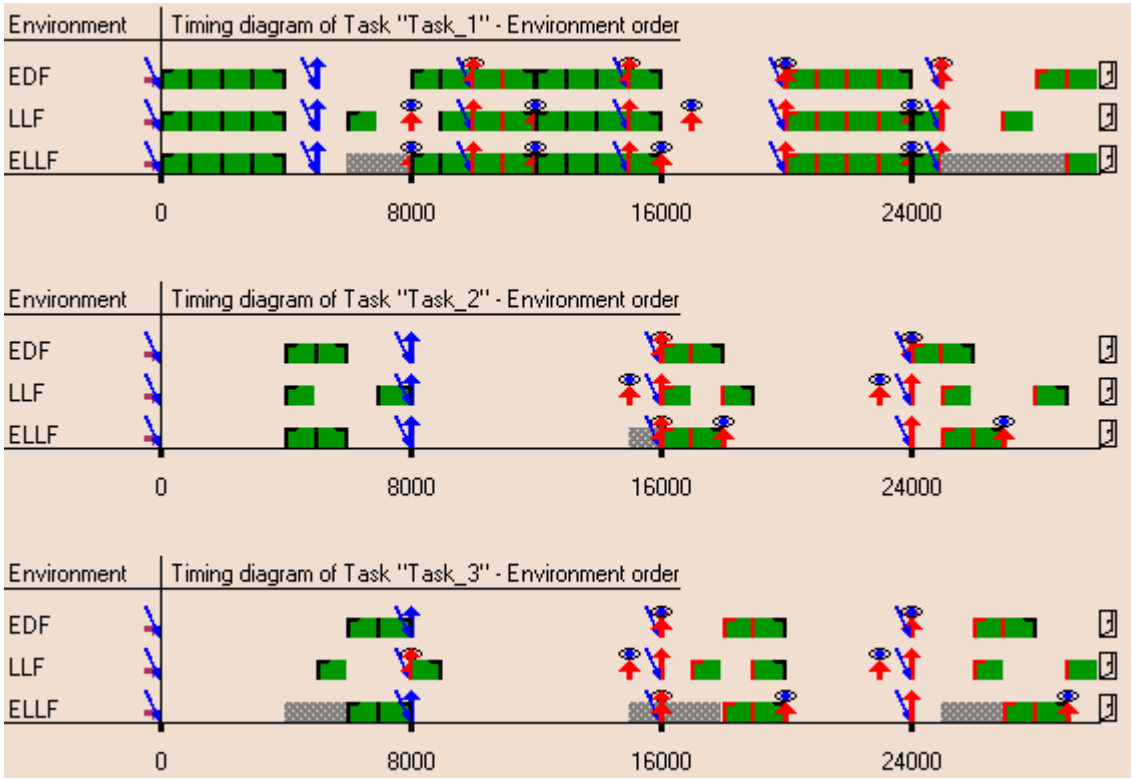

Abbildung 6: Vergleich verschiedener Schedulingalgorithmen

Weiterhin gibt es sehr umfangreiche Möglichkeiten der statistischen Auswertung, die dem Echtzeitsystementwickler wichtige Informationen über sein System liefern. YASA berechnet aus den aufgezeichneten Daten über 100 unterschiedliche Einzelinformationen, wie z.B. durchschnittliche Berechungszeit, Unterbrechungen, prozentuale und absolute Auslastung, Reaktionszeiten, Jitter u.v.m. Die ermittelten Informationen können in verschiedenen Darstellungsformen (Tabellen in Task- bzw. Environmentanordnung) miteinander verglichen werden.

6. Zusammenfassung

Das hier vorgestellte Framework unterstützt den Entwickler beim Entwurf von harten Echtzeitsystemen. Es stützt sich dabei wesentlich auf die Entwicklung und Auswahl neuer Schedulingalgorithmen und Synchronisationsprotokolle, die auch in realen Systemen einsetzbar sind. Es ist in der Lage, das Schedulingverhalten von Programmen frühzeitig mit echter Last auszuführen und zu analysieren. In der derzeitigen Implementierung stehen RT- Linux und eine Simulationsumgebung als Zielplattform zur Verfügung. Eine Umsetzung von RT-Linux, die durch Scheduling-Coprozessoren unterstützt wird, befindet sich in der Entwicklung [2].
Das Entwicklungsframework ist vollständig modular aufgebaut. Es sind Erweiterungen bezüglich Profiling-Analysen, Worst-case execution time Analyse und das Hinzufügen neuer Executives zur Unterstützung weiterer Echtzeitbetriebssysteme möglich.

7. Literaturverzeichnis

[1] Golatowski, F.; Hildebrandt, J.; Timmermann, D.:
Rapid Prototyping with Reconfigurable Hardware for Embedded Hard Real-Time Systems. 19th IEEE Real-Time Systems Symposium 98, WIP -Session, Madrid, Spanien, 1998

[2] Golatowski, F.; Hildebrandt, J.; Timmermann, D.:
Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems, 11th Euromicro Conference on Real-Time Systems, York, Großbritannien, 1999

[3] *Yasa- Yet Another Schedulability Anayzer*, http://www-md.e-technik.uni-rostock.de/ma/gol/yasa/, Universität Rostock, 1998

[4] Blumenthal, J.; Golatowski, F.; Hildebrandt, J.; Timmermann, D.:
"Framework for validation, test and analysis of real-time scheduling algorithms and scheduler implementations", 13th IEEE International Workshop on Rapid System Prototyping, July 1-3, 2002, Darmstadt

[5] Liu, J. W. S.; Liu, C. L.; Deng, Z.; Tia, T. S.; Sun, J.; Storch, M.; Hull, D.; Redondo, J. L.; Bettati, R.; Silberman, A.: *PERTS: A prototyping environment for real-time systems.* International Journal of Software Engineering and Knowledge Engineering, 6(2):161-177, 1996.

[6] Wang, Y-C.; Lin, K-J.: *Implementing a General Real-Time Scheduling Framework in the RED-Linux Real-Time Kernel.* 20th IEEE Real-Time Systems Symposium, 1999, Phoenix, Arizona

[7] Ripoll, I.: *RTGA Real-time Graphic Analyzer*, 2002,
http://bernia.disca.upv.es/rtportal/apps/rtga/index.html

[8] Audsley, N.C.; Burns, A.; Richardson, M.F.; Wellings, A.J.: *Stress: A Simulator for Hard Real-Time Systems*. Software-Practice and Experience, Vol. 24(6), p. 543, 564 (June 1994).

[9] *Timewiz- An Architectural modelling, analysis, and simulation environment for real-time systems*. White paper, http://www.timesys.com/pdf/timewiz_ds.pdf

[10] Chen, M.-I.; Lin, K.-J.: *Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems*, Journal of Real-Time Systems, 1990

[11] Blumenthal, J: *Automatisches Entwurfs- und Entwicklungssystem für harte Echtzeitsysteme*, Universität Rostock, 2002

[12] Chauvin, N.; Demurget, S.: *„The Cheddar project"*, http://beru.univ-brest.fr/~singhoff/chedda, Universität Brest, 2002

[13] Balbastre, P.; Ripoll, I.: *„Integrated Dynamic Priority Scheduler for RTLinux"*, Universität Valencia, 2000

[14] Baker , T.P.: *„A Stack-Based Resource Allocation for Realtime Processes"*, Proceedings 11th IEEE Real Time Systems Symposium, IEEE Computer Society Press (December 1990)