ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Robert Natau, Matthias Bader

Porting TOPSY v3 to the IXP2400

Semester Thesis SA-2007-18
November 2006 to March 2007

Tutor: Dr. Lukas Ruf
Supervisor: Prof. Dr. Bernhard Plattner

Abstract

TOPSY ist ein modulares Microkernel-Betriebssystem, das an der ETH Zürich für Unterrichtszwecke entwickelt wurde. In einer weiteren Entwicklung zu TOPSY v3 wurde dieses um eine leistungsfähige Speicherverwaltung erweitert und architektonisch verfeinert, um auch als Node OS in Netzwerkumgebungen eingesetzt werden zu können. Im Rahmen unserer Semesterarbeit im Winter 2006/2007 portierten wir TOPSY v3 auf den Intel IXP2400-Netzwerkprozessor und eine Emulatorplattform. Wir führen in die Thematik der Funktionsweise und der Programmierung der hardwarenahen Module von TOPSY v3 ein, und ermöglichen dem Leser, anhand dieses Dokuments unsere Arbeit nachzuvollziehen. Dieser Bericht fasst unsere Arbeit und die Lösungsansätze zusammen, die es uns erlaubten, diese Semesterarbeit erfolgreich abzuschliessen.

Abstract

TOPSY is a modular microkernel operating system which has been developped at the ETH Zürich with applications in teaching in mind. In a separate effort, it was extended by an advanced memory management, refined architecturally, and reissued as TOPSY v3. For our semester thesis in winter 06/07, we ported this system to the IXP2400 network processor hardware and an emulator platform. In this report, we give an introduction to the functionality and the programming of the hardware abstraction layer of TOPSY v3 and strive to enable the reader to reproduce our work by giving all the necessary information within this document. This report summarizes our work and the approaches which allowed us to complete this term paper successfully.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die Betriebssystemprogrammierung ist ein interessantes und anspruchsvolles Betätigunsfeld. Sie ist nicht nur von Bedeutung für die Entwicklung und Pflege dieser Softwareprodukte selbst, sondern bietet auch den Entwicklern von Anwendungsprogrammen die Wissensgrundlage für das Schreiben von effizienten und zuverlässigen Anwendungen innert kurzer Zeit. Aus diesem Grund entstand an der ETH Zürich das Bedürfnis, für Unterrichtszwecke ein von den Sachzwängen produktiv einzusetzender Software freies, eigenes Betriebssystem zu schaffen, um es im Rahmen der Lehrtätigkeit einsetzen zu können - TOPSY, das Teachable OPerating SYstem [11].
TOPSY ist ein kompaktes und modulares Betriebssystem mit präemptivem Multitasking [3]. Es wurde am Elektrotechnik- und Informationstechnologiedepartement der ETH Zürich von George Fankhauser entwickelt und in der Folge bis zum Ende des Sommersemesters 2006 in der Lehre verwendet.
Um eine leistungsfähige Speicherverwaltung erweitert und unter dem Namen TOPSY v3 neu lanciert wurde TOPSY in einer Masterarbeit von Lukas Ruf mit dem Ziel, dieses Betriebssystem auch als Node OS[13] auf Netzwerkgeräten einzusetzen. Ausserdem arbeiteten rund ein Dutzend Studenten an verschiedenen Projekten zur Weiterentwicklung und Portierung von TOPSY und TOPSY v3 mit.

1.1 Motivation

Im Rahmen einer Semesterarbeit stellte sich die Aufgabe, dieses Betriebssystem auf eine neue Plattform zu portieren. Diese abwechslungsreiche und herausfordernde Arbeit hatte zum Inhalt, den wenigen auf Netzwerkprozessoren [5, 9, 15] etablierten Betriebssystemen ein flexibles, freies, kompaktes Betriebssystem mit einer fortschrittlichen Architektur gegenüberstellen zu können, das auch in Zukunft aktiv weiterentwickelt werden soll. Ausserdem bot es die Gelegenheit, wertvolle praktische Erfahrung und tiefe Einblicke in die Betriebssystemprogrammierung zu gewinnen und an einem in der praxiserprobten Produkt mitzuarbeiten.

1.2 Dokumentstruktur

Dieses Dokument führt in die betroffenen Themenbereiche ein und legt Rechenschaft über das Vorgehen bei der Erfüllung dieser Aufgabe ab. In den verbleibenden Abschnitten dieses Berichts geben wir zunächst einen Überblick über relevante Vorleistungen anderer Entwickler, auf denen unsere Arbeit aufbaut. Im Anschluss führt Kapitel 3 in die Einzelheiten der gestellten Aufgabe ein, und Kapitel 4 beschreibt die Vorgehensweise, die wir wählten, um unsere Arbeit effizient angehen zu können und zuverlässig zu evaluieren und zu validieren. Das nachfolgende Kapitel geht auf die Einzelheiten der Implementation ein und beschreibt in je einem Abschnitt die vorgenommenen Änderungen an der Codebasis von TOPSY. Die Folgekapitel beurteilen die Erreichung der gesteckten Ziele, geben einen Überblick über die verwendete Methodik zur Evaluation und beleuchten ohne Anspruch auf Vollständigkeit diejenigen Bereiche, die sich für

Verbesserungen und Erweiterungen der Funktionalität anbieten. Der Bericht schliesst mit einer Zusammenfassung und einigen Eindrücken, die für vergleichbare Arbeiten von Bedeutung sein könnten.
In den Anhängen finden sich zusätzliche Informationen, die es dem Leser erlauben, unsere Arbeit nachzuvollziehen: Die Konfiguration der verwendeten Toolchain und Angaben zur Funktionsweise und Veränderungen des Binary Patchers, Angaben zum Code Cleanup, der aufgestellte Zeitplan sowie die Aufgabenstellung, die dieser Arbeit zugrunde liegt.

1.3 Acknowledgements

Der grösste Dank gebührt unserem Betreuer, Herrn Dr. Lukas Ruf. Es war die angenehme Art der Zusammenarbeit, seine Begeisterung für das Projekt, und nicht zuletzt sein Erfahrungsschatz und das Fachwissen, was uns überhaupt befähigte und motivierte, diese Arbeit erfolgreich auszuführen. Zudem möchten wir Herrn Prof. Bernhard Plattner danken für die Bereitstellung der Ressourcen dieser spannenden Semesterarbeit, seine konstruktive Kritik und die Führung seines Departements, die eine Arbeitsatmosphäre wie die beschriebene zulässt.
Nicht unerwähnt bleiben darf auch Dipl. Ing. Hampa Hug, der mit der Entwicklung des ARM-Emulators [1] eine hervorragende Leistung vollbracht und uns einen unschätzbaren Dienst erwiesen hat, und ohne dessen Hilfsbereitschaft und kompetente Unterstützung dieser Bericht nicht in seiner heutigen Form vorliegen könnte.

[1] Wir sind uns der Tatsache bewusst, dass der Gebrauch dieses Begriffes der geläufigen Definition zuwiderläuft. Da aber der Autor diese Software ebenfalls als Emulator bezeichnet und sich dieser Begriff während der Entwicklung eingebürgert hatte, verzichten wir darauf, die Software als Emulator und den Vorgang als Simulation zu bezeichnen, und verwenden im Interesse der Einheitlichkeit durchgängig die Begriffe Emulator und Emulation.

Kapitel 2

Related Work

In seiner heutigen Form realisiert TOPSY v3 [12] beispielhaft einen radikalen Microkernel-Approach, der aus Sicherheits- und Stabilitätsüberlegungen den mit vollen Privilegien auszuführenden Code auf ein Minimum zu beschränken sucht. Im Unterschied zu den gängigsten Betriebssystemen werden nicht nur Kernel- und User-Mode [3] unterstützt, sondern praktisch beliebig viele, voreinander geschützte sogenannte Protection Domains, welche mit unterschiedlichen Rechten und Prioritäten versehen werden und die Ausführung von Anwendungsprogrammen – aber auch von Device Drivern oder den Modulen des Kernels selbst – mit feinkörniger Staffelung der Zugriffsprivilegien gestatten.
Während viele andere Betriebssysteme hinsichtlich herausragender Performance optimiert wurden, steht bei TOPSY der Einsatz im Lehrbetrieb im Vordergrund. TOPSY wurde als Anschauungsmodell und für praktische Übungen in der Vorlesung “Technische Informatik II” der ETH Zürich, und an weiteren Universitäten im deutsch- und englischsprachigen Ausland, verwendet, um den Studenten die Grundlagen der OS-Programmierung zu veranschaulichen, ohne sie mit den komplizierten Algorithmen und Architekturen der üblicherweise produktiv eingesetzten Betriebssysteme wie Linux, Unix oder Windows vertraut machen zu müssen. Deshalb wurde bei der Entwicklung von TOPSY stets auf übersichtliche Architektur und verständlichen, leserlichen Code Wert gelegt, auch wenn damit stellenweise Einbussen bei der Performance in Kauf genommen werden mussten. Ausserdem erlaubt die rigorose Trennung von hardwarenahem und hardwareunabhängigem Code die einfache Portierung auf eine Vielzahl von unterschiedlichen Plattformen.
Inzwischen läuft die Familie der TOPSY-Betriebssysteme nebst der Referenzarchitektur MIPS R3000 [18] auch auf den Intel x86- [21], StrongARM- [19] und Motorola Dragonball 68000-Plattformen [20] sowie als Applikation im Linux-User-Mode, und unterstützt damit Desktop-PCs, Erweiterungskarten für Server sowie einzelne Handhelds von Palm und Compaq. Seit der Fertigstellung dieser Semesterarbeit wird auch eine PCI-Erweiterungskarte auf Basis des Intel IXP2400-Netzwerkprozessor unterstützt.
Im folgenden werden einige Vorgängerarbeiten vorgestellt, die für das Erreichen dieses Ziels von besonderer Bedeutung waren.

2.1 TopsyIXP (Semesterarbeit)

2.1.1 Motivation und Fokus

Nachdem TOPSY 1.1 erfolgreich entwickelt und in der Lehre eingesetzt worden war, verfolgte diese Semesterarbeit das Ziel, TOPSY auch auf den Core-Prozessor der damals neuartigen und viel beachteten Intel-IXP1200-Netzwerkprozessoren zu portieren. Damit konnte einerseits die Portierbarkeit des Systems beispielhaft unter Beweis gestellt werden, andererseits rückte dadurch auch die Vision des Einsatzes als Node OS, also auf einem eingebetteten Netzwerksystem, näher.

2.1.2 Leistungen und Resultate

In der zur Verfügung stehenden Zeit implementierten die Autoren den Start-Up-Code von TOPSY auf dem IXP1200, passten das Memory Management an die neuen Gegebenheiten an und vervollständigten die hardwarenahen Codebereiche. Ausserdem schufen sie die Grundlage für spätere Versionen von TOPSY auf der IXP1200-Plattform und demonstrierten, dass sich TOPSY auch für den Einsatz auf Netzwerkprozessoren eignet.

2.1.3 Einschränkungen und Möglichkeiten zur Weiterentwicklung

Aufgrund von Qualitätsmängeln der eingesetzten Hardware traten während des Projekts Verzögerungen auf, die in der zur Verfügung stehenden Zeit nicht mehr wettgemacht werden konnten. Diese Verzögerungen führten dazu, dass das Resultat der Arbeit zwar die meisten Module des Betriebssystems selbst in lauffähiger Form enthielt, aber keine Möglichkeit, zusätzliche Programme zur Ausführung zu bringen. Diese Mängel konnten erst nach dem Ende der Semesterarbeit in einem separaten Projekt behoben werden.
Zudem baute diese Arbeit auf der Codebasis von TOPSY 1.1 auf, und konnte damit nicht an den Leistungen partizipieren, die im Rahmen der Entwicklung von TOPSY v3 (siehe 2.2) erbracht wurden. Und während die eingesetzte Hardware zum Zeitpunkt der Entwicklung als sehr modern galt, ist der IXP1200-Prozessor inzwischen technisch überholt.

2.2 TOPSY v3 (Masterarbeit)

2.2.1 Motivation und Fokus

Nach und nach setzte sich in der TOPSY-Entwicklergemeinde die Überzeugung durch, dass dieses Betriebssystem noch mehr zu leisten im Stande wäre. Bei der Entwicklunng von TOPSY 1.1 wurden – im Interesse der Einhaltung von Zeitplänen, aber auch in Übereinstimmung mit den damals vorherrschenden Paradigmen – viele die Struktur von TOPSY betreffenden Entscheide getroffen, die die weitere Entwicklung des Projekts grundsätzlich einschränkten. Um diese Beschränkungen zu überwinden, entschlossen sich die Autoren dazu, eine neue Version zu erarbeiten, die sich durch eine fortschrittliche Architektur auch für den Forschungs- und produktiven Einsatz auf eingebetteten Systemen eignen sollte, ohne aber die Prinzipien von TOPSY und dessen Einsetzbarkeit in der Lehre in Frage zu stellen.

2.2.2 Leistungen und Resultate

Im Rahmen dieser Diplomarbeit entstand die Architektur von TOPSY v3, die die Grundlage unserer Arbeit bildet, und lauffähige Implementationen auf dem MIPS R3000- und dem StrongARM-Core des Intel IXP1200-Prozessors. Im Unterschied zur Architektur von TOPSY 1.1 wurde das Microkernel-Modell um das fortschrittliche Konzept der Protection Domains für die Speicherverwaltung erweitert, und die dafür notwendigen internen Anpassungen vorgenommen, ohne die bewährten Interfaces zu den User-Programmen zu verändern. So wurde sichergestellt, dass bestehende Anwendungen mit minimalen Änderungen auch auf dem neuen System ausführbar sein würden. Ausserdem wurde die Trennung zwischen Verwaltungsteilen des Betriebssystems und dem auf die jeweilige Hardware angepassten Code noch strikter vorgenommen, um die Portierung auf weitere Plattformen zu erleichtern.

2.2.3 Einschränkungen und Möglichkeiten zur Weiterentwicklung

Obschon die Arbeit nebst der Implementation auf der MIPS-Referenzplattform als proof of concept eine erste Portierung auf der IXP1200-Plattform enthielt, wurde dadurch nicht bewiesen, dass die Trennung zwischen Kernel- und hardwarenahem Code konsequent vollzogen wurde und sich die neue Architektur auch von Aussenstehenden effizient auf eine neue Plattform portieren lässt. Zudem ist die IXP1200-Plattform inzwischen in die Jahre gekommen und damit als Plattform für ein eingebettetes Betriebssystem nur noch von beschränktem Interesse.

2.3 Porting Topsy To The iPAQ Pocket PC (Semesterarbeit)

2.3.1 Motivation und Fokus

Nach der Fertigstellung von TOPSY v3 wurde in dieser Arbeit die erste unabhängigen Portierung auf eine neue Plattform in Angriff genommen. Die Zielplattform war ein Taschencomputer der iPAQ 3700-Serie, und damit stellte diese Arbeit die erste eigenständige Portierungsleistung von TOPSY v3, und gleichzeitig die erste Portierung auf eine mobile Plattform dar. Der iPAQ, der einen Intel StrongARM-Prozessor [19] einsetzt, stand dabei stellvertretend für eine ganze Klasse von mobilen und Netzwerkgeräten.

2.3.2 Leistungen und Resultate

Seit der Fertigstellung dieser Arbeit läuft TOPSY v3 auch auf dem StrongARM-Prozessor eines mobilen Geräts, und unterstützt das Laden und Ausführen von User-Programmen. Die Portabilität von TOPSY v3 wurde erfolgreich deomnstriert, und abgesehen von einigen Einzelheiten wurde ein störungsfreies Zusammenspiel zwischen Hardware und Betriebssystem erreicht.

2.3.3 Einschränkungen und Möglichkeiten zur Weiterentwicklung

Verzichtet wurden wegen des beschränkten Zeitrahmens und der Zielsetzungen des TOPSY-Projekts auf die Implementation von Stromsparmechanismen, iPAQ-spezifischen Eingabe- und Anzeigegeräten und weiteren Eigenheiten der mobilen Hardware. Ausserdem verwendet das beschriebene iPAQ-Modell einen Intel StrongARM-Prozessor, und damit Hardware, die insbesondere dem im Rahmen der Entwicklung von TOPSY v3 verwendeten Intel IXP1200-Prozessor sehr ähnlich ist. Die Hauptleistung dieser Arbeit bestand nicht in der Portierung an sich, sondern im Erkennen und Beheben von Problemen, die sich bei der Entwicklung und Portierung von Betriebssystemen auf mobilen Plattformen stellen.

Kapitel 3

Aufgabenstellung

Während die Aufgabenstellung im Anhang E auf die vorzunehmenden Arbeiten im Detail eingeht, bleibt die Zielarchitektur weitgehend unerwähnt. Die Gemeinsamkeiten, noch mehr aber die Unterschiede zwischen der IXP1200- und der IXP2400-Plattform waren jedoch für die Erfüllung dieser Aufgabe von zentraler Bedeutung. Die folgenden Abschnitte geben deshalb einen Überblick über die relevanten Eigenschaften des eingesetzten IXP-Prozessors und der PCI-Karte mit Bezug auf diese Arbeit. Anschliessend vergleichen wir die eingesetzte Hardware mit derjenigen unserer Vorgängerarbeit, um eine solide Entscheidungsrundlage für die im Kapitel 4 gewählten Vorgehensweisen zu schaffen.

3.1 Die IXP2400-Plattform RadiSys ENP-2611

3.1.1 Der Intel IXP2400 Netzwerkprozessor

Der Intel IXP2400 ist ein Mitglied der zweiten Generation von Intels Netzwerkprozessorserie. Er wurde Mitte 2002 auf den Markt gebracht und ist primär für den sowohl durchsatz- als auch rechenintensiven Netzwerkeinsatz ausgelegt. Dieser 32bit-Prozessor enthält neben dem ARMv5-kompatiblen XSCALE-Core [2] acht synchron getaktete, autonome Paketprozessoren, die zur Entlastung des Core-Prozessors repetitive Aufgaben wie Paket- und Datenstromklassifizierung, Ver- und Entschlüsselung, Überprüfung auf Viren oder vergleichbare Aufgaben übernehmen können. Diese Paketprozessoren werden in der Intel-Terminologie *Microengines* genannt und verfügen über unabhängige Instruktions- und Datenspeicher, Unterstützung für Hardware-Multithreading und werden im proprietären MEv2-Instruktionssatz [3] programmiert. Die für den IXP2400 angegebenen Kenndaten [4] sind in Tabelle 3.1 zusammengefasst.

Leistungsaufnahme (max.)	16W
integrierter XScale Core:	
Instruktionssatz	ARMv5TE
Taktfrequenz (max.)	600MHz [a]
Instruktionscache	32kb
Datencache	32kb
Mini-Datencache	2kb
integrierte Paketprozessoren:	
Instruktionssatz	Intel MEv2
Taktfrequenz (max.)	600MHz, synchron zum Core
Instruktionsspeicher	8x 4k Instruktionen
Integrierter Datenspeicher	8x 2.5kB

[a] Übertaktung ist offenbar nicht nur möglich, sondern sogar vorgesehen. Für den standardkonformen Betrieb der offiziell unterstützten UTOPIA-Interfaces wird von Intel ein benötigter Prozessortakt von 613MHz vorgeschrieben [16].

Tabelle 3.1: Kenndaten des Intel IXP2400

RadiSys ENP-2611:	
Prozessor	Intel IXP2400
Taktfrequenz	600MHz
Rechenleistung (theor. max.)	(1+8)x 600MIPS = 5.4GIPS
DDR SDRAM	512MB
QDR SRAM	2x 8MB
Speicher-Bandbreite SDRAM (theor. max)	19.2 Gbps
Speicher-Bandbreite SRAM (theor. max)	2x 12.8 Gbps
Leistungsaufnahme (max.)	25W

Tabelle 3.2: Kenndaten der RadiSys ENP-2611-PCI-X-Karte

Ausserdem wurden etliche leistungsfähige, für unsere Arbeit aber unwesentliche, Hardwarebausteine in den IXP2400 integriert, unter anderem eine Hardware-Hashing-Einheit und programmierbare Interfaces für den Anschluss anderer Hochleistungsnetzwerkgeräte. Andere Komponenten sind mehrfach ausgeführt, beispielsweise das RAM-Interface (zweifach, für SRAM und SDRAM) oder die vier Hardware-Timer.
Nebst des Prozessors selbst werden von Intel auch zyklentreue Software-Simulatoren für Core und Packetprozessoren, Linux-SDKs und weitere, den Entwickler unterstützende, Werkzeuge angeboten. Da allerdings auf Grund der restriktiven Lizenzierungspolitik von Intel schon die Prozessor-Referenzmanuals [5] [6] einem Non-Disclosure Agreement unterlagen, verzichteten wir auf diese Hilfsmittel und verwendeten als Hilfsmittel für die Portierung ohne Ausnahme die Open Source-Alternativen, die in der Aufgabenstellung vorgeschrieben wurden.

3.1.2 RadiSys ENP-2611

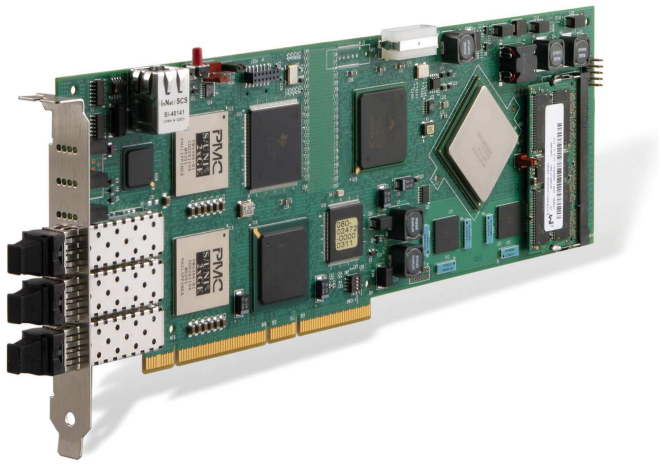

Abbildung 3.1: Die RadiSys ENP-2611-PCI-X-Karte

Für unsere Semesterarbeit wurde uns die IXP2400-Plattform RadiSys ENP-2611 [7] zur Verfügung gestellt, die in Abbildung 3.1 abgebildet ist. Nebst dem Netzwerkprozessor umfasst diese PCI-X-Karte drei 1000base-SX-Anschlüsse, einen 100base-TX-Anschluss für Debugging und Control-Plane-Datenverkehr, 512MB SDRAM, 16MB schnelles SRAM und die nötige Logik zur Verschaltung der Komponenten. Tabelle 3.2 liefert einen Überblick.
Diese Karte ist als Entwicklungsplattform nicht dafür vorgesehen, das Betriebssystem dauerhaft im nichtflüchtigen Speicher zu halten. Deshalb wird über das PCI-Host-Interface bei jedem Systemstart als erstes das Betriebssystem-Image vom integrierten Bootloader in den Speicher geladen. Genauere Angaben dazu finden sich im Abschnitt 5.1. Ausserdem liefert der Hersteller zusammen mit der Hardware ein Softwarepaket, das Hilfssoftware für das Intel IXA SDK, den Bootloader, Debug- und Monitor-Utilities im Flash ROM der Karte und eine detaillierte Dokumentation umfasst [7].

3.1.3 Vergleich mit der IXP1200-Plattform

Als Ausgangspunkt für unsere Arbeit diente die Semesterarbeit 'Porting TOPSY to the IXP1200' unserer Vorgänger Silvio Dragone und David Reist [8]. Im Rahmen dieser Arbeit wurde TOPSY bereits auf einen IXP-Netzwerkprozessor der ersten Generation portiert. Da die Datenblätter dieses Prozessors und der Bericht über die betreffende Arbeit selbst ausführliche Beschreibungen aller relevanten Eingenschaften des IXP1200 enthalten, verzichten wir auf eine detaillierte Beschreibung und präsentieren stattdessen die wichtigsten Unterschiede, die wir diesen Quellen entnehmen konnten [4, 9, 8]:

- Der offensichtlichste Unterschied zwischen der IXP1200- und der IXP2400-Plattform besteht in der Veränderung des *Core-Instruktionssatzes*. Während die Prozessoren der ersten IXP-Generation einen StrongARM-Core und damit den ARMv4-Instruktionssatz einsetzten, verwendet der IXP2400-Core den ARMv5TE-Instruktionssatz. Während diese Änderung für den in C verfassten Code relativ belanglos war, verlangte sie die Überprüfung aller und Änderungen einiger Assemblermodule, angefangen bei der start.S-Routine (siehe 5.2).

- Mit der Einführung des neuen Cores wurden auch die integrierten Subsysteme angepasst, und dabei war für uns der Coprozessor 15 von besonderem Interesse. Diese Prozessorerweiterung bildet die *Schnittstelle zum Memory Management* und war in ähnlicher Form bereits im ARMv4-Standard integriert, wurde aber hinsichtlich Funktionsumfang und Ansteuerung überarbeitet. Die Inbetriebnahmen der Memory Management Unit (MMU) setzte damit die Anpassung dieser Codebausteine voraus, und diese Teilaufgabe wird im Abschnitt 5.4 erläutert.

- Da die in der IXP1200-Arbeit vorgenommene Ansteuerung der *Hardware-Timer* nicht mit der IXP2400-Plattform kompatibel war, zog dies auch Änderungen im Interrupt Handling und im Thread Management nach sich. Auf diese Änderungen wird in 5.5 und 5.6 eingegangen.

- Ausserdem wurden in der IXP-1200-Arbeit Karten von Intel verwendet, während die IXP-2400-Karte von RadiSys gefertigt wurde und beide Systeme unterschiedliche, proprietäre *Bootloader* einsetzen.

Kapitel 4

Methodik- und Architekturentscheide

Nach der Feststellung der in 3.1.3 aufgeführten Unterschiede zwischen Vorgänger- und Zielplattform entschieden wir uns, die Arbeit unter Verwendung eines iterativen Rapid Prototyping-Ansatzes voranzutreiben, auf welchen auch im Abschnitt 6.3 eingegangen wird. Da wir diese Aufgabe als Doppelsemesterarbeit in Angriff nehmen konnten, bot sich ausserdem die Verwendung eines vereinfachten Extreme Programming-Ansatzes [10] an, der die Programmierarbeit in Zweiergruppen vorschreibt und im Interesse einer teamtauglichen Verantworlichkeitsstruktur eine ständig wechselnde Rollenverteilung vorsieht.
Dazu motivierten uns primär folgende Gründe:

- Während eine Aufteilung der Anpassungsarbeiten in disjunkte Teilmengen grundsätzlich möglich schien und in vergleichbaren Arbeiten erfolgreich demonstriert worden war [8], war uns doch klar, dass nicht alle Designentscheide vor dem Beginn der Codierarbeiten getroffen werden können. Ebenfalls von Anfang an klar war aber auch, dass die Portierung innerhalb einer Prozessorfamilie keine grundsätzlichen Änderungen der Struktur von TOPSY bedingen würden. Um Reibungsverluste und das Risiko von Inkompatibilitäten zu reduzieren und gleichzeitig die Motivation aller Beteiligten auf hohem Niveau zu halten, entschieden wir uns deshalb, jeweils gemeinsam denselben Codebereich zu bearbeiten.

- Während das Hauptziel der Arbeit darin bestand, innert eines Semesters TOPSY v3 auf der IXP2400-Plattform zur Lauffähigkeit zu bringen, wollten wir auch die Gelegenheit ergreifen, in allen berührten Teilgebieten der Betriebssystemprogrammierung praktische Erfahrung zu sammeln und so unseren Lernerfolg zu steigern.

Deshalb gingen wir die im Kapitel 5 beschriebenen Arbeiten nach und nach an und überprüften unseren Fortschritt, indem wir regelmässig bis zum jeweils letzten lauffähigen Punkt testeten. Auf die Themen der Validierung und Evaluation wird im Kapitel 6 nochmals näher eingegangen.

Kapitel 5

Implementation

Dank der modularen Microkernel-Architektur von TOPSY lassen sich die für diese Arbeit wichtigen Codeabschnitte sehr genau bestimmen. Da TOPSY in einen hardwareabhängigen und einen hardwareunabhängigen Bereich unterteilt ist und wir keine Erweiterung der Funktionalität anstrebten, war für diese Aufgabe primär die hardwareabhängige, unterste Schicht, das Hardware Abstraction Layer (HAL), von Interesse. Abbildung 5 vermittelt einen Überblick über den Aufbau des Betriebssystems und zeigt horizontal die Hardware Abstraction-, Kernel- und User-Schicht, und innerhalb der Blöcke im Privileged-Bereich die verschiedenen Module, die TOPSY umfasst.
Während der Kernel die eigentlichen Verwaltungsaufgaben eines Betriebssystems wie Thread-

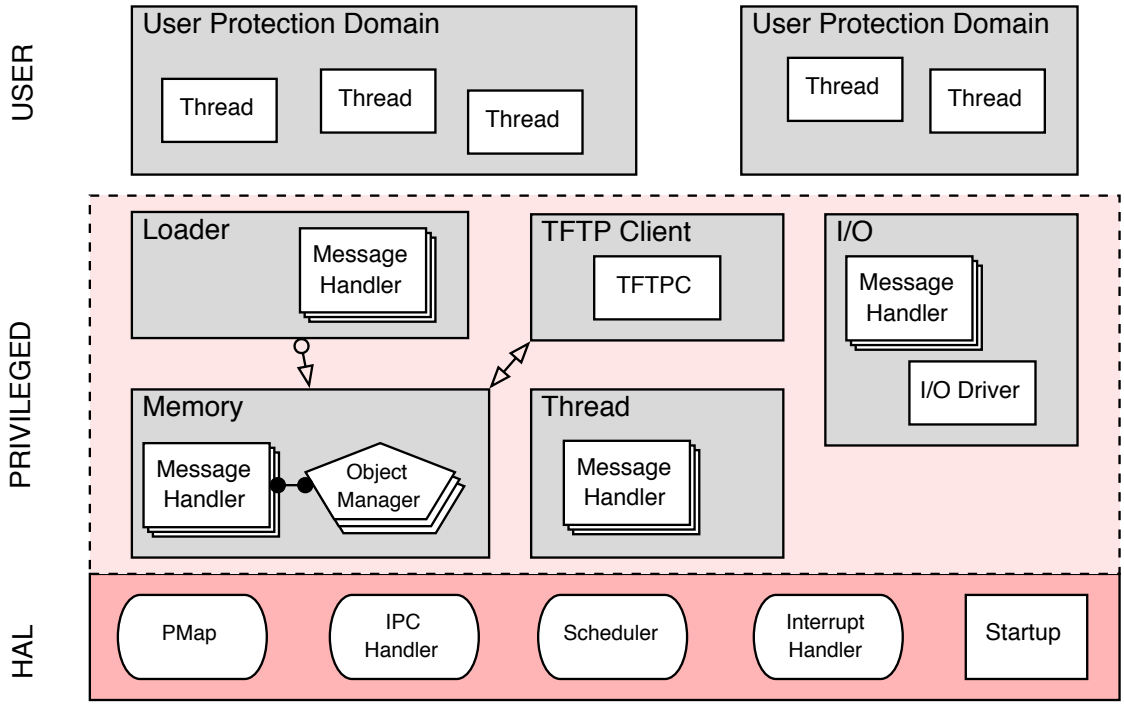

Abbildung 5.1: Der modulare Aufbau von Topsy

und Memory-Management übernimmt und die dafür nötigen Mechanismen (System Calls und Message Passing) [3] bereitstellt, adaptiert das Hardware Abstraction Layer den Kernel an die jeweilige Hardware. Es stellt beispielsweise dem Thread Manager über den Interrupt Handler die Zeitbasis zur Verfügung, die das präemptive Multitasking benötigt. Bevor aber diese Module im Einzelnen vorgestellt werden, soll im Abschnitt 5.1 auf den Startvorgang eingegangen werden, da von dessen erfolgreicher Ausführung alle weiteren Module abhängig sind.

5.1 Boot-Ablauf

Nach dem Einschalten der Versorgungsspannung beginnt der IXP2400 mit der Hardware-Initialisierung und dem Selbsttest. Darauf wird der Bootloader ausgeführt, erst dann beginnt die Abarbeitung des eigentlichen TOPSY-Codes. Dieser Abschnitt soll die Komponenten, die nicht Teil von TOPSY sind, in einem Überblick vorstellen, bevor im Anschluss detaillierter auf die einzelnen TOPSY-Module und ihre Funktion eingegangen wird. Da die Entwicklung sowohl auf der Hardware als auch in der Emulator-Umgebung stattfand, beschreiben die folgenden zwei Abschnitte kurz die jeweils nötigen Schritte für eine Inbetriebnahme der beiden Systeme.

5.1.1 Laden des Boot-Image in den Arbeitsspeicher

Im Flash ROM der ENP-2611-Karte ist ab Werk ein Bündel von Hilfsprogrammen installiert. Da wir jedoch die gesamte Entwicklungsarbeit in der wesentlich geeigneteren Emulator-Umgebung vornahmen, verzichteten wir auf die On-board Diagnostic Suite und den Monitor, und verwendeten nur den Boot Manager für das Laden der Betriebssystem-Images. Unter der Annahme, dass das Host-Betriebssystem ein Linux-System ist, gliedert sich dieser Vorgang in folgende Schritte:

- Öffnen einer SFTP-Verbindung auf den Host, der die IXP2400-Karte beherbergt, um die `kernel.bin`- und `user.bin`-Images hochzuladen.
- Bereitstellen des zu ladenden Images im Verzeichnis ~/tftp/

  ```
  sftp user@ixp2k4.host.name
  cd tftp
  put user.bin
  put kernel.bin
  ```

- Öffnen einer ersten SSH-Verbindung, um die Konsole anzuzeigen:

  ```
  ssh user@ixp2k4.host.name
  sudo minicom -C minicom-ixp-$(hostname)-‘date +%Y%m%d‘.log ixp
  ```

- Öffnen einer zweiten SSH-Verbindung, um die IXP2400-Karte zu resetten:

  ```
  ssh user@ixp.host.name
  sudo enptool reset
  ```

  Nach dem Erscheinen der Anzeige

  ```
  == Executing boot script in 2.000 seconds - enter ^C to abort
  ```

  im ersten SSH-Terminal muss die Ausführung des Bootscriptes mit Ctrl-C abgebrochen werden, damitImages mit den folgenden Befehlen manuell geladen werden können:

  ```
  load -r -b 0xc208000 -m tftp -h 192.168.4.2 /var/tftp/kernel.bin
  load -r -b 0xd000000 -m tftp -h 192.168.4.2 /var/tftp/user.bin
  ```

  Die Eingaben werden jeweils quittiert mit

  ```
  Raw file loaded 0x0c208000-0x0c22589c
  Raw file loaded 0x0d000000-0x0d0090a0
  ```

- Der Bootvorgang wird gestartet mit

  ```
  RedBoot> go 0xc208000
  ```

  und erzeugt folgende Ausgabe (stark gekürzt), die mit dem TOPSY v3-Prompt endet:

```
Topsy 2.1 (c) 1996-2001, ETH Zurich [Arch:ixp2400,
bert@bert, Mon Mar 12 15:40:47 CET 2007]Stack @ 0x0c234fbc
vm segmap:
physical mappings:
0x00000000 (0000) - 0x10000000 (0x10000)
code mappings:
text: 0x0c208000 - 0x0c21e508 (0x16508)
data: 0x0c21f000 - 0x0c22355c (0x455c)
bss : 0x0c22355c - 0x0c223e2c (0x8d0)
bootsack: 0x0c233000 - 0x0c235000 (0x2000)
opt. segment: 0x0d000000 - 0x0d008cd0 (0x8cd0)
[...]
vm_manager_init: anonymous pager loaded
vm_manager_init: file manager loaded
vm_manager_init: page stealer loaded
vm_manager_init: loan-out manager loaded
vm_manager_init: direct mapped manager loaded
Kernel Protection Domain: mapping regions ...
starting tmMain...
starting virtual Memory Manager...
[...]
Topsy Shell 3.0 (c) 1998, ETH Zurich, TIK
Type 'start licence' to print the licence terms governing your use
of this product.

($Id: Shell.h 2911 2007-03-07 13:27:12Z rnatau $)
```

5.1.2 Laden des Boot-Image in den Emulator

Im Unterschied zur Situation, die sich in den Vorgängerarbeiten präsentierte, waren wir in der Lage, nicht nur auf der eigentlichen Plattform zu entwickeln, sondern verfügten mit dem IXP2400-Emulator [1] über ein Werkzeug, das zu beliebigen Zeitpunkten vollständigen Einblick in praktisch alle Register und den gesamten Speicher ermöglichte. Er diente als verlässliche, praxiserprobte Basis für unsere Entwicklungsarbeit, was wir mit selbstgeschriebenen Debug-Routinen nicht in dieser Form zur Verfügung gehabt hätten.
Der von uns verwendete Emulator wurde in einer Masterarbeit [1] von Hampa Hug erstellt und emuliert (nebst vielen anderen Plattformen) den IXP2400-Core und dessen Coprozessoren, das via SRAM- und SDRAM-Schnittstelle angeschlossene RAM sowie die serielle Schnittstelle. Im Unterschied zu anderen Produkten, beispielsweise dem in Abschnitt 3.1.1 erwähnten Intel-IXP-Emulator, ist die Emulation allerdings nicht notwendigerweise zyklentreu, und lässt damit grundsätzlich die Möglichkeit eines sich von der Hardware unterscheidenden Timingverhaltens zu.
Dieser Emulator wird nicht über die Kommandozeile konfiguriert, sondern bezieht von dort lediglich den Pfad zu einer Konfigurationsdatei. Neben anderen Angaben enthält diese auch die Pfade zu den zu ladenden Images. Eine für die Emulation von TOPSY v3 auf dem IXP2400 geeignete Konfigurationsdatei ist im folgenden wiedergegeben:

```
# simarm.cfg

section cpu {
        model = "armv5"
        bigendian = 1
}
section rom {
        base = 0xffff0000
        size = 65536
}
section ram {
        base = 0x00200000
```

```
        size = 0x0fddf000
}
section load
{
        format = "binary"
        base   = 0x0C208000
        file   = "kernel.bin"
}
section load
{
        format = "binary"
        base   = 0x0D000000
        file   = "user.bin"
}
section timer {
        scale = 5
}
section serial {
        io  = 0xc0030000
        irq = 2
}
section console {
        serial = 0
}
```

Unter der Voraussetzung, dass bereits eine geeignete Konfigurationsdatei mit dem Namen `simarm.cfg` vorliegt, gliedert sich der Startvorgang in folgende Schritte:

- Starten des Emulators mittels

  ```
  simarm -c simarm.cfg
  ```

- Öffnen eines separaten Terminals zur Darstellung der Ausgabe der seriellen Schnittstelle:

  ```
  tail -f uart.out
  ```

- Setzen von Breakpoints (optional), beispielsweise auf die Adresse 0xc208000:

  ```
  bs c208000
  ```

- Die Ausführung wird gestartet mit dem Befehl

  ```
  gb
  ```

  und wird erst unterbrochen, wenn ein Breakpoint erreicht oder die Tastenkombination Ctrl-Space eingegeben wird.

5.2 Die Startroutine start.S

Nach dem Laden des Betriebsystem-Images springen Bootloader und Emulator an die physikalische Adresse 0xc208000. An diese Speicherstelle wurde bereits das start.S-Assemblerfile geladen, das die Hardware initialisiert und die korrekten Parameter (beispielsweise die Endianness, Translation Modes und MMU-Erst-Initialisierung) in die Kontrollregister lädt. Diesen Initialisierungsschritten kommt deshalb besondere Bedeutung zu, weil sie den Prozessor an die in der Toolchain eingestellten Parameter anpassen. Wann immer diese Parameter geändert werden, müssen auch Einstellungen in der Toolchain angepasst werden, um die Konfiguration des Prozessors und die Konfiguration des Compilers wieder zur Übereinstimmung zu bringen. Die für TOPSY auf dem IXP2400 verwendeten Toolchain-Einstellungen finden sich im Anhang A.

5.3 Input/Output

Aus Architektur- und Sicherheitsgründen greifen generell weder der TOPSY-Kernel noch User Space-Programme direkt auf angeschlossene Hardwaregeräte wie die Netzwerk-Interfaces oder die serielle Schnittstelle zu. Stattdessen wird ein Gerätetreiber geladen, der die Funktionen des unterstützten Geräts in einem standardisierten Schema anbietet und die gerätespezifischen Eigenheiten des Zugriffs vor dem Aufrufer weitestgehend verbirgt. Das Input/Output-Modul (I/O) stellt dem Betriebssystem und den User Space-Programmen die benötigen Funktionen zur Verfügung, um über System Calls auf diese Gerätetreiber zu greifen zu können.

5.3.1 I/O über die serielle Schnittstelle

Das Input/Output-Subsystem von TOPSY bietet über das Syscall-Interface die Möglichkeit, auf die serielle Schnittstelle zuzugreifen, die die Ausgabe des Bootvorgangs und der Konsole sowie die Entgegennahme von Tastatureingaben übernimmt, und stellt damit die primäre Schnittstelle zur Umgebung zur Verfügung.
Im Rahmen dieser Arbeit beschränkten sich die Änderungen am I/O-Modul auf die Inbetriebnahme der seriellen Schnittstelle, die die RadiSys-Karte über das PCI-Interface mit dem Hostcomputer beziehungsweise den Emulator mit einem Output-File verbindet. Dieser Schritt war für das Gelingen der Arbeit von besonderer Bedeutung, da der Emulator Debugging nicht mit besonderen Einrichtungen unterstützt und wir damit auf den Debug-Output des Startvorgangs angewiesen waren, um die anstehenden Probleme zu identifizieren und komplexere Datenstrukturen ausgeben zu können, um Änderungen zu validieren. Die Anpassung des seriellen Schnittstellentreibers war überraschend aufwendig, da trotz des vergleichbaren Aufbaus der Schnittstelle das Layout des Memory Mapped I/O komplett anders ausgelegt worden war, und damit grössere Anpassungen des Treibers erforderlich wurden. Da sich die Kommunikation via Interrupts nicht stabil implementieren liess, wurde ausserdem im Interesse von Zuverlässigkeit und Termintreue eine Polling-Lösung umgesetzt.

5.4 Memory Management

Dem Memory Management kommt in TOPSY v3 besondere Bedeutung zu, weil es im Unterschied zu den gängigsten Betriebssystemen die Zugriffsberechtigungen auf einen Speicherbereich nicht nur in zwei oder vier Kategorien einteilt, sondern jedem Speicherbereich eine separate Protection Domain zuweist, die eine feinkörnige Zuteilung und Aberkennung von Zugriffsprivilegien auf andere Protection Domains ermöglicht. Die Verwaltung dieser Protection Domains ist vergleichsweise kompliziert, vereinfacht aber auch die Arbeit der Kernel- und User Space-Programmierer und macht die Speicherverwaltung von TOPSY zu einem sehr fortschrittlichen, den etablierten Lösungen überlegenen System. Insbesondere lässt es die Partitionierung des User Space in verschiedene Protection Domains zu, um die laufenden User Space-Programme von unerwünschten Zugriffen und Manipulationsversuchen seitens anderen unprivilegierten Codes zu schützen. Diese Funktionalität wird vom Memory-Subsystem bereitgestellt, während die Anpassung an die spezifische Hardware im PMap-Modul des HAL vorgenommen wird.

5.4.1 Memory Management mittels der MMU

Der IXP2400 verfügt über leistungsfähige Speicherverwaltungseinrichtungen, die über den Coprozessor 15, den System Control Coprocessor, des ARM Cores angesprochen werden. Die Speicherverwaltung unterstützt einen mehrstufigen Translation Lookaside Buffer (TLB) [3] mit Unterstützung für Pages unterschiedlicher Grössen, Kontrolle über das Caching-Verhalten und das Zusammenfassen von mehreren Pages zu Domains mit übereinstimmenden Zugriffsberechtigungen zwecks einfacherer und effizienterer Verwaltung.
Auf dem IXP2400 initialisiert TOPSY die MMU in zwei Schritten: In den ersten ausgeführten Instruktionen wird in start.S die MMU vor-initialisiert, die Endianness korrekt gesetzt, die TLBs

zurückgesetzt und der virtuelle Speicher direkt und ohne Zugriffsbeschränkungen auf den physikalischen Speicher gemappt. Erst in einem zweiten Schritt wird die MMU zu einem späteren Zeitpunkt vollständig initialisiert, die Translation Tables aufgesetzt und der Speicher in Protection Domains mit korrekten Zugriffsrechten unterteilt.
Der Intel IXP2400 unterstützt neben dem SDRAM auch das schnellere SRAM. Um die Einsetzbarkeit von TOPSY in Unterrichtsumgebungen nicht zu gefährend, verzichteten wir jedoch bewusst darauf, performancesteigernde Massnahmen zu treffen und beispielsweise die Translation Tables im SRAM abzulegen, statt der TOPSY-Standardseitengrösse von 4kb auch die hardwareseitig unterstützten 1kb-Pages einzusetzen oder den Versuch zu wagen, das Protection Domain-Modell von TOPSY mit dem Domain-Modell des IXP2400 zusammenzuführen, um von der Hardwarebeschleunigung profitieren zu können. Mehr zu diesen und anderen Erweiterungsmöglichkeiten findet sich im Abschnitt 7.

5.5 Interrupt Handling und Interprozesskommunikation

Der Interrupt Handler ist zuständig für die Verarbeitung aller Interrupts, die im TOPSY-System auftreten können. Interrupts sind asynchrone oder synchrone Signale, die von Hardwarebausteinen oder Softwareinstruktionen ausgelöst werden und den Eintritt eines bestimmten Ereignisses anzeigen [3]. Hardware-Interrupts werden üblicherweise von Hardwaretimern, Elementen des Prozessors und Schnittstellen zu Peripheriegeräten erzeugt, während Softwareinterrupts durch den Code selbst ausgelöst werden; entweder mit dedizierten Instuktionen, durch den Gebrauch von ungültigen oder unzulässigen Instruktionen oder durch den Zugriff auf geschützte Speicherbereiche. Beim Auftreten eines Interrupts speichert der Interrupt Handler den Kontext der unterbrochenen Anwendung, interpretiert die Nachricht und leitet gegebenenfalls eine Message an das zuständige Subsystem weiter, stellt den Kontext der unterbrochenen Anwendung wieder her und gibt die Kontrolle an das unterbrochene Programm zurück. In einigen Fällen wird ausserdem durch das Ausschalten übriger Interrupts verhindert, dass die Abarbeitung des Interrupts nochmals unterbrochen werden kann.
Das Interrupt Handling ist ein zentrales Element in allen gängigen Betriebssystemen, da vom korrekten Verhalten beim Auftreten von Interrupts viele andere Module abhängen: Gerätetreibern zeigt das Auftreten eines Interrupts an, dass neue Daten gesendet oder empfangen werden können, und das Memory Management ist darauf angewiesen, beim Zugriff auf nicht im TLB gecachte Seiten die Umsetzung von virtuellen in physikalische Adressen vornehmen zu können. Auch Syscalls lösen mittels einer spezifischen Instruktion einen Software-Interrupt aus, um die Kontrolle an den Kernel zu übergeben.
Aus architektonischen Gründen sind Interrupt Handler und die Interprozesskommunikation miteinander eng verzahnt und werden deshalb sinnvollerweise gemeinsam behandelt. Der von Interprozesskommunikationssystem bereitgestellte Syscall-Mechanismus ermöglicht den User Space-Programmen begrenzten Zugriff auf Teile des privilegierten Code in den Kernel-Modulen, beispielsweise die Funktionalität bestimmter Gerätetreiber. Ein direkter Aufruf hätte eine Zugriffsverletzung und die sofortige Terminierung des fehlbaren Threads zur Folge, da den User Space-Programmen generell keinerlei Zugriffsrechte auf die Kernel-Speicherbereiche gewährt wird. Anstelle eines direkten Aufrufs legt das User Space-Programm deshalb zuerst die jeweiligen Parameter im Speicher ab, um darauf durch das Auslösen eines Software-Interrupts die Kontrolle an das Betriebssystem abzugeben. Dieses interpretiert die übergebenen Parameter, überprüft die Berechtigung des Aufrufers, und führt den entsprechenden Aufruf aus. Üblicherweise wird daraufhin die Kontrolle in den User Space bzw. an den aufrufenden Thread zurückgegeben.

5.5.1 Interrupt Handling und IPC in TOPSY

Als direkte Folge der leistungsfähigen Speicherverwaltungsarchitektur verwendet TOPSY ein relativ komplexes Interrupt Handling. Im Unterschied zu den herkömmlichen Betriebssystemen, die üblicherweise bloss privilegierten und unprivilegierten Code unterscheiden, lässt das Protection Domain-Modell den Fall zu, dass bei der Abarbeitung eines Interrupts weitere Interrupts ausgelöst werden, beispielsweise durch den Zugriff auf eine fremde Protection Domain innerhalb des Kernels. Dabei ist es von eminenter Wichtigkeit, dass diese Interruptkaskade nicht

zu einer Endlosschleife wird, weil ein nachfolgender Interrupt wieder und wieder durch einen vorhergehenden unterbrochen wird und das System in einen sogenannten *Live Lock* gerät. Die Voraussetzungen für das korrekte Funktionieren des Interrupt Handlers mit Protection Domains wurden bereits mit der Architektur von TOPSY v3 geschaffen, unsere Aufgabe bestand in diesem Zusammenhang darin, den Coprozessor 15 korrekt anzusprechen und die verschiedenen Speicherbereiche mit den nötigen Rechten zu versehen.
Der Umstand, dass das Betriebssystem nicht monolithisch ist, sondern beim Transfer von Daten zwischen den verschiedenen Kernel-Modulen die Grenzen von Protection Domains überschritten werden können, stellt ausserdem zusätzliche Anforderungen an die Interprozesskommunikation (Inter-Process Communication, IPC). Die aus Effizienzgründen gebräuchliche Methode, anstelle grosser Datenmengen innerhalb des Kernels nur die Anfangsadresse der Datenstruktur zu übergeben, wird dadurch obsolet. Stattdessen implementiert der Message Handler zusammen mit dem IPC-Handler im HAL einen Mechanismus, der es zulässt, Daten in dafür vorgesehene Speicherbereiche zu schreiben, und dann statt einer physikalischen Adresse die entsprechende Seite zusammen mit den Zugriffsrechten vom Sender zum Empfänger des Seiteninhalts zu transferieren.

5.6 Thread Management und Scheduling

Als präemtives Multitasking-Betriebssystem verfügt auch TOPSY über ein Thread Management. Der Thread Manager ist dabei zuständig für die Erfassung der Prozessorbelastung durch die einzelnen Prozesse und Threads, die Verwaltung der Tabellen aller zu einem Zeitpunkt existierenden Threads und die Zuteilung von Prozessorzeit, wobei nicht nur Priorität und der gegenwärtige Status des jeweiligen Threads berücksichtigt werden, sondern auch die Priorität der Protection Domain, der der Thread angehört. Für die Generierung einer verlässlichen Zeitbasis verwendet der Thread Manager dafür den Scheduler im HAL. Ausserdem ist der Thread Manager in Zusammenarbeit mit dem Memory Management für die Terminierung von Threads und Prozessen verantwortlich, die in unzulässiger Weise auf den Speicher zuzugreifen versuchen.

5.6.1 Präemptives Multitasking mittels der OS Timer

Damit TOPSY als Betriebssystem mit präemptives Multitasking bezeichnet werden kann, muss es in der Lage sein, auch Prozessen die Kontrolle zu entziehen, die sich unkooperativ verhalten und dem Betriebssystem nicht freiwillig die Kontrolle übertragen. Um dieses Verhalten zu erreichen, wird der Thread Manager durch periodische Interrupts seitens des Schedulers aufgerufen und überprüft so regelmässig, ob der laufende Prozess fortgesetzt werden darf, oder die Kontrolle an einen anderen, höher priorisierten Prozess abgegeben werden muss.
Durch den Umstand, dass sich der Thread Manager weitgehend auf andere Kernel-Module abstützt, und die eingesetzten Verfahren zur Tabellenverwaltung und Priorisierung nicht von hardwarespezifischen Eigenheiten Gebrauch machen, waren im Bereich des Thread Managements nur kleine Anpassungen nötig. Das Sicherstellen des einwandfreien Funktionierens des Interrupt Handlers und des Memory Managements sowie die Implementation der Ansteuerung der Hardware Timer im Scheduling-Modul des HAL machten den Grossteil dieser Teilaufgabe aus.

5.7 Code Cleanup

In seiner heutigen Form ist TOPSY das Resultat harter Arbeit von über einem Dutzend Entwicklern aus dem universitären Umfeld. Ein wesentlicher Teil des investierten Aufwands wurde im Rahmen von Master- und Semesterarbeiten über eine Zeitspanne von mehreren Jahren erbracht, und obwohl die Codequalität über weite Strecken durchaus annehmbar ist, weisen doch viele, besonders die hardwarenahen, Module Artefakte dieses Entwicklungsmodus' auf.
Wir sind allerdings nicht die ersten, die nach einem einheitlicheren Quellcode rufen. Auf der Projektwebsite[1] findet sich eine Konvention, die von der Formatierung von Codeblöcken bis zur Benennung von Variablen unterschiedlichen Typs verschiedene Standards setzt.

[1] http://www.topsy.net

Deshalb war es unser Ziel, als Grundlage für Folgearbeiten und als nützliche Übung für das Berufsleben die “Taktik der verbrannten Erde” nicht weiterzuführen, sondern unseren Code in Übereinstimmung mit den Prinzipien von TOPSY zu schreiben und die vorliegenden Dateien mit maschinellen Mitteln an den verbindlichen Standard anzupassen.

5.7.1 Verwendete Programme und Verfahren

Da sich das ursprüngliche Vorhaben, den vorliegenden Code mittels bestehender Scripts in einem Stream Editor zu parsen und entsprechend den Konventionen im Anhang C neu zu formatieren, wegen fehlender entsprechender Scripts nicht realisieren liess, und die Schaffung eines solchen Toolkits den zeitlichen Rahmen dieser Semesterarbeit gesprengt hätte, wurde diese Teilaufgabe mittels unterschiedlicher, jeweils für einen Teilaspekt geeigneter Programme erfüllt. Dabei fiel die Wahl für die Reformatierung des Codes auf die Entwicklungsumgebung von Microsoft, das Microsoft Visual Studio 2005, weil dieses Produkt ab Werk den geforderten Formatierungsstandard unterstützt. Für das effiziente Refactoring von Makros und die Überprüfung der Initialisierung von Variablen bot sich die Entwicklungsumgebung eclipse an, die mächtige, auf regulären Ausdrücken basierende Suchwerkzeuge anbietet und als Freie Software lizenziert wird. Damit war es mit vergleichsweise kleinem Aufwand möglich, diesen Aspekt der Codequalität nachhaltig zu steigern, ohne die Kompilierbarkeit des Quellcodes zu gefährden.
Ausserdem wurde in die Ausgabe des Startvorgangs ein Verweis auf die GNU General Public Licence [14], der dieser Quellcode untersteht, eingefügt, und die Sammlung der User-Programme wurde um den Befehl `start licence` ergänzt, der diese Lizenz im Terminal ausgibt.

Kapitel 6

Evaluation und Resultate

Dieses Kapitel präsentiert die Resultate unserer Arbeit und beschreibt die Ansätze und Verfahren, die wir verwendeten, um die portierte Version von TOPSY v3 zu evaluieren.

6.1 Methodik

Im Rahmen dieser Aufgabe lässt sich die für die Sicherstellung der Qualität des Resultats geleistete Arbeit in zwei Teilbereiche unterscheiden: Wie bereits im Kapitel 4 angeführt, setzten wir zunächst während der Entwicklung einen iterativen Rapid Prototyping-Ansatz ein: Wir identifizierten zunächst durch schrittweises Ausführen, Register- und Speicherinspektion sowie die Ausgabe von Debug-Output mittels `printf()` den ersten Codebereich, in dem die Ausführung vom erwarteten Verhalten abwich. Dann behoben wir unter Zuhilfenahme von Dokumentation und Linux-Source Code, gelegentlich auch Trial and Error, den Fehler, und verifizierten die korrekte Ausführung des Abschnittes — falls die Prüfung erfolgreich ausfiel, begann die nächste Iteration.
Da wir kein von Grund auf neues Betriebssystem, sondern lediglich eine auf eine neue Plattform der nächsten Hardwaregeneration portierte Version eines bestehenden Produkts erarbeiteten, beschränkte sich die Validierung dabei auf das Sicherstellen des korrekten Ausführung des Hardware Abstraction Layers und dessen einwandfreie Zusammenarbeit mit den Kernelmodulen. Wir verzichteten deshalb darauf, aktiv nach Fehlern im portablen Teilen des Kernels zu suchen, und konzentrierten uns primär auf die Abarbeitung des Startvorgangs und die Ausführung der in TOPSY v3 enthaltenen User Space-Programme. Dieser Ansatz vereinfachte die Aufgabe der Validierung erheblich, und liess trotz des stark reduzierten Aufwands verlässliche Aussagen über die Codequalität zu.
Vor diesem Hintergrund wurde nach dem Abschluss der Codierarbeiten in einem zweiten Schritt die Durchführung einer relativ einfachen qualitativen und quantitativen Evaluation vorgesehen. Dafür wurden im Emulator die Start- und Endzeiten von vier für das Funktionieren des Betriebssystems zentralen Vorgängen gemessen, und die im folgenden abgedruckte Ausgabe der `start hello`- und `start demo`-Befehle erzeugt.

6.2 Resultate der Evaluation

Seit dem Abschluss dieser Semesterarbeit laufen der TOPSY v3-Kernel und die User Space-Programme wie in der Aufgabenstellung spezifiziert gleichermassen stabil auf dem IXP2400-Emulator und Hardwareplattform. Die Initialisierung der Hardware und aller Module von TOPSY verläuft einwandfrei, und die grosse Mehrheit der User Space-Programme werden anstandslos und zuverlässig ausgeführt.
In der Tabelle 6.1 finden sich die Resultate der quantitativen Evaluation, die in der Emulatorumgebung gemessen wurden.
Stellvertretend für das laufende Betriebssystem wird zur qualitativen Evaluation ausserdem die Ausgabe der `start hello`- und `start demo`-Befehle in der TOPSY v3-Shell wiedergegeben:

Messgrösse	Start (hex)	Ende (hex)	Cycles	benötigte Zeit
Message Passing von Kernel- zu Kernel-Protection Domain	0x7dda58	0x7ddf7d	1317	2.195μs
Message Passing von User- zu User-Protection Domain	0x5cd945b	0x5cea4c1	69734	116.2μs
Laden des user.bin-Images und Starten der Shell (mit `printf`)	0x8a4393	0x9176d4	471873	0.786ms
Laden des user.bin-Images und Starten der Shell (ohne `printf`)	0x8144f2	0x86fa9c	374186	0.623ms

Tabelle 6.1: Resultate der quantitativen Evaluation

```
Topsy Shell 3.0 (c) 1998, ETH Zurich, TIK
Type 'start licence' to print the licence terms governing your use
of this product.

($Id: Shell.h 2911 2007-03-07 13:27:12Z rnatau $)

> start hello
COMMAND: start hello

threadBuild: stack base 0xcfdfe000 mappedBase 0x60000000, stackStart
0xcfefdffc, mappedStart 0x600ffffc
Exception: cause 4, 0x5 0x600fffe0 0 [PC 0x0c208e5c]
vm_region_free: free region form 0x60000000 - 0x600fffff
Exception: cause 4, 0x5 0xcfefdfac 0 [PC 0x0d002de8]
Hello World

vm_region_free: free region form 0xcfdfe000 - 0xcfefdfff
> start demo
COMMAND: start demo

threadBuild: stack base 0xcfdfe000 mappedBase 0x60000000, stackStart
0xcfefdffc, mappedStart 0x600ffffc
Exception: cause 4, 0x5 0x600fffe0 0 [PC 0x0c208e5c]
vm_region_free: free region form 0x60000000 - 0x600fffff
parent started

threadBuild: stack base 0xcfcfd000 mappedBase 0x60000000, stackStart
0xcfdfcffc, mappedStart 0x600ffffc
Exception: cause 4, 0x5 0x600fffe0 0 [PC 0x0c208e5c]
vm_region_free: free region form 0x60000000 - 0x600fffff
Exception: cause 4, 0x5 0xcfdfcfb4 0 [PC 0x0d002f38]
parent running

child started

parent received message from child

child running

vm_region_free: free region form 0xcfcfd000 - 0xcfdfcfff
parent continuing

exit of child

vm_region_free: free region form 0xcfdfe000 - 0xcfefdfff
```

6.3 Beurteilung

Die Validierung eines Basisbetriebssystems ist eine notorisch schwierige Aufgabe. Bedingt durch den im Vergleich zu anderen Softwareprojekten typischerweise grossen Funktionsumfang, die Nebenläufigkeit der verschiedenen Prozesse und die Eigenheit, dass die weite Teile der Software direkt mit der Hardware kommunizieren, ist es grundsätzlich sehr schwierig, die Stabilität und das erwartungsgemässe Funktionieren eines Betriebssystems in allen denkbaren Situationen zu garantieren.
In unserem Fall kommt erschwerend hinzu, dass wir den grössten Teil der Entwicklungs- und Validierungsarbeit in der Emulatorumgebung vornahmen, und deshalb die Zuverlässigkeit der Resultate stark vom übereinstimmenden Verhalten von Hardware und Emulator abhängt.
Da wir allerdings aufgrund des verfolgten Entwicklungsansatzes einen grossen Teil der Validierung des Codes während der Entwicklungsarbeit erbracht haben, gehen wir trotzdem davon aus, dass unser Resultat die Ansprüche für im Rahmen von Semesterarbeiten entwickelte Software weitestgehend erfüllt. Der Umstand, dass sich bei der Inbetriebnahme der Hardware nach der Fertigstellung der Emulatorversion kaum Unterschiede im Ausführungsverhalten auftraten, entkräftet die Vorbehalte gegenüber der Entwicklung in einer solchen Umgebung wenigstens teilweise. Und während die Portierung von der IXP1200-Plattform auf den IXP2400-Emulator den Grossteil der investierten Zeit in Anspruch nahm, benötigte die Inbetriebnahme der Hardware nur einige Arbeitstage, woran die serielle Schnittstelle den grössten Anteil hatte. Diese Gründe rechtfertigen die Annahme, dass sich Aussagen über die während der Entwicklung erreichte Stabilität in der Emulatorumgebung grundsätzlich auf die Hardware übertragen lassen.
Für den Fall, dass TOPSY v3 den produktiven Betrieb aufnehmen soll, wäre es allerdings ratsam, die Einheit von Betriebssystem und entwickelter User Space-Software nochmals eingehend und sorgfältig auf ihre Zuverlässigkeit im Dauereinsatz und unter hoher Belastung zu überprüfen. Dieser Aspekt erhielt als Konsequenz der verfolgten Entwicklungsmethodik ein relativ geringes Gewicht.

Kapitel 7

Ansätze zur Weiterentwicklung und Problembehebung

In diesem Abschnitt möchten wir einen Ausblick auf zukünftige Entwicklungsmöglichkeiten geben. Durch den beschränkten Umfang einer Semesterarbeit war beispielsweise nur vorgesehen, den Core des IXP2400 einzusetzen. Die in der Summe leistungsfähigeren Packetprozessoren bleiben beim Einsatz von TOPSY auf dem IXP2400 ungenutzt. Es wäre eine sinnvolle Erweiterung, Module wie den Netzwerkstack teilweise auszulagern. Wesentlich anspruchsvoller, aber auch von höherem akademischem Wert, wäre die Entwicklung eines Frameworks, das die Ansteuerung der Packetprozessoren aus dem User Space erlaubt, oder sogar eine Just-In-Time-Kompilierung von ARM-Instruktionen in ein Programm vornimmt, dessen repetitive Abschnitte auf den Packetprozessoren ausgeführt werden.
Neben den Paketprozessoren bietet die IXP2400-Hardware noch eine Reihe weiterer attraktiver Features, die von TOPSY in seiner heutigen Form nicht verwendet werden: Es wäre beispielsweise relativ einfach, einen Treiber für die Hardware-Hashing Unit zu schreiben, um ohne Belastung des Cores 48-, 64- und 128-bit-SHAc-Hashes zu errechnen. Gleichermassen könnte man die drei heute unbenutzten Timer in Betrieb nehmen und den User Space-Programmen anbieten oder für das Betriebssystem einsetzen.
Ein besonderes Merkmal von TOPSY v3 ist die ausgefeilte Speicherverwaltung. Deren Features wurden zwar implementiert, machen aber von der hardwareseitigen Unterstützung nicht vollen Gebrauch. So unterstützt der Prozessor neben dem TOPSY-Standard von 4kb auch Seitengrössen von 1kb und 64kb, und bei 4kb- und 64kb-Seiten auch Subpages von einem Viertel der Grösse mit separaten Zugriffsberechtigungen. Darüber hinaus ist auch das Zusammenfassen von mehreren Seiten zu sogenannten Domains möglich, deren Rechte dann mit einem einzelnen Zugriff verändert werden können. Der Einsatz einiger dieser Features im Memory Management wäre eine naheliegende, sinnvolle Erweiterung.
Im Kapitel 2 gingen wir auf die Grundsätze und Zielsetzungen der Entwicklung von TOPSY ein. Die dort beschriebene Ausrichtung auf den universitären Unterricht ist sowohl ein Vor- als auch ein Nachteil: Die Maximen der Portabilität und der Verständlichkeit des Codes beschränken die Ausrichtung auf bestimmte Plattform und die Komplexität der Programmierung. In der Praxis erschweren sie, dass TOPSY auf einer spezifischen Plattform jemals seine volle Leistungsfähigkeit unter Beweis stellen kann, da diesbezügliche Optimierungen den Entwicklungsprinzipien zuwiderlaufen. Falls allerdings ein praktischer Einsatz, beispielsweise als Node OS auf einem Embedded Device, realisiert werden soll, dürfte es gerade dank diesen Prinzipien sowohl bestehenden Entwicklern wie Neuzugängen der TOPSY-Entwicklergemeinschaft leicht fallen, die performancerelevanten Teilsysteme zu identifizieren, von den Eigenschaften der eingesetzten Hardware Gebrauch zu machen, und so die Performance des Betriebssystems deutlich zu steigern.

Kapitel 8

Zusammenfassung und Fazit

In diesem Bericht wurde die Portierung des freien Betriebssystems TOPSY v3 von der IXP1200- auf die IXP2400-Plattform präsentiert. Er beschrieb die spezifischen Probleme, die sich bei der Betriebssystemprogrammierung auf diesem Netzwerkprozessor und der Überwindung eines Generationsschrittes innerhalb derselben Serie ergaben, und stellte einen einfachen, aber effizienten Ansatz zur Bewältigung solcher Aufgaben vor. In einer Einführung in die hardwarenahen Module von TOPSY v3 wurden die Funktionen der Module von TOPSY vorgestellt, und die am Code vorgenommenen Änderungen umrissen, danach wurden die Resultate dieser Arbeit kritisch gewürdigt. Ausserdem versuchten wir, zusammen mit den Angaben in den Anhängen alle nötigen Informationen zu geben, um den interessierten Leser die vorgenommenen Arbeitsschritte nachvollziehen oder die vorgestellen Erweiterungs- und Weiterentwicklungsmöglichkeiten realisieren zu lassen.

8.1 Erreichte Ziele

Im Laufe dieser Arbeit wurde TOPSY v3 zuerst in der Emulatorumgebung zur Lauffähigkeit gebracht, und schliesslich auch auf der Hardware in Betrieb genommen. Das Hardware Abstraction Layer für den IXP2400 wurde mit den in Kapitel 7 beschriebenen Einschränkungen implementiert und evaluiert. Diese Version findet sich als Revision 2919 im Subversion-Repository der Projektwebsite.
Im Anschluss daran wurde der Code Cleanup durchgeführt, der Quelltext also nach den Konventionen im Anhang C refactored und neu formatiert. Der überarbeitete Code findet sich als Revision 2937 im Subversion-Repository. Unter Revisionsnummer 2938 findet sich die endgültigen Abgabe, die auch die Schlussversion dieses Berichts enthält.

8.2 Fazit

“Die Betriebssystemprogrammierung ist ein interessantes und anspruchsvolles Betätigunsfeld” – Mit diesen Worten leiteten diesen Bericht ein, und viel treffender könnte man in einem Satz dieses Fachgebiet schwerlich charakterisieren. Es ist ohne Zweifel ein aussergewöhnliches Erlebnis, sich an der Weiterentwicklung eines Betriebssystems beteiligen zu dürfen – wir haben im Rahmen dieser Arbeit ein Betriebssystem geschaffen, das die Entwicklung leistungsfähiger Applikationen auf der neusten Generation der Intel-Netzwerkprozessoren erlaubt, brachten das TOPSY-Projekt einen weiteren Schritt voran, und konnten für unseren weiteren akademischen Weg wertvolle praktische Erfahrung im eigentlichen Handwerk der Programmierung, aber auch im strukturierten Angehen eines Projekts von diesem Umfang sammeln.
Nebst diesem Dokument umfasst die Dokumentation dieser Arbeit auch eine CD-ROM, die unter anderem die eingesetzten Hilfsmittel, den gesamten Source Code, die erwähnten Shell-Scripts und den Quelltext dieses Berichts in digitaler Form enthält. Das Lizenzmodell von TOPSY verpflichtet uns, den Quelltext allen Interessenten zur Verfügung zu stellen. Wenden Sie sich deshalb für die Bereitstellung einer Kopie bitte an bader@topsy.net oder natau@topsy.net.

Anhang A

Toolchain

Dieser Anhang enthält die nötigen Befehle zur Installation der von uns verwendeten Toolchain und gibt das zur Arbeit gehörende Script wieder, das diese Aufgabe automatisiert, TOPSY v3 kompiliert und im Emulator startet.
Für die Entwicklung verwendeten eine aus folgenden Hilfsmitteln bestehende Toolchain:

- GNU C Compiler Version 4.1.1
- Newlib C-Bibliotheken Version 1.14.0
- GNU Binutils Linker und Assembler Version 2.17

Das folgende Script funktioniert unter der Voraussetzung, dass

- die Dateien `gcc-4.1.1.tar.bz2`, `newlib-1.14.0.tar.gz` und `binutils-2.17.tar.bz2` im Arbeitsverzeichnis vorliegen, und
- der Compiler für die Host-Plattform bereits installiert ist und die dafür nötigen Libraries eingerichtet sind. Unter Debian Linux auf der x86-Plattform sind das die Pakete `make`, `bin86` und `build-essential`.

```
#!/bin/bash

# Entpacken der Quelldateien
tar -jxf gcc-4.1.1.tar.bz2 &
tar -zxf newlib-1.14.0.tar.gz&
tar -jxf binutils-2.17.tar.bz2

# Konfiguration und Installation von Binutils
cd binutils-2.17
./configure --target=arm-elf --prefix=‘pwd‘/../toolchain \
  --enable-interwork --enable-multilib --with-float=soft
make all install
cd ..
export PATH="$PATH:‘pwd‘/toolchain/bin"

# Konfiguration von GCC
cd gcc-4.1.1
./configure --target=arm-elf --prefix=‘pwd‘/../toolchain \
  --enable-interwork --enable-multilib --with-float=soft \
  --enable-languages="c,c++" --with-newlib               \
  --with-headers=../newlib-1.14.0/newlib/libc/include
make all-gcc install-gcc

# Konfiguration und Installation von Newlib
cd ../newlib-1.14.0
./configure --target=arm-elf --prefix=‘pwd‘/../toolchain \
```

```
  --enable-interwork --enable-multilib --with-float=soft
make all install

# Installation von GCC
cd ../gcc-4.1.1
make all install
cd ..

# Konfiguration und Installation des Emulators
./emu.sh

# Compilation von TOPSY v3
cd TopsyIXP2400
./make clean
./make ixp2400

# Starten des Emulators
../IXP-Emulator/src/arch/simarm/simarm -c simarm.cfg
```

Anhang B

Binary Patcher

Zur Erleichterung der Aufgabe des Loader-Moduls des Memory Managements enthalten die Image-Files in einem vorbereiteten Header die Adressen der .text-, .data-, .bss- und .optional-Segmente [12]. Diese Adressen werden für die Initialisierung der Protection Domains benötigt, die der Loader beim Laden der Images anlegt.
Aufgrund der von uns verwendeten, aktuelleren Compiler-Version traten beim Einsatz des ursprünglichen Binary Patchers Kompatibilitätsprobleme auf, die die folgenden Anpassungen erforderlich machten:

- Die von uns eingesetzte Compilerversion verwendete neue Bezeichnungen für die Segmente des Kompilats. Diese Änderung machte auch eine Anpassung des Parsers im Binary Patcher notwendig.
- Während die ursprüngliche Compilerversion nur ein .data-Segment anlegte, unterteilte ihre Nachfolgerin das .data-Segment in zwei einzelne Teile. Der Parsing-Mechanismus konnte diee nicht verarbeiten und musste erst dafür vorbereitet werden, diese beiden Teile zu einem einzelnen zusammenfassen zu können.
- Da die von uns eingesetzte Karte ihren Speicher bei einem Reset nicht löschte, der Compiler jedoch aus Performancegründen einen mit Nullen initialisierten Speicher voraussetzt, war es nötig, an die erzeugten Images einen mit Nullen gefüllten Speicherbereich mit der Grösse des .bss-Segments anzuhängen. So konnte sichergestellt werden, dass alte Daten in den .bss-Bereichen nach einem Reset der Karte zuverlässig initialisiert wurden. Der ursprüngliche Binary Patcher verfügte über keine solche Funktionalität.

Anhang C

Konventionen des Code Cleanup

Initialisierung der Variablen
Jede Variable muss bei der Definition initialisiert werden.
Beispiel:

```
int foo(int x)
{
  int y = 0;  // <--- Diese Initialisierung ist obligatorisch
  ...
  y = -1;
  ...
  return y;
}
```

Einrückung
Der TOPSY-Source Code verwendet die sogenannte *Tab Expansion*. Ein Tabulator entspricht zwei Leerschlägen.

Klammern

- *Öffnende Klammern*
  Öffnende Klammern stehen immer auf einer eigenen Zeile, auf der Höhe der Einrückung der vorhergehenden Zeile.
- *Schliessende Klammern*
  Schliessende Klammern stehen immer auf einer eigenen Zeile, auf der Höhe der Einrückung der zugehörigen öffnenden Klammer.
- *Einrückung*
  Die Einrückung folgt dem *cindent*-Modus des Editors vim. Die Einrückungstiefe entspricht dabei einem Tabulator oder zwei Leerschlägen.

Funktionsdefinitionen
Wir verzichten auf den Kernighan & Ritchie-Stil und führen alle Statements im Funktionsheader auf.
Beispiel:

```
int main(int argc, char **argv, char **envp)
{
  if (argc > 0)
    return 1;
  else
    return 0;
  return 0;
}
```

Typedefs und Structs

- Der Name eines Typedefs endet immer mit `_t`.
- Der Name eines Structs endet immer mit `_s`.

Beispiel:

```
typedef struct vm_map_s
{
  ...
} vm_map_t;

vm_map_t vm_map;

struct mm_map_s
{
  ...
} mm_map;
```

Einfache Variablentypen
In TOPSY werden ausschliesslich die folgenden einfachen Variablentypen eingesetzt. Falls der Speicherbedarf eine Rolle spielt, wird der Bit Counter ans Ende gestellt.
Die folgende Liste enthält alle zulässigen einfachen Variablentypen:

- `uint8`
  8 bit vorzeichenloser Integer
- `uint16`
  16 bit vorzeichenloser Integer
- `uint32`
  32 bit vorzeichenloser Integer
- `uint64`
  64 bit vorzeichenloser Integer
- `int8`
  8 bit vorzeichenbehafteter Integer
- `int16`
  16 bit vorzeichenbehafteter Integer
- `int32`
  32 bit vorzeichenbehafteter Integer
- `int64`
  64 bit vorzeichenbehafteter Integer
- `size_t`
  Variablentyp für die Aufnahme der Grösse von Speicherbereichen
- `offset_t`
  Variablentyp für die Aufnahme von Offsets
- `address_t`
  Variablentyp für die Aufnahme von Speicheradressen
- `char`
  gewöhnlicher vorzeichenbehafteter Char
- `short`
  gewöhnlicher vorzeichenbehafteter Short Integer
- `int`
  gewöhnlicher vorzeichenbehafteter Integer

- `long`
  gewöhnlicher vorzeichenbehafteter Long Integer
- `long long`
  gewöhnlicher vorzeichenbehafteter Long Long Integer
- `unsigned char`
  gewöhnlicher vorzeichenloser Char
- `unsigned short`
  gewöhnlicher vorzeichenloser Short Integer
- `unsigned int`
  gewöhnlicher vorzeichenloser Integer
- `unsigned long`
  gewöhnlicher vorzeichenloser Long Integer
- `unsigned long long`
  gewöhnlicher vorzeichenloser Long Long Integer
- `boolean`
  binärer Boole'scher Variablentyp

Makros und #defines

- verwenden ausschliesslich Grossbuchstaben. Ausnahmen sind nur für üblicherweise kleingeschriebene Registernamen und Makros für einzelne Assemblerinstruktionen erlaubt.
- einfache Makros müssen in öffnende und schliessende Klammern gesetzt werden
- bei der Verwendung von Inline-Funktionen müssen diese mit

  ```
  do { ... } while(0);
  ```

  eingefasst werden

Anhang D

Timetable

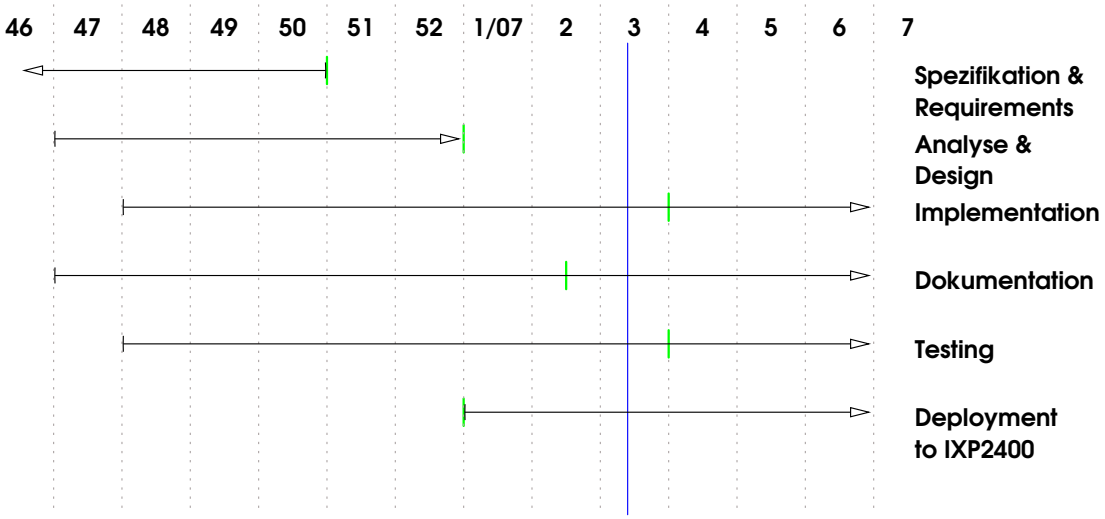

Abbildung D.1: Zeitplan

Anhang E

Original Problem

E.1 Einführung

Netzwerkprozessoren [16, 15] (NPs) werden in einer Chip-Multicore Prozessor Architektur mit einem sog. Control Prozessor (CP) und mehreren Paket Prozessoren (PPs) gefertigt. In Netzwerkinterfacekarten (NICs) eingebettet, sind sie durch die Parallelschaltung mehrerer PPs in der Lagen, Netzwerkverkehr mit hohen Durchsatzraten zu bearbeiten.
Topsy v3 [12] – a Teachable Operating System [11] – wurde an der ETH Zürich entwickelt und wird an verschiedenen Universitäten und Fachhochschulen im Unterricht eingesetzt. Einer Schichtenarchitektur folgend wurde Topsy v3 als portables Betriebssystem konzipiert, das für unterschiedliche Plattformen zur Verfügung steht. Unter anderen wurde Topsy v3 in zwei vorangehenden Arbeiten [8, 12] auf die Evaluation Plattform des Intel IXP1200 NPs [9] portiert.

E.2 Aufgaben:

In dieser Doppelsemesterarbeit von Matthias Bader und Robert Natau soll Topsy v3 auf die Nachfolgeversion des IXP1200, den Intel IXP2400, portiert werden. Diese Portierung umfasst die Anpassungen des Hardware Abstraktionsschicht (HAL, Hardware Abstraction Layer) und, wo benötigt, die Implementierung und Anpassung geeigneter Mechanismen in der portablen Kernelschicht von Topsy v3.

E.3 Vorgehen

Die Portierung von Topsy v3 erfolgt in zwei Phasen. In der ersten Phasen wird Topsy v3 auf den binär kompatiblen Emulator der IXP2400 Plattform portiert, in der zweiten wird die Portierung so erweitert, dass Topsy v3 auf der realen Radisys ENP2611 [7] Entwicklungsplattform des Intel IXP2400 ausgeführt werden kann.
Zur Erreichung dieses Ziels wird die Arbeit nach folgendem Vorgehen durchgeführt:

- Richten Sie sich eine Entwicklungsumgebung (GNU Tools) unter Linux ein.
- Machen Sie sich vertraut mit den Unterlagen (Dokumentation und Source Code) zum bestehenden Topsy v3 für den Intel IXP1200.
- Erstellen Sie einen Zeitplan, in welchem Sie die von Ihnen zu erreichenden Meilensteine Ihrer Arbeit identifizieren.
- Arbeiten Sie sich in die Dokumentation des Intel IXP2400s [5, 4] und Radisys ENP2611s ein.
- Analysieren Sie verschiedene Protokolle und definieren Sie Funktionsmodule, die in verschiedenen Protokollen benötigt werden.
- Identifizieren Sie die anzupassenden Funktionen und Parameter in Topsy v3, die für die Portierung angepasst werden müssen.
- Implementieren Sie diese für den Intel IXP2400.
- Beachten Sie insbesondere die Problematik des Startvorgangs des Radisys ENP2611s.

optional
- Entwickeln Sie die benötigten Mechanismen, um die Paket Prozessoren des IXP2400s einsetzen zu können.
- Zeigen Sie die erfolgreiche Implementierung auf dem IXP2400 durch eine Ausführung der Demo-Applikationen unter Topsy v3.

Auf eine klare und ausführliche Dokumentation wird besonders Wert gelegt. Es wird empfohlen, diese laufend nachzuführen und insbesondere die entwickelten Konzepte und untersuchten Varianten vor dem definitiven Variantenentscheid ausführlich schriftlich festzuhalten.

E.4 Organisatorische Hinweise

- Am Ende der zweiten Woche ist ein Zeitplan für den Ablauf der Arbeit vorzulegen und mit dem Betreuer abzustimmen.
- Mit dem Betreuer sind regelmässige, zumindest wöchentliche Sitzungen zu vereinbaren. In diesen Sitzungen sollen die Studenten mündlich über den Fortgang der Arbeit und die Einhaltung des Zeitplanes berichten und anstehende Probleme diskutieren. Die Sitzungen können mittels geeigneten Telekommunikationsmitteln abgehalten werden, falls möglich.
- Am Ende des ersten Monates muss eine Vorabversion des Inhaltsverzeichnis zur Dokumentation dem Betreuer abgegeben und mit diesem besprochen werden.
- Nach der Hälfte der Arbeitsdauer soll ein kurzer mündlicher Zwischenbericht abgegeben werden, der über den Stand der Arbeit Auskunft gibt. Dieser Zwischenbericht besteht aus einer **viertelstündigen**, mündlichen Darlegung der bisherigen Schritte und des weiteren Vorgehens gegenüber Professor Plattner.
- Am Schluss der Arbeit muss eine Präsentation von **20 Minuten** im Fachgruppen- oder Institutsrahmen gegeben werden. Anschliessend an die Schlusspräsentation soll die Arbeit Interessierten praktisch vorgeführt werden.
- Ein einheitlicher Coding Style muss eingehalten werden.
- Bereits vorhandene Software kann übernommen und gegebenenfalls angepasst werden; Quellen müssen korrekt zitiert werden.
- Die Dokumentation muss mit dem Satzsystem LaTeX Graphiken müssen mit Open Source Werkzeugen erstellt werden, während für die Präsentation auch Staroffice oder Microsoft PowerPoint verwendet werden kann.

- Quellcode der Arbeit muss regelmässig (mind. täglich) auf dem Topsy subversion Server gesichert werden (`https://svn.topsy.net`).
- Es ist ein mit Bindespiralen (am TIK vorhanden) gebundener Schlussbericht über die geleisteten Arbeit abzuliefern (4 Exemplare). Dieser Bericht besteht aus einer Zusammenfassung, einer Einleitung, einer Analyse von verwandten und verwendeten Arbeiten, sowie einer vollständigen Beschreibung der Konfiguration von den eingesetzten Programmen. Der Bericht ist in Deutsch oder Englisch zu halten. Die Zusammenfassung muss in Deutsch und Englisch verfasst werden.
- Die Arbeit muss auf CDROM archiviert abgegeben werden. Stellen Sie sicher, dass alle Programme sowie die Dokumentation und die Präsentationen sowohl in der lauffähigen, resp. druckbaren Version als auch im Quellformat vorhanden, lesbar und verwendbar sind. Mit Hilfe der abgegebenen Dokumentation muss der entwickelte Code zu einem ausführbaren Programm erneut übersetzt und eingesetzt werden können.
- Diese Arbeit steht unter der GNU General Public License [14] (GNU GPL) v2.
- Für die Arbeit werden Informationen eingesetzt, welche nur durch das Unterzeichnen eines NDA (Non-Disclosure Agreement) zwischen Intel Corp. und der ETH Zürich erhalten werden konnten. Alle Informationen, die unter das NDA fallen, dürfen nur an Dritte weitergegeben werden, wenn eine schriftliche Einwilligung von Intel Corp. vorliegt.
- Diese Arbeit wird als Diplomarbeit an der ETH Zürich durchgeführt. Es gelten die Bestimmungen hinsichtlich Kopier- und Verwertungsrechte der ETH Zürich.

Literaturverzeichnis

[1] Hampa Hug, *PCE - Der PC-Emulator*,
TIK, ETH Zürich, 2001.

[2] ARM, INC., *ARM Architecture Reference Manual*,
www.arm.com, 2000.

[3] D. Patterson et al., *Computer Organization and Design: The Hardware/Software Interface*,
Morgan Kaufmann Publishers, 1994.

[4] Intel, *IXP2400 Product Brief*,
www.intel.com, 2004.

[5] Intel Corp., *IXP2400 and IXP2800 Network Processor Programmer's Reference Manual*,
Intel Corp., 2004.

[6] Intel Corp., *IXP2400 Network Processor Hardware Reference Manual*,
Intel Corp., 2003.

[7] RadiSys Corporation, *RadiSys ENP-2611 Datasheet*,
www.radisys.com, 2006.

[8] Silvio Dragone, David Reist, *TopsyIXP – Porting TOPSY to the IXP1200*,
Computer Engineering and Networks Laboratory (TIK), 2001.

[9] Intel Corp., *Intel IXP1200 network processor – datasheet*,
http://www.intel.com, 2000.

[10] Kent Beck, *Extreme Programming – das Manifest. Die revolutionäre Methode für Softwareentwicklung in kleinen Teams*,
Addison Wesley, 2000.

[11] G. Fankhauser, C. Conrad, E. Zitzler, and B. Plattner, *Topsy – A Teachable Operating System*,
Computer Engineering and Networks Laboratory (TIK), 1997.

[12] C. Jeker and B. Lutz. Memory, *IO and Process/Thread Management for Topsy v3*,
Computer Engineering and Networks Laboratory (TIK), 2002.

[13] L. Ruf, C. Jeker, B. Lutz, B. Plattner, *Topsy v3: A NodeOS For Network Processors*,
Proceedings of the 2nd International Workshop on Active Network Technologies and Applications (ANTA 2003), Osaka, Japan, May, 2003.

[14] GNU, *GNU General Public License*,
http://www.gnu.org, 2005.

[15] IBM Corp. *Datasheet IBM NP4GS3*,
http://www.ibm.com, 2004.

[16] Intel Corp., *Intel IXP2xxx hardware reference manual*,
http://www.intel.com, 2003.

[17] Intel Corp., *Intel IXP2400/IXP2800 Network Processor: Programmer's Reference Manual*,
http://www.intel.com, 2004.

[18] MIPS Corp., *MIPS32 24Kc Processor Core Datasheet*, http://www.mips.com, 2002.

[19] ARM, INC., *ARM Technical Reference Manual*, http://www.arm.com, 2001.

[20] Motorola, *68000 Family Programmer's Reference Manual*, http://www.freescale.com, 1992.

[21] Intel Corp., *IA-32 Architectures Software Developer's Manual*, http://www.intel.com, 2006.