INTERNETINTEGRATION VON EMBEDDED SYSTEMS

Peter Balog

Fachhochschule Technikum Wien

ZUSAMMENFASSUNG:

Die Kommunikation mit Embedded Systems über Internetprotokolle nimmt stetig an Bedeutung zu. Die notwendige Voraussetzung für die *Internetintegration von Embedded Systems* ist, neben einer entsprechenden Kommunikationsschnittstelle, ein TCP/IP-fähiger Protokollstapel (*TCP/IP-Stack*). Der vorliegende Beitrag skizziert eine Variante der Integration eines *TCP/IP-Stacks* mit dem Betriebssystemkern µC/OS-II für eine C167 basierte Mikrocontrollerplattform mit Ethernet-Schnittstelle, wie sie derzeit in diversen Lehrveranstaltungen der Fachhochschule Technikum Wien angewandt wird.

EINLEITUNG

Der Begriff *Internetworking* ist in der Fachwelt ein Schlagwort, das immer wieder in Diskussionen und Fachartikeln auftaucht. Gemeint ist mit *Internetworking* die Fähigkeit von Computern, elektronischen Geräten und Systemen ganz allgemein, mit dem Internet in Verbindung treten zu können /1/. Über dieses Medium können dann Daten jeglicher Art mittels Computer an einem beliebigen Ort (gegebenenfalls über Funk) zu einer beliebigen Zeit ausgetauscht werden. Wird nun dieser Begriff auf *Embedded Internetworking* erweitert, so meint man *Internetworking* mit Embedded Systemen.

Embedded Internetworking ist bereits Realität und wird sich mit zunehmender technologischer Entwicklung sehr schnell verbreiten. Diese Vorhersage „die Anzahl der am Internet angeschlossenen elektronischen Geräte wird schneller wachsen als die Zahl der Personal Computer (PCs)“ basiert auf folgenden zwei Fakten:

- Bereits heute sind Mikrocontroller den PCs zahlenmäßig bei weitem überlegen.
- Jeder noch so billige Mikrocontroller wird immer leistungsstärker und stellt somit genügend Rechenleistung zur Verfügung, um ausreichend Ressourcen für den Netzwerktransfer zu haben. In den meisten Applikationen wird ein neues Design verwendet, welches einen leistungsstärkeren Mikrocontroller (breitere Datenworte und schnellerer Systemtakt) als im vorhergehenden Design beinhaltet. Netzwerk-funktionalität kann bereits in Hardware integriert sein oder kann durch minimale zusätzliche Aufwendungen hinzugefügt werden.

Es ist also nur eine Frage der Zeit bis der Punkt erreicht wird, bei dem die Anzahl der *Embedded Systems* "*online*" die Anzahl der PCs, die ans Internet angeschlossen sind, übersteigt.

Es gibt viele potentielle Anwendungen im Bereich *Embedded Internetworking*. Nicht alle benötigen als Netzwerk zwingend das Internet – es erleichtert nur den Gebrauch der Applikation. Durch den Einsatz des Internets kann der Benutzer das gewohnte Interface (*Web-Browser*, *Telnet-Client*, etc.) verwenden, um die Applikation zu steuern oder benötigte Informationen abzurufen.

Braucht unsere Gesellschaft wirklich Toaster oder Kaffeemaschinen mit Internetanschluss? Welche tatsächlichen Vorteile würde die Gesellschaft durch das Anbinden der bereitgestellten Applikationen an das Internet erhalten?

Die Antworten darauf sind zahlreich und zum Teil auch widersprüchlich. Mögliche und sinnvolle Applikationen sind etwa *Software-Updates*, die automatisch über das Internet durchgeführt werden oder Wartungsprogramme, die selbständig im Hintergrund die Funktionalität des Systems gewährleisten und eventuell auftretende Probleme dem Benutzer sowie Hersteller zuverlässig und schnell zur Kenntnis bringen.

Grundlage für die *Internetintegration von Embedded Systems* ist ein *TCP/IP Stack*, der entweder als dedizierter Teil einer Anwendung oder im Zusammenspiel mit einem Echtzeitbetriebssystem implementiert werden kann.

ZIELE

Entsprechend der Bedeutung dieser Thematik wurde auch das Angebot an Lehrveranstaltungen unserer technischen FH-Studiengänge aus dem Bereich der Informations- und Kommunikationstechnologie um die *Internetintegration von Embedded Systems* erweitert. Mit der gewählten Hardwareplattform (Keil MCB167net) konnte sowohl die bekannte und bewährte Softwareentwicklungsumgebung (Keil µVision2 für C167 Mikrocontroller) als auch der Betriebssystemkern inklusive Portierung (µC/OS-II Rel. 2.5x) beibehalten werden.

Um nun *Embedded Internet Applications* realisieren zu können, ist ein *TCP/IP-Stack* erforderlich. Ausgehend von allgemeinen Ansätzen /2/, bestehenden Teillösungen und frei verfügbaren Softwaremodulen wurden im Rahmen einer Diplomarbeit /4/ Lösungsvarianten diskutiert, und eine einfach Einprozesslösung für den Betriebssystemkern µC/OS-II implementiert.

TCP/IP-STACK INTEGRATION

Unter der gegebenen Voraussetzung, dass ein multitaskingfähiger Betriebssystemkern vorhanden ist, gibt es prinzipiell drei Möglichkeiten, TCP/IP-Funktionalität zu realisieren. Eine mögliche Implementierungsvariante stellt die **Mehrprozesslösung** dar. Diese Variante geht von der Philosophie aus, jedes einzelne Protokoll in einem eigenen Prozess bzw. Task ablaufen zu lassen. Voraussetzungen sind ein strikter Protokollaufbau (*Layering*) und eine genaue Definition der Kommunikationspunkte zwischen den einzelnen Protokollschichten. Dieser Ansatz bietet einige Vorteile wie z.B. die Möglichkeit des Hinzufügens von Protokollen während der Laufzeit, ein einfacheres Verständnis hinsichtlich des Codes und dadurch vereinfachtes Debuggen des Protokollstapels und der darauf laufenden Anwendungen. Offensichtlich sind aber auch die Nachteile, wie z.B. der Overhead der einzelnen Kommunikationsschichten (Protokolle) und vor allem der *Context-Switch,* der ausgeführt werden muss, sobald von einer Protokollschicht zur nächsten Protokollschicht gewechselt wird. Für ein eingehendes TCP-Segment würde das folgendes bedeuten:

1. *Context-Switch* vom Netzwerk-Interface-Prozess zum IP-Prozess
2. *Context-Switch* vom IP-Prozess zum TCP-Prozess
3. *Context-Switch* vom TCP-Prozess zum Applikations-Prozess

Eine andere mögliche Implementierung besteht darin, den TCP/IP-Stack direkt in den Betriebssystemkern einzubetten. Im Falle dieser ***Kernel*-Lösung** müssen die Applikationsprozesse mit dem Protokollstapel direkt über Systemaufrufe (*System Calls*) kommunizieren. Da hier die Kommunikationsprotokolle nicht streng voneinander getrennt sind, kommt es zur Überschneidung der einzelnen Protokollschichten (*Layer*). Der Vorteil dieses Ansatzes liegt sicher daran, dass aus Sicht der Anwendungsprogrammierung nur eine Schnittstelle zu einem „Betriebssystem mit TCP/IP-Funktionalität“ gegeben ist.

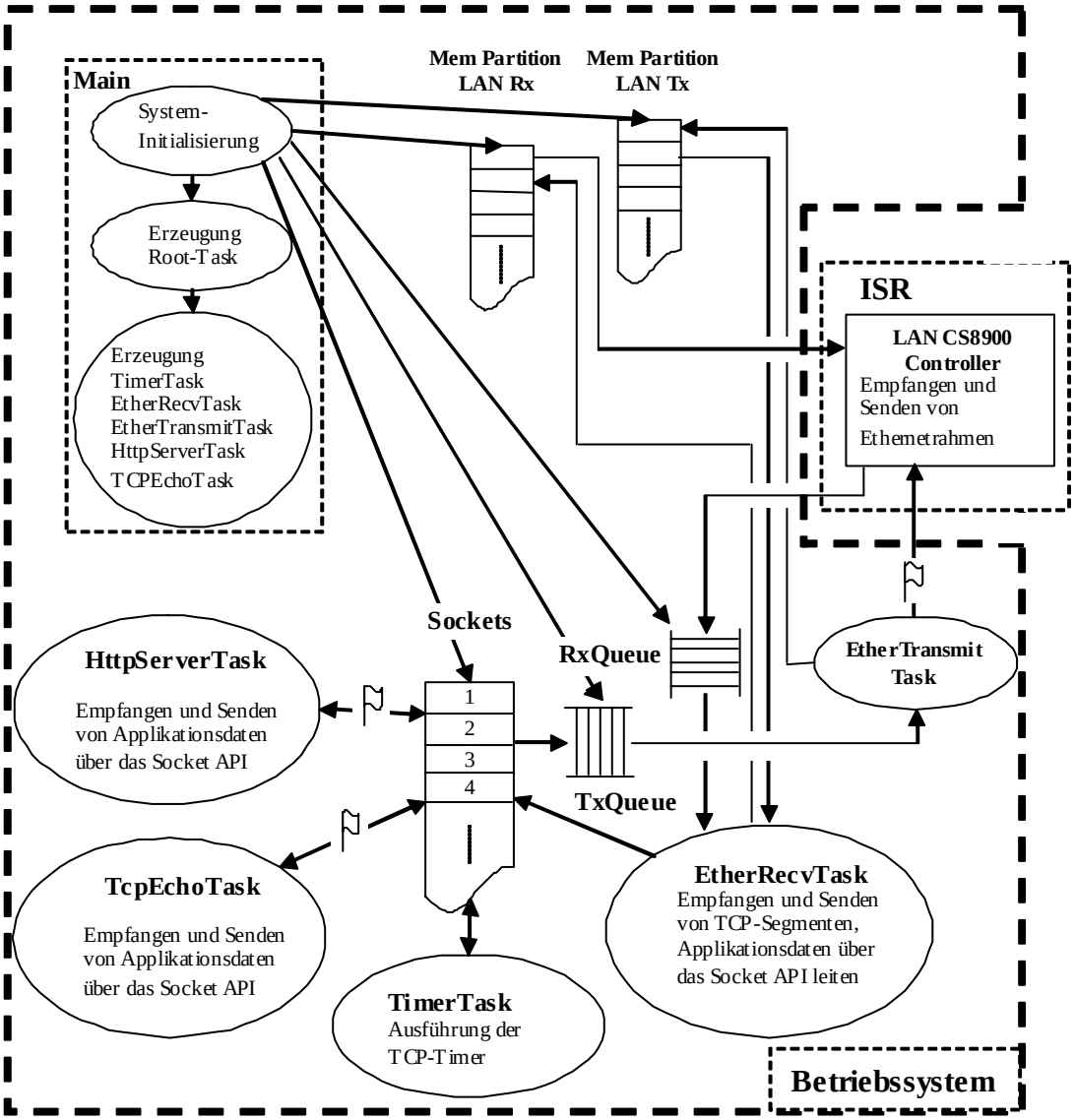
Unsere Lösung ist eine **Einprozesslösung**; d.h. der TCP/IP-Stack läuft zur Gänze in einem einzelnen Prozess ab und ist somit vom Betriebssystemkern getrennt. Applikationsprogramme können innerhalb des TCP/IP-Prozesses oder in separaten Applikationsprozessen ausgeführt werden. Die Kommunikation zwischen den Applikationen und dem TCP/IP-Stack kann entweder über einzelne Funktionen (nur möglich, wenn das Applikationsprogramm innerhalb des TCP/IP-Prozesses ausgeführt wird) oder durch das implementierte abstrakte Socket-API erfolgen. Als Hauptvorteil dieser Lösungsvariante kann angeführt werden, dass Portierungen auf andere Betriebssysteme infolge der Kapselung meist leichter und "schmerzfreier" durchgeführt werden können. Nachteilig ist sicherlich, dass, wie bei der *Kernel*-Lösung, keine klare Trennung zwischen den Protokollschichten gegeben ist.

IMPLEMENTIERUNG

Der äußere Rahmen der folgenden Abbildung symbolisiert die Einbettung des *TCP/IP-Stacks* und der darauf aufsetzenden Anwendungen in das Betriebssystem. Der Programmstart mit der Funktion `main()` ist als dünn umrandetes Rechteck abgebildet. Der Block außerhalb stellt die ISR (Interrupt Service Routine) dar, die für die Übernahme der Ethernet Frames vom Ethernet Controller verantwortlich ist.

Der Netzwerktreiber, also die Schicht unterhalb der IP-Schicht, liegt außerhalb des eigentlichen *TCP/IP-Stacks* und ist mit zwei parallelen Prozesse zum Senden und Empfangen von Ethernet-Frames, und einer Interrupt-Service-Routine, die den Event-Handler zum Ethernet-Controller darstellt, implementiert. Die Software-Schnittstelle zum *TCP/IP-Stack* bzw. zu direkten Ethernet-Anwendungen wird über zwei µC/OS-II Message Queues realisiert. Durch die Anwendung der Intertask Kommunikations- und Synchronisationsobjekte ist somit weder ein direkter Zugriff auf die Hardware noch auf die Treiberfunktionen notwendig.

Die wesentliche Software-Schnittstelle zwischen TCP/IP-Anwenderprozessen und dem *TCP/IP-Stack* wird über die Socket-Datenstruktur realisiert. Die Anwenderprozesse verwenden ein, an das UNIX-BSD-Socket-Interface /3/ angelehntes, API, welches Funktionen wie *bind(), listen(), read(), write(),* etc. zur Verfügung stellt.



Neben dem eigentlichen *TCP/IP-Stack* und der Implementierung der Netzwerkschicht wurden ein HTTP-Dienst und ein TCP-ECHO-Dienst als Testfälle implementiert. Ausgehend vom Quelltext dieses Projekts können sich die Studierenden einerseits leicht in den *TCP/IP-*

Stack einarbeiten und andererseits sehr schnell eigene eingebettete Internetapplikationen realisieren.

AUSBLICK

Die oben skizzierte Implementierung steht als fertiges Projekt für die Keil-IDE zur Verfügung. Testanwendung, *TCP/IP-Stack* und Betriebssystemkern stellen eine Applikation dar, welche für die freie Evaluierungsversion der Keil-IDE (8kByte Speicherlimit) zu groß ist. Derzeit muss deshalb mit der lizenzpflichtigen Vollversion der Software-Entwicklungsumgebung gearbeitet werden. Um künftig die Anwendungen wieder mit der Evaluierungsversion der Keil-IDE entwickeln zu können, muss eine Flashportierung des *TCP/IP-Stacks* gemeinsam mit dem Betriebssystemkern entwickelt werden. Im Vorfeld ist geplant, einerseits die vorliegende Lösung zu optimieren und andererseits eine *Kernel*-Lösung zu entwerfen und zu untersuchen.

LITERATUR

1. W. Richard Stevens, “TCP/IP Illustrated,Volume 1, The protocols”, Addison Wesley, 1994, ISBN 0-201-63346-9
2. Gary R. Wright, W. Richard Stevens, “TCP/IP Illustrated,Volume 2, The Implementation”, Addison Wesley, 1995, ISBN 0-201-63354-X
3. W. Richard Stevens, “UNIX Network Programming,Volume 1, Networking APIs: Sockets and XTI”, Prentice Hall, 1998, ISBN 0-13-490012-X
4. Martin Zauner, “Implementierung eines TCP-IP-Kontrollstapels in ein Echtzeitbetriebssystem für ein Mikrocontroller-system”, ausgeführt am Technikum Wien, 2003