1 Einleitung

Systeme, die neben den ohnehin vorhandenen funktionalen Anforderungen zusätzlich noch zeitlichen Anforderungen – typischerweise im Sekundenbereich und darunter – genügen müssen, werden als Realzeitsysteme bezeichnet.

Als wesentliche Komponente eines solchen Realzeitsystems gilt die Steuerung, die über Ein- und Ausgabeinterfaces, über Sensoren und Aktoren mit dem Benutzer und der Umwelt kommuniziert. Klassische Realzeitsteuerungen bestehen aus einer Singlecore-CPU, die häufig am leistungstechnischen Limit betrieben wird. Als Systemsoftware wird – wenn überhaupt – ein schlankes, vielfach selbst entwickeltes Realzeitbetriebssystem eingesetzt.

Dabei wird der Entwickler oft mit dem Problem konfrontiert, viele unterschiedliche Aufgaben in einer einzelnen Task zu implementieren. Der Zugriff auf Hardware erfolgt direkt, ohne Interaktion mit der Systemsoftware. Höhere Funktionalitäten wie Internetdienste sind in diesem Umfeld nur mit erheblichem Aufwand zu realisieren.

Neue Entwicklungen auf dem Gebiet der Prozessoren hin zu mehr Leistungsstärke, zu Energieeffizienz und zu Multicore eröffnen neue Möglichkeiten, moderne Realzeitsysteme zu verwirklichen. In modernen Realzeitsteuerungen löst der sogenannte Tickless-Betrieb die periodisch ablaufende Systemsoftware ab. Intuitive Mensch-Maschine-Schnittstellen und Vernetzung, insbesondere die Internetanbindung, stehen dem Entwickler über standardisierte Interfaces zur Verfügung. Anstelle proprietärer Systemsoftware bietet sich ein echtzeitfähiges Standardbetriebssystem an, bei dem beispielsweise ein einzelner Prozessorkern für Realzeitaufgaben reserviert werden kann.

Die Applikation passt sich an. Wo klassisch ein einzelner Thread ereignisgesteuert viele unterschiedliche Aufgaben abarbeitet, werden bei einem modernen Systementwurf unabhängige Aufgaben identifiziert und in separate Tasks implementiert. Die dabei notwendige Inter-Prozess-Kommunikation bringt allerdings kritische Abschnitte mit sich, die mit geeigneten Methoden zu schützen sind.

Für die Kodierung greift der Entwickler auf die vielfältigen, vorgefertigten und ebenfalls standardisierten Funktionen des Betriebssystems

zurück, um beispielsweise Tasks zu erzeugen, zu priorisieren oder auf einen Rechnerkern zu fixieren; auf Funktionen, um den Scheduler zu konfigurieren, zwischen seinen Tasks Daten auszutauschen oder um kritische Abschnitte zu schützen, und auf Funktionen, die für das Zeitmanagement benötigt werden.

Die Erstellung eines modernen Echtzeitsystems erfordert über die Kenntnis der technischen Möglichkeiten hinaus ein theoretisches Basiswissen. Neben der formalen Beschreibung seines Entwurfs muss der Entwickler aus der Aufgabenstellung die Echtzeitparameter für den Realzeitnachweis extrahieren und die zugehörigen Realzeitbedingungen aufstellen und anhand des von ihm erstellten Entwurfs überprüfen können. Für diesen notwendigen, in der Praxis häufig ausgelassenen Schritt benötigt der Entwickler Kenntnisse über das eingesetzte Scheduling-Verfahren und über die detallierte Verarbeitung kritischer Abschnitte.

Ziel dieses Buches ist es damit,

- Grundkenntnisse moderner Realzeitsysteme kompakt und praxisorientiert zu vermitteln,
- die Parameter vorzustellen, die die zeitlichen Anforderungen auf der einen Seite und das Zeitverhalten der zur Lösung eingesetzten Rechenprozesse auf der anderen Seite beschreiben,
- die Möglichkeiten zum Entwurf und zur Realisierung moderner Realzeitsysteme auf Basis von Single- oder Multicore-Hardware aufzuzeigen,
- die vielfältigen Einsatzmöglichkeiten von (Embedded) Linux im Realzeitumfeld zu verdeutlichen,
- die praxisrelevanten Funktionen zur Realisierung der Applikation anhand von Codebeispielen vorzustellen,
- in die Grundlagen der Betriebssicherheit (Safety) und der Angriffssicherheit (Security) einzuführen,
- einfache und weitestgehend intuitiv einzusetzende formale Beschreibungsmethoden vorzustellen und
- das Rüstzeug für einen Realzeitnachweis zu vermitteln, der auch Blockierzeiten berücksichtigt.

Scope

Das Buch gibt einen kompakten Überblick über die wesentlichen Aspekte von Realzeitsystemen. Es ist damit weder ein themenumfassendes Handbuch, noch ein auf Einzelaspekte fokussiertes Werk.

Das Buch richtet sich sowohl an Neulinge auf dem Gebiet der Realzeitsysteme, wie beispielsweise Studenten der Informatik, Elektrotechnik oder Mechatronik, als auch an den erfahrenen Entwickler. Es soll dabei den Systemarchitekten ebenso ansprechen wie den Programmierer. Da

softwarespezifische Aspekte im Vordergrund stehen, richtet es sich nicht an Hardwareentwickler.

Grundkenntnisse auf dem Gebiet der Betriebssysteme sind zum Verständnis von Vorteil, die Codebeispiele im Praxisteil sind in der Programmiersprache C geschrieben. Sie sind grundsätzlich unabhängig von einer spezifischen Systemversion, wir haben sie auf einem Ubuntu 12.04 LTS mit Kernel 3.2 getestet.

Das Buch ist aufgrund der Verbreitung und der herausragenden Eigenschaften Linux-zentriert, allerdings dabei unabhängig von einer spezifischen Plattform oder einer bestimmten CPU. In Beispielen referenzieren wir meist die verbreiteten x86- oder ARM-Architekturen.

Realzeitsysteme werden im amerikanischen Sprachraum sehr häufig mit eingebetteten Systemen (Embedded Systems) gleichgesetzt. Streng genommen ist das nicht korrekt, denn die integrierte, mikroelektronische Steuerung – so die Definition eines eingebetteten Systems – muss nicht zwangsweise zeitliche Anforderungen erfüllen. Dennoch besteht zwischen Realzeitsystemen und eingebetteten Systemen eine enge Verwandtschaft und ist daher auch Thema dieses Buches. Sogenannte Deeply Embedded Systems – Systeme, die auf einfachen Prozessoren und oft selbst geschriebener Systemsoftware beruhen – sind allerdings außerhalb des Scopes.

Aufbau des Buches

In Kapitel 2 werden zunächst die Beschreibungsgrößen eines Realzeitsystems definiert und die Bedingungen abgeleitet, die zur Erfüllung der zeitlichen Anforderungen einzuhalten sind.

Das dritte Kapitel stellt die für das Realzeitverhalten wesentlichen Komponenten der Systemsoftware vor. Dazu gehören beispielsweise die innerhalb der Realzeitsysteme auswählbaren und parametrierbaren Scheduling-Strategien und die Zeitverwaltung.

Aspekte der nebenläufigen Realzeitprogrammierung behandelt das vierte Kapitel anhand von vielen Codebeispielen. Hier werden die wichtigsten Systemcalls und Funktionen vorgestellt, mit denen Tasks erzeugt, Zeiten gelesen oder kritische Abschnitte geschützt werden.

Verschiedene, mit Linux gut realisierbare Realzeitarchitekturen, basierend auf Single- und Multicore-Hardware, werden im fünften Kapitel vorgestellt.

Realzeitsysteme werden häufig im sicherheitskritischen Umfeld eingesetzt. Daraus resultieren erweiterte Anforderungen an die Korrektheit, an die Ausfallsicherheit, aber auch an den Schutz vor unbefugtem Zugriff, lokal, aber auch über das Internet. Kapitel 6 behandelt Aspekte der Betriebs- und der Angriffssicherheit.

Kapitel 7 gibt dem Entwickler ausgewählte Methoden an die Hand, mit denen die Taskgebilde eines Realzeitentwurfs dargestellt und die Abläufe innerhalb der Tasks modelliert werden können. Vielen Lesern dürften die vorgestellten Verfahren bekannt sein; das hat uns motiviert, den Abschnitt weiter hinten im Buch zu platzieren.

Das letzte Kapitel widmet sich schließlich dem Realzeitnachweis, mit dem formal das Einhalten der zeitlichen Anforderungen unter Berücksichtigung verschiedener Systemparameter (zum Beispiel Scheduling) nachgewiesen werden kann.

Embedded Linux

Im Rahmen des Buches referenzieren wir immer wieder auf Linux. Tatsächlich ist Linux in den letzten Jahren zu einem Standardbetriebssystem mit Realzeiteigenschaften umgebaut worden. Dabei ist Linux – natürlich abhängig von der eingesetzten Hardware – in der Lage, harte Zeitgrenzen im Mikrosekundenbereich einzuhalten. Allerdings benötigt der Kernel hierzu zurzeit noch den sogenannten PREEMPT-RT-Patch ([http://www.kernel.org/pub/linux/kernel/projects/rt/]). Da Linus Torvalds mit jeder neuen Kernel-Version Teile des Patches übernimmt, wird das Patchen jedoch schon sehr bald unnötig sein.

Langzeitmessungen am Kernel belegen, dass das deterministische Zeitverhalten unabhängig von der eingesetzten Prozessorarchitektur und auch langzeitstabil ist [QuKu2012-63]. Das wird durch die sogenannten High Resolution Timer (hrtimer) und den Tickless-Betrieb erreicht. Das Zeitmanagement des Kernels basiert intern nicht mehr auf Timerticks, sondern auf einer Zeitbasis in Nanosekundenauflösung. Zeitaufträge werden also nicht mehr mit dem nächsten Timertick, sondern zu exakt den Zeitpunkten angestoßen, zu denen der Auftrag terminiert ist.

Das für ein Realzeitbetriebssystem typische prioritätengesteuerte Scheduling unterstützt Linux bereits seit vielen Jahren, ebenso wie Mutexe, die eine sogenannte Prioritätsvererbung ermöglichen.

Linux zeichnet aber noch mehr aus. Mithilfe von Control-Groups lassen sich Tasks gezielt auf einzelne Kerne einer Multicore-Maschine fixieren und per Threaded Interrupts werden Interrupt-Service-Routinen als priorisierbare Threads abgearbeitet.

Das Applikationsinterface orientiert sich für Realzeitanwendungen am POSIX-Standard.

Neben den Realzeiteigenschaften weist Linux weitere Merkmale auf, die es für den Einsatz als Embedded System prädestinieren. Wesentliche Eigenschaften sind

- Skalierbarkeit,
- Modularität,
- Portierbarkeit,
- Quelloffenheit,
- eine große Entwicklergemeinde und
- gute und umfangreiche Dokumentation.

Um ein Embedded System zusammenzustellen und zu bauen, können Sie auf diverse Hilfsmittel wie beispielsweise Buildroot (siehe Abschnitt 3.3) oder OpenEmbedded ([http://www.openembedded.org]) zurückgreifen. Buildroot bietet sich für kleinere Projekte an, auf OpenEmbedded greifen Sie erst zurück, wenn Sie sehr komplexe Systeme aufbauen müssen. Beide Programme bauen nicht nur das eigentliche System zusammen, sondern sorgen selbstständig dafür, dass die notwendigen Entwicklungswerkzeuge wie beispielsweise Crosscompiler zur Verfügung stehen.