平成17年度 修士論文

ネットワークを利用した組込み計算機向け開発用資源拡張システム

電気通信大学 大学院情報システム学研究科

情報システム設計学専攻

0450032 明神 智之

指導教員 多田 好克 教授
大森 匡 助教授
田野 俊一 教授

提出日 平成18年2月23日

目 次

図 目 次

表 目 次

第 1 章

背景と目的

1.1 背景

近年、携帯電話や情報家電の普及に伴って、組込み計算機の利用は業務用製品にとどまらず大きく拡大してきている。組込み計算機とは特定の用途のために作られた小型の計算機であり、一般的に省スペース、省コスト、省電力といった特徴を持つ。今後、組込み計算機を用いたシステムの利用は益々大きくなるものと予想される。

組込み計算機は上記のような特徴を持つ代償として、資源や拡張性が乏しいという欠点を持つ。資源とは、メインメモリ、CPU、HDD などの補助記憶装置、キーボード、ビデオ、マウス等である。また拡張性とは、上に述べた資源を追加するための機構である。

資源と拡張性に乏しい組込み計算機では、キーボードやビデオ等の入出力デバイスがないため、システムの開発、保守を効率的に行えない。これらを解決するために、単純にハードウェアを増設する手段があるが、組込み計算機の持つ省スペース性を大きく失うことになる。また、情報家電などの組込み計算機は、その性質上大量生産される。このとき、わずかでもハードウェアが省略可能であればコストの大幅な低減が見込まれるため、開発時、運用時においてのみ必要なハードウェアを搭載することはコストの観点からも好ましくない。

ハードウェアを増設する以外に資源を追加する手法として、ネットワークを介し

て外部の計算機から資源を借りてくる手法があるが、既存手法では組込み計算機に対してハードウェア要求が高いなどの理由により、必ずしも適当とはいえない。

上記のことから、組込み計算機に適した資源拡張システムが必要とされる。

1.2 目的

本研究では、ネットワークを利用した組込み計算機向け開発資源拡張システムを提案する。本システムは、既存手法と比較して、組込み計算機に対して最適なものを提供することを目的とする。

本システムは非常にコンパクトに実装され、組込み計算機に対してハードウェア要求が低いものとなる。本システムを用いることで、組込み計算機からネットワークを介して外部の資源を簡単に利用することが可能となる。本システムを用いることで、組込み計算機上での開発、保守が容易となる。ハードウェアには一切変更を行わないので、組込み計算機の省スペース性を失うことがなく、追加のハードウェアコストもかからない。

1.3 構成

以下に、本論文の構成を示す。2 章で組込み計算機の特徴を示し、本研究で扱う組込み計算機の範囲を明確にして、組込み計算機での資源拡張手法の必要性を述べる。3 章では既存手法とその問題点を説明する。4 章で本研究で構築するシステムの概要を述べ、5 章で本システムの実装について説明する。6 章では本システムの評価を行い、既存手法と比較し、本システムの組込み計算機に対する妥当性を検証する。7 章で関連研究と本研究について述べ、8 章で本研究の今後の課題について議論する。最後に 9 章で全体のまとめを述べる。

第 2 章

組込み計算機の特徴

本章では、まず、一般的な計算機と組込み計算機の違いを述べ外部の資源を利用する必要性について述べる。

2.1 一般的な計算機と組込み計算機の違い

組込み計算機は特定の用途のために設計されているため、一般的な計算機とは求められる点が異なる。ここでは比較のために組込み計算機の一例として BRAINS 社の超小型マイクロサーバ (図 2.1) を挙げる。組込み計算機と一般的な計算機との違いを表 2.1 に示す。

2.2 組込み計算機の要求点

組込み計算機は特定用途の要求に応えるように作られている。組込み計算機の要求点は大きく分けて三つあり、省スペース、省電力、低コストである。ここではそれら三つの要求点について述べる。

省スペース

組込み計算機の適応範囲は多岐にわたる。設置スペースが限られている場合は、組込み計算機を省スペースに設計する必要がある。図 2.1 が示すように、組込み計算機では非常に小さなサイズを実現できる。

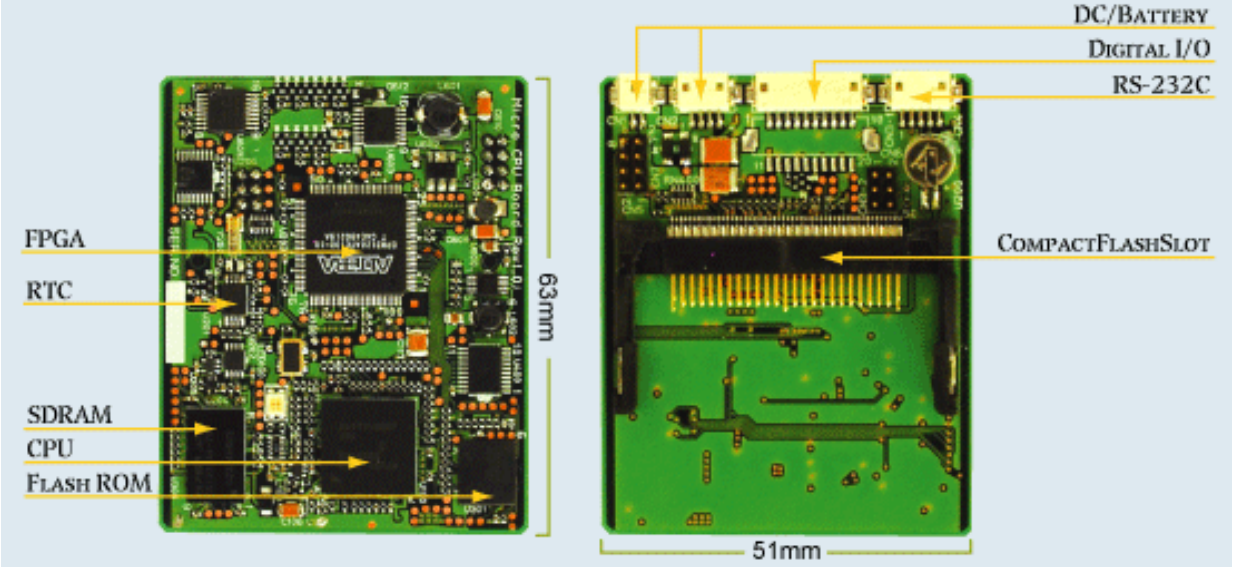

図 2.1: 組込み計算機の実例 (BRAINS 社の製品カタログから引用)

表 2.1: 一般的な計算機と組込み計算機の比較

	一般的な計算機	組込み計算機
CPU	○ (数 GHz)	△ (数十 MHz ～ 数百 MHz)
メインメモリ	○ (数百 MByte ～ 数 GByte)	△ (数十 MByte)
補助記憶装置	○ (HDD で数十 GByte)	△ (FlashROM で数十 MByte)
入出力機能	○	×
拡張性	○	×
サイズ	大	小
イーサネット	○	○

省電力

情報家電など家庭内で使用される組込み計算機の場合、大きな電力を消費することは、ランニングコスト上好ましくない。また、携帯電話などバッテリーで駆動する場合、稼働時間を長くするために省電力にする必要がある。

低コスト

組込み計算機は特定用途に向けて大量に生産される。このとき、必要のない機能を搭載しないことで、使用されるハードウェアやソフトウェアを削減でき、大幅なコストダウンにつながる。そのため、個々の組込み計算機は低コストにする必要がある。

2.3 組込み計算機の用途と種類

組込み計算機は前述した点を要求する分野で利用される。その主な利用例としては、家庭用電化製品、自動車制御、ロボット制御、監視・観測システムなどが挙げられる。これらはそれぞれ用途が異なるため、使用される組込み計算機の性能が異なる。図 2.2 に様々な分野で使われる組込み計算機の性能とコストを示し、表 2.2 で比較する。

表 2.2: 組込み計算機の用途と種類

	性能	省スペース	省電力	低コスト
家庭用電化製品	弱	強	強	強
自動車制御	弱〜中	強	強	中
ロボット制御	中〜強	強	強	中
監視・観測システム	中	強	強	強

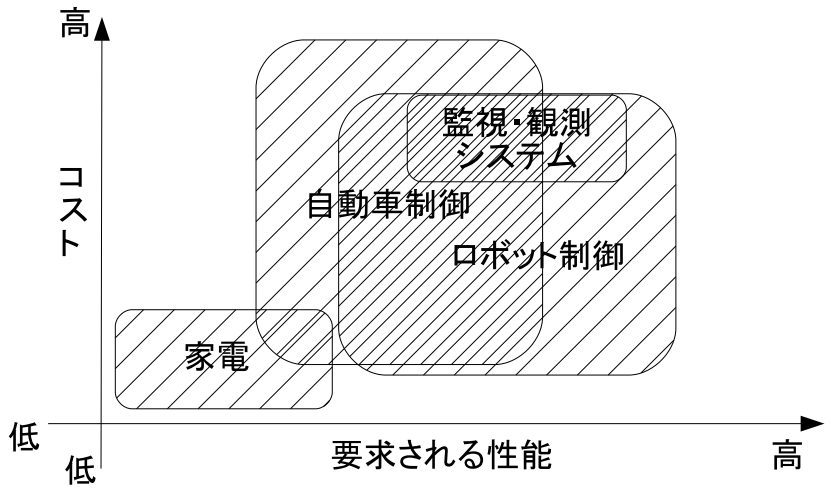

図 2.2: 組込み計算機の用途と種類

エアコン、冷蔵庫、炊飯器といった家庭用電化製品のような単純な機器では、性能への要求は大きくない。コスト削減が第一に求められるため、低性能な組込み計算機を用いて低コストを実現している。たとえば、家庭用電化製品で用いられる組込み計算機として、東芝社の TMP86FS49 というマイクロコントローラがある。8 ビットのプロセッサ TLCS-870/C を搭載し 16MHz で動作して、メインメモリを 2KByte 搭載している。性能が低い分、サンプル価格が 600 円と非常に低コストを実現している。

自動車制御では様々なところで組込み計算機が用いられている。この中でも特にエンジン制御ではコストへの要求よりも、応答速度保証、耐熱性、耐久性などの信頼性が求められている。たとえば、自動車のエンジン制御で用いられる組込み計算機では、Freescale 社の MPC5554 といったマイクロコントローラが搭載されているものがある。これは 80 MHz ～ 150 MHz で動作し、応答速度を保証するために高速で高性能なタイマを内蔵している。さらに、自動車のエンジンルームのような過酷な環境でも動作するように、動作温度が −40 °C ～ 150 °C となっている。

ロボット制御の分野では、大小さまざまなロボットを制御するために組込み計

算機が用いられている。最近では、ネットワークなどを用いて学習するロボットがあり、高性能な組込み計算機への要求が高い。たとえば、高性能な人型ロボットとしてユニー社の HR-Trainer II がある。このロボットの制御には Freescale 社の MPC5200 を利用しており、400MHz で動作する。メインメモリは 128MByte 搭載し、OS (Operating System) は NetBSD ベースの Speecys OS を用いており、非常に高性能な組込み計算機であると言える。

組込み計算機を用いたシステムとして、気象観測システムや自動販売機の状態報告をする遠隔テレメトリングシステムがある。この分野では極力コストを抑えつつ、要求された機能を実現しなければならない。すなわち、できるだけ低性能な組込み計算機を用いて要求された機能を実現するために、いかにして組込み計算機上で動かすソフトウェアを小さくしてハードウェア資源要求を抑えるかが問われている。この分野では観測、収集したデータを扱うためにネットワーク機能を用いたり、開発コストを削減したりするために用いるソフトウェアを安価なものにしている。そのため、OS には Linux などの Unix 系の OS が用いられることが多い。情報処理推進機構が実施した 2005 年版組込みソフトウェア産業実態調査 [1] によると、組込み計算機に搭載した OS として Linux は ITORON 仕様の OS に次いで多く利用されている。Linux 以外の POSIX/UNIX 仕様の OS を含めると、その割合は ITRON 仕様の OS とほぼ同程度で、全体の 2 割を占めている。しかし Linux やその周辺のソフトウェアは、組込み計算機向けのソフトウェアとしてはハードウェア要求が必ずしも低いとは言えない。本研究では、この問題点に注目し、議論を進める。

2.4 組込み計算機の問題点

組込み計算機は一般的な計算機と要求点が異なるため、ハードウェアの構成も異なってくる。省スペースのために、搭載できるハードウェアが限られてしまい、

一般的な計算機のような大容量の補助記憶装置を搭載することができなくなる。省電力を実現するために、一般的な計算機より低速な CPU を搭載せざるを得なかったり、ディスクのような電力を消費するハードウェアを搭載できなかったりする。また、低コストのために、運用時に必要のないインタフェイスを削減したり、実装するメモリを可能な限り削減したりする必要がある。

また、組込み計算機は一般的な計算機より資源に制限を受けるだけではなく、その資源を拡張する方法も制限されている。一般的な計算機と異なって、CPU の乗せ変え、メインメモリの増設、補助記憶装置の増設、といったことは想定されていないため行えない。これらの拡張を行うためには、あらかじめハードウェアの設計段階で考慮しなければならず、拡張性を高めたために、ハードウェアのコストやサイズが増大してしまうことが予想される。開発用にこれらの拡張が行える組込み計算機を用意する方法もあるが、コストの削減や開発サイクルの短縮の点で、開発用と運用で同じ組込み計算機を使用できることは利点となる。

組込み計算機はその要求点のために、利用できるハードウェア資源と拡張性に制限を受けるという現状がある。本研究では、このような資源の制限下における、組込み計算機自身での開発環境、及び運用時の保守環境を問題とする。

たとえば、組込み計算機上で動作するアプリケーションプログラムの開発の際には、組込み計算機自身でのデバッグやテストが必要である。しかし、組込み計算機は低コスト実現のために、運用時に必要のないキーボード入力、ビデオ出力機能を持たない。そのため、組込み計算機自身での開発が非常に困難となる。また、運用状態にある組込み計算機で、保守のために一時的に大きなアプリケーションプログラムを動作させたい場合や、大きなデータを出力したいといった場合、資源が制限されている組込み計算機では実現が困難である。

このように、組込み計算機には開発、保守が阻害されるという問題があり、組込み計算機を利用したシステム構築の際、大きな障害となる。

2.5 外部資源の利用

前節で述べた問題点を解決するために、組込み計算機では外部資源の利用が効果的であると考えられる。外部資源の利用とは、組込み計算機とは別の計算機に存在する資源を組込み計算機から利用することである。

組込み計算機は資源が制限されており、それを補うための拡張性も制限されている。そこで、組込み計算機に直接ハードウェアを増設するような拡張を行わず、外部資源を利用することで組込み計算機の要求点を満たしたまま、資源と拡張性の制限を解決できる。資源と拡張性の制限を解決できれば、組込み計算機での開発、保守環境の問題も解決される。

本研究では、外部資源を利用することで組込み計算機の開発、保守環境の改善を目指す。

第 3 章

既存手法の問題点と解決策

前章で、組込み計算機の開発、保守環境の改善のために外部資源の利用が効果的であると示した。本章では、外部資源を利用するための既存の手法を示し、その問題点について議論する。また、問題点に対して本研究での解決手法を提案する。

3.1 ソフトウェアによるハードウェア要求の問題

ある計算機から別の計算機のディスク資源を利用する手法として一般的に NFS (Network File System) などのネットワークファイルシステムが多く利用されてきた。また、キーボード入力やビデオ出力機能を持たない計算機に対しては telnet などのソフトウェアを用いてアクセスできる。ところが、これらのソフトウェアはプログラムサイズ、メモリ使用量といった面において必ずしも組込み計算機にとって適当であるとはいえない。

前章では、本研究で取り上げる範囲を気象観測や自動販売機の遠隔テレメトリングシステムで用いられる組込み計算機とした。たとえば、テレメトリングで用いる組込み計算機として BRAINS 社の超小型マイクロサーバがある。この組込み計算機の仕様を表 3.1 に示す。表 3.1 が示すように、非常にメモリや補助記憶装置が限られている場合、利用する OS やアプリケーションソフトウェアのプログラムサイズやメモリ使用量を押さえなければならない。アーキテクチャなどの環境にもよるが、NFS や telnet は概ね数百 KByte から数 MByte のメモリを使用し、

数百 KByte のプログラムサイズがある。この組込み計算機で動作する Linux などの Unix 系 は OS の動作だけでも数 MByte のメモリや補助記憶装置を占有する。このようにメモリや補助記憶装置の資源が制限されている状況において、NFS や telnet を利用することはメモリや補助記憶装置の不足を招く。また、仮想記憶のための補助記憶装置も十分でないため、メモリに関しては特に逼迫していると言える。

表 3.1: 超小型マイクロサーバの仕様 (BRAINS 社のカタログから引用)

CPU	SH3(100MHz)
メインメモリ	16MByte
補助記憶装置	Flash Memory 8MByte
基盤サイズ	51mm × 63mm
重量	約 20g
消費電力	0.6W ～ 2W

このように、組込み計算機のような資源が少ない環境において、組込み計算機でプログラムサイズやメモリ使用量を押さえるために、BusyBox [2] というソフトウェアがよく用いられている。BusyBox とは ls、cp、cat、sh など、Unix の基本コマンドを集めたプログラムである。通常これらのコマンドは独立したプログラムから成り立っており、コマンド毎にファイルシステムやメモリを利用する。BusyBox ではこれらのコマンドを単一のプログラムで実現しているため、プログラムサイズとメモリ使用量が少なくなるという利点がある。BusyBox では環境によるが数百 KByte から数 MByte のメモリ使用量削減を実現しており、この程度の単位での削減は十分に効果的で需要があるものと考えられる。たとえば、i386 アーキテク

チャ の Linux 2.4.27 での BusyBox と、それに対応する通常のコマンド 79 個のプログラムサイズを比較した場合、プログラムサイズは 2680KByte から 750KByte まで削減される。

したがって、組込み計算機では利用可能な資源は制限されているので、可能な限りプログラムサイズやメモリ使用量が小さい、コンパクトなソフトウェアで外部の資源を利用できることが好ましい。

3.2 通信規格の問題

従来、キーボード入力やビデオ出力機能がない組込み計算機に対しては、シリアルポート通信による入出力が用いられてきた。しかし、シリアルポートによる通信は低速で、組込み計算機で多く用いられる RS-232C の規格では最大ケーブル長を 15m としているため、遠隔地との通信には向いていない。そのため、遠隔地に設置した組込み計算機の遠隔保守が難しくなり、現地まで出向いて保守しなければならないため、コストが増大してしまう。

3.3 インタフェイスの問題

組込み計算機は一般的な計算機と比べてインタフェイスが乏しい。これは、組込み計算機が運用時の仕様に合わせて設計されるためである。それゆえ、開発、保守用に外部の資源を利用するためのインタフェイスを装備することはコストの増大を招く要因となる。また、組込み計算機自身のサイズを増大させ、省スペース性が失われる。

3.4 本研究で提案する手法

前節を踏まえて、本研究では組込み計算機の資源の制限を克服し、開発環境、保守環境を改善するための手法を提案する。本研究の提案手法では以下の点に留意する。

まず、資源拡張手法にはソフトウェアによる手法をとる。これは、ハードウェアによる手法では組込み計算機のコストやサイズを増大させてしまうためである。次に、資源拡張に用いるソフトウェアはプログラムサイズが小さかったり、メモリ使用量が少なかったりするなどのハードウェア要求の低いものとする。組込み計算機は資源が制限されているため、ハードウェア要求の高いソフトウェアは適さない。さらに、外部資源を利用するための通信手段として組込み計算機に搭載されているもの、または容易に搭載しうるものを用いる。本研究では、この条件を満たすイーサネットデバイスを用いる。

第 4 章

システムの概要

本章では本システムの前提条件を挙げ、本システムの概要を述べ、その利用法を提案する。さらに、より具体的な利用場面を挙げることで本システムの有用性を述べる。また、本システムの設計方針を示す。

4.1 システムの概要

本システムの概要を述べる。本システムは組込み計算機で動作するソフトウェアの開発者、及び運用者を対象とし、ネットワークを介して外部の計算機の資源を利用できるシステムを提供する。本システムの概要を図 4.1 に示す。

本システムを利用することで、以下のことが実現される。

外部資源の利用

本システムは組込み計算機に対して、他の計算機の資源を提供する。これによって、組込み計算機から様々な資源を利用することが可能となる。たとえば、キーボード入力やビデオ出力機能を持たない組込み計算機でキーボード入力やビデオ出力が可能となる。また、組込み計算機から大容量の補助記憶装置が利用可能となる。

コストの削減

本システムを利用することで、組込み計算機のコストが低くなる。これは既

存手法と比較して本システムがコンパクトであるため、ハードウェア要求が低くなるためである。

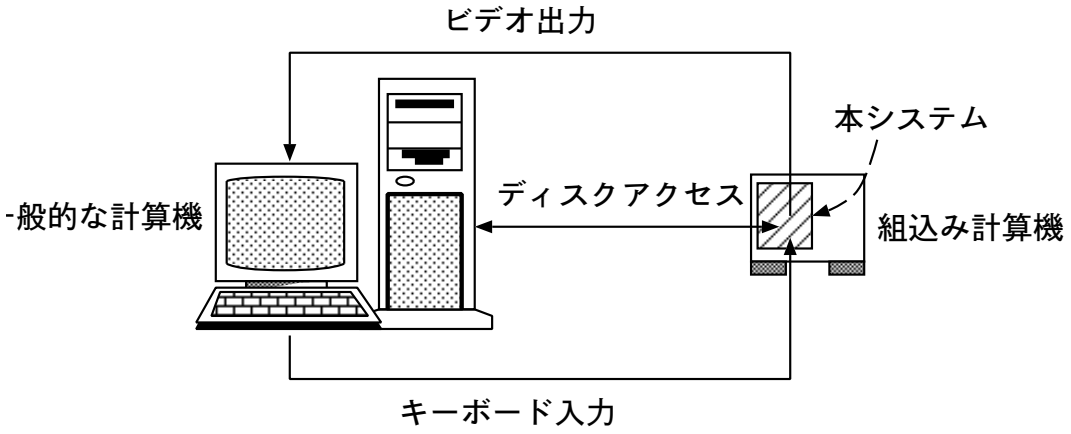

図 4.1: システムの概要図

4.2 システムの前提条件

本システムでは組込み計算機上で Unix または Unix ライクな OS が動作していることを前提とする。これは、近年は組込み計算機を用いたシステムでも複雑な処理が必要になり、ネットワークが多用され、OS も Unix 系が使われているためである。外部の計算機の資源を利用するための手段としてネットワークを用いるため、組込み計算機にはイーサネットが利用できることを前提とする。さらに、ネットワークは切断されることなく、常に接続されているものとする。

4.3 想定する利用法

本システムを利用することで、組込み計算機をネットワークに接続するだけで、様々な資源を利用できるようになる。利用可能になる資源としては、キーボード、ビデオ、ディスクなどが挙げられる。

たとえば、遠隔地での利用を想定してみる。遠隔地のような人が立ち入るのに手間や時間がかかる場所に組込み計算機を設置した場合、機器にトラブルが生じた際に実地に赴いてメンテナンスを行うことは、人的コストの上昇を招く。このような場合でも本システムを用いると、組込み計算機がワイヤレスネットワークなどを用いて、ネットワークにさえつながっていれば、本システムが提供するキーボード入力、ビデオ出力機能により遠隔地からメンテナンス可能である。

また、保守のときにのみ必要なアプリケーションプログラムがある場合、それを組込み計算機のディスク資源に格納しておくことは、資源に制限のある組込み計算機にとって望ましくない。このような場合、ネットワーク上にある別の計算機のディスク資源を利用することで解決できる。

4.4 実際の利用場面

組込み計算機を用いたシステムの開発者が、実際に本システムを用いて開発、保守を行う場面を想定してみる。

ある組込みシステムの開発者はクライアントから山間部の気象観測システムの構築を依頼された。クライアントからの依頼の要件は次のようなものであった。

- 山間部の気象観測システムを保守してほしい。
- 広範囲にわたってカメラやセンサを用いた観測をしたい。
- 都市部にある研究室からネットワークを用いて観測したい。
- 保守のために定期的にデータ採取ソフトウェアを走らせ、そのデータを都市部にある計算機に格納したい。
- コストはできるだけ最小限に押さえたい。

開発者はこの依頼に対して、図 4.2 のようなシステムを構築することにした。コストを抑えるために、現場に設置する観測システムには組込み計算機を用い、OS

は Linux を利用することにした。また、カメラやセンサを制御するソフトウェアを組込み計算機向けに用意することにした。

しかし、コストの制限から開発者が想定していたよりも低い性能の組込み計算機を利用せざるを得なくなった。そのため、利用可能なメインメモリは当初の予定の半分の 16MByte、補助記憶装置は FlashROM で 8MByte となった。このままでは OS と制御ソフトウェアだけでメインメモリがいっぱいとなってしまい、クライアントの要求である、遠隔地からの観測とデータ採取ソフトウェアが利用できなくなってしまう。

ここで問題となっているのは、必要なソフトウェアが組込み計算機に対してハードウェア要求が高い点である。まず、データ採取ソフトウェアを格納するための補助記憶装置の空き容量が不足していた。これを解決するために、都市部の計算機にデータ採取ソフトウェアを格納しておき、NFS を用いて組込み計算機に読み込むことを考えた。また、開発者は入出力を行うソフトウェアに対しては telnet を、保守のとき採取したデータを都市部にある計算機に格納するためのソフトウェアには NFS を用いようと思っていたが、これらのプログラムを利用するにはメインメモリが不足していた。

この問題に対してこの開発者はいくつかの案を考えた。一つは、性能の高い組込み計算機を導入することである。これによって、telnet や NFS といった既存のソフトウェアでシステムを構築することが可能である。しかし、これではコスト面でクライアントの要求を満たすことができない。

次に、開発者は本システムを用いることを考えた。本システムを用いることで、telnet の代わりに本システムの提供する入出力機構を用いての遠隔地からの制御が可能となった。また、 NFS の代わりに本システムの提供するディスク機構を用いてデータ採取ソフトウェアを組込み計算機に読み込んだり、採取したデータを都市部の計算機に格納することを可能となった。本システムは非常にコンパクトに設計、実装されているため、telnet や NFS を利用するには十分でなかった空きメモ

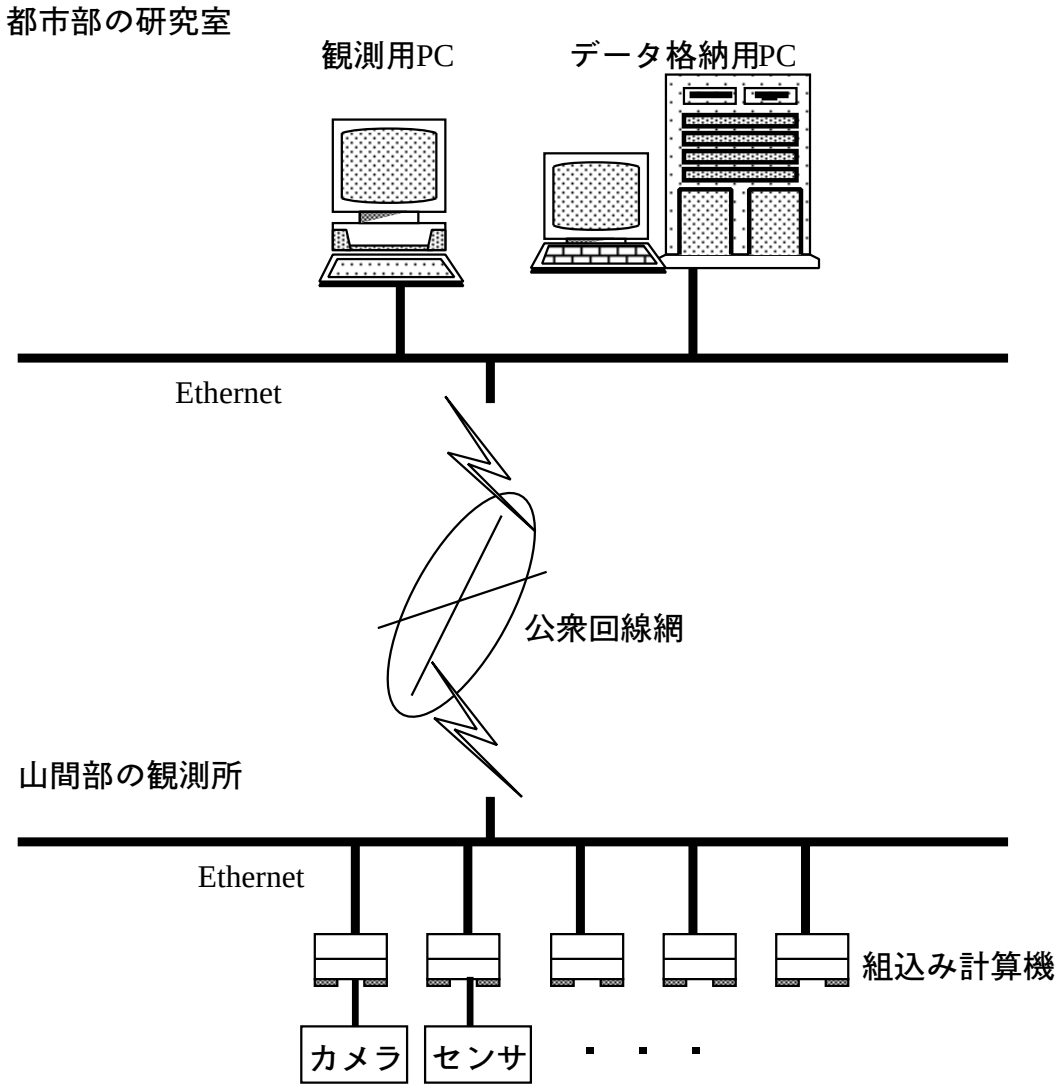

図 4.2: 実際の利用例 (気象観測システム)

リでも利用することが可能であった。

4.5 システムの設計方針

本システムの設計方針を以下に挙げる。

- 既存手法のソフトウェアよりコンパクト

  本システムは資源に制限のある組込み計算機上で動作するため、利用するためのハードウェア要求を低くしなければならない。

- 既存の資源と同等のセマンティクスを提供

  本システムを利用するためのセマンティクスを既存の資源を扱うものと同等にすることで、組込みシステム開発者が外部の資源を簡単に利用することが可能となる。たとえば、組込み計算機が Unix 環境の場合、デバイスファイルを用いて透過的に外部の資源を利用できるようにする。

- 既存システムへの影響を最小限に

  本システムを利用するにあたって、既存システムへの影響を最小限にする。近年の Unix ではカーネルの肥大化を防ぎ、複雑なハードウェア環境に適応するために追加モジュールによる拡張が一般的になっている。この機構はローダブルカーネルモジュール機構と呼ばれる。本システムでは、ローダブルカーネルモジュール機構を用いることで、本システムの導入を容易にし、既存システムへの影響を少なくする。

- ネットワークの利用

  本システムで組込み計算機が外部の計算機の資源を利用するためにネットワークを用いる。最近の組込み計算機にはイーサネットポートが搭載されていることが多い。イーサネットは RS-232C によるシリアルポート通信よりも距離的制限が厳しくない。また、既存のネットワーク環境との接続も容易である。

第 5 章

システムの実装

本章では本システムについて、構成、実装したプログラム、および、その使い方を述べる。また、本システムの実装において、ターゲット上で動作するプログラムに、デバイスファイルによる実装を採用した経緯についても議論する。

5.1 システムの構成

本システムの実装は Linux 上で行った。Linux は Unix ライクなオペレーティングシステムで、組込み計算機でも広く利用されている。

本システムの構成について述べる。本システムの構成を図 5.1 に示す。本システムはネットワークで接続された二つの計算機と、その上で動くプログラムで構成される。一方の計算機は資源が潤沢な計算機であり、もう一方の計算機に資源を提供する。この計算機をホストと呼ぶ。もう一方の計算機は資源が限られており、ホスト計算機から資源の提供を受ける。この計算機をターゲットと呼ぶ。ターゲットは組込み計算機を想定している。

前章で述べたシステムを実装するには、ターゲットではユーザプログラムとの仲介を行う必要があり、ホストでは実際の資源と仲介を行う必要がある。またこの二つ計算機を仲介する必要がある。本システムでは、ターゲットでユーザプログラムと仲介を行うプログラムとしてデバイスファイルを実装し、ホストで実際の資源と仲介を行うプログラムとしてデーモンプロセスを実装した。そして、この二

つのプログラムの間をソケット通信でデータのやり取りを行うことにした。

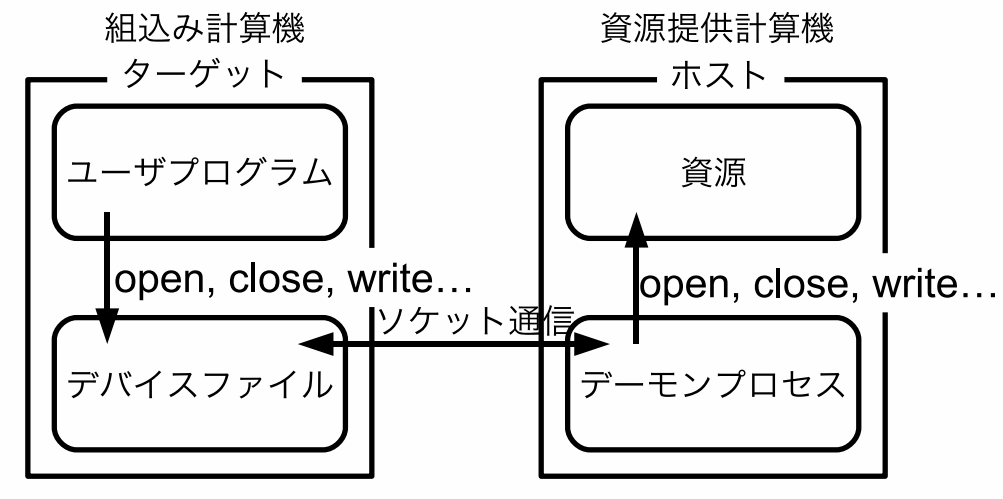

図 5.1: システム構成図

以下にそれぞれのプログラムの概要を述べる。

デバイスファイル

ターゲット上で動作するアプリケーションプログラムはデバイスファイルを介してホスト上の資源にアクセスできる。デバイスファイルはそのためのインタフェイスを提供する。

デーモンプロセス

デーモンプロセスはターゲットのデバイスファイルからの要求に対して返答する。たとえば、デバイスファイルから画面出力を要求されたとき、デバイスファイルから送られたデータをホストの画面に出力する。

ターゲットをデバイスファイルで実装することで、ターゲット上で動作するアプリケーションプログラムは既存ハードウェアと同じセマンティクスで資源を利用することが可能となるため、本システムの導入コストが低くなる。また、デバイスファイルによる実装では、Linux の持つローダブルカーネルモジュール機構を用い

て実装することができる。ローダブルカーネルモジュール機構を用いると、必要な機能を必要なときにのみ OS に組み込むことができるため、非常に柔軟な運用が可能となる。ターゲットの実装方式の検討についての詳細は次節で述べる。本システムでは、デバイスファイルはターゲット上のアプリケーションからの資源要求をホストにリダイレクトする役割のみを持つ。そのため、実装が簡略化されて、ターゲット上での本システムのプログラムサイズやメモリ使用量の削減が見込まれる。

ホストで動作するデーモンプロセスはユーザプログラムであるため、デバイスファイルよりも OS への依存度が低く、移植性が高くなる。そのため、ホストとなる計算機の選択の幅が広がる。また、ユーザプログラムはカーネルモジュールと比較してプログミングがし易いため、デーモンプロセスの実装が容易になる。

本システムは多種のホストの資源をターゲットから利用できるフレームワークを提供している。今回の実装では、2 種類のデバイスファイルとデーモンプロセスを実装した。一つはコンソール機構で、もう一つはディスク機構である。コンソール機構はターゲットに対してコンソールを提供し、ディスク機構はターゲットに対して仮想的なディスクを提供する。

5.2 実装方式

5.2.1 他の実装方式

本システムではターゲット上で動作するプログラムは、ローダブルカーネルモジュール機構を用いてデバイスファイルとして実装したが、ローダブルカーネルモジュール機構を用いた実装以外にも、実装方式として次のようなものが考えられる [3]。

ライブラリ実装

アプリケーションプログラムから呼び出し可能なユーザライブラリ関数とし

て実装する方式である。ライブラリ実装方式では、ホスト上の資源を利用するために、アプリケーションプログラムは本システムの用意したユーザライブラリ関数を用いる。

カーネル実装

カーネル実装方式では、元々カーネルに備わっているシステムコールを置き換え、ホスト上の資源を利用できるようにする。そのため、ターゲット上のアプリケーションプログラムは特に意識することなくホスト上の資源を利用できる。

5.2.2 実装方式の比較

表 5.1 に本研究の方式と上の二つの方式の比較を示す。

表 5.1: 実装方式の比較

	インタフェイス	セマンティクス	実装のし易さ	適用のし易さ
本研究の実装	○	○	△	○
ライブラリ実装	○	×	○	△
カーネル実装	×	○	×	×

以下に表 5.1 の各項目について比較し検討する。

インタフェイス

インタフェイスとは、ターゲット上のアプリケーションプログラムがホスト上の資源を利用するための明示的なインタフェイスが用意されているかという点である。本研究の実装はホスト上の資源に対応したデバイスファイルに

アクセスすることで、ホスト上の資源を利用できる。ライブラリ実装では用意されたライブラリ関数を用いることでホスト上の資源を利用できる。これらの実装に対して、カーネルによる実装ではカーネルのシステムコールを直接置き換えているため、ホスト上の資源を明示的に指定することができない。

セマンティクス

セマンティクスとは、ホスト上の資源を利用するためのインタフェイスが、一般的なハードウェア資源を利用するためのものと同等であるかという点である。資源を利用するためのセマンティクスが同等であれば、ターゲット上のアプリケーションプログラムは従来と同じセマンティクスでホスト上の資源を利用できるため、アプリケーションプログラムは開発が容易で、汎用的になる。デバイスファイルやカーネルによる実装では、アプリケーションプログラムは一般的なハードウェア資源を利用するのと同じように、open や write といったシステムコールを発行すればよい。ライブラリ実装ではこのようなセマンティクスを実現できない。

実装のし易さ

実装のし易さとは、本システム自体の実装の難易度を表す。一般的にカーネル空間でのプログラミングは困難で、ユーザ空間でのプログラミングは容易であるとされる。ライブラリ実装はユーザ空間で動作するため、プログラミングが容易で実装もし易い。カーネル実装では OS のカーネルそのものを置き換えるが、置き換える部分だけではなく、関係する箇所全ての理解が必要となり実装が困難である。本研究の実装方式では、ローダブルカーネルモジュール機構を用いているため、必要な機能のみを作成すればよく、カーネル実装よりも実装は容易である。

適用のし易さ

適用のし易さとは、本システムをターゲットに実際に適用するときの手順の

複雑さを表す。本研究の実装方式では、ローダブルカーネルモジュール機構を用いているため、ターゲットを停止させることなく、簡単に組込みと取り外しが可能である。また、一般的なハードウェア資源を利用するのと同じセマンティクスを持っているためアプリケーションプログラムの変更が少ない。ライブラリ実装では、アプリケーションプログラムに静的、または動的に組み込む必要があるため、アプリケーションプログラムの変更を要する。カーネル実装では、アプリケーションプログラムの変更は必要ないが、変更したカーネルをターゲットに適用するには、ターゲット自体をいったん停止する必要がある。

5.2.3 実装方式の検討

前章で述べた設計方針で、本システムは既存の資源利用法と同等のセマンティクスを提供し、既存システムへの影響を最小限とすることを目標に掲げた。ライブラリ実装では、そのようなセマンティクスは提供できないという点で不適格である。また、カーネル実装ではそもそもホスト上の資源を利用するための明示的なインタフェイスを提供できない。さらに既存システムへの影響が大きいという点でも不適格である。そこで、本研究ではこれらの設計方針を満たす、ローダブルカーネルモジュール機構を用いたデバイスファイルによる実装方式を採用した。

5.3 通信の流れ

デバイスファイルとデーモンプロセスの間の通信の流れを示す (図 5.2)。通信には Unix のネットワークで一般的に用いられている TCP/IP によるソケット通信を用いる。

1. デーモンプロセスはあらかじめ決められたポート番号でデバイスファイルからの接続を待つ。

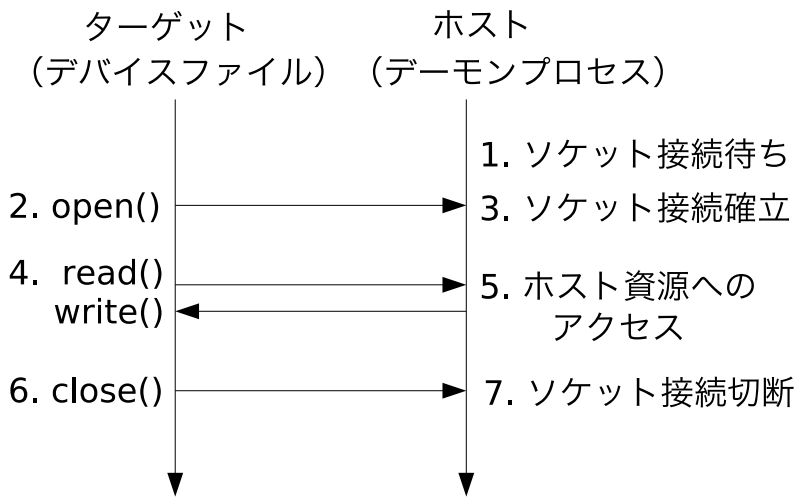

図 5.2: 通信の流れ

2. アプリケーションプログラムがデバイスファイルに対して `open` システムコールを発行すると、デバイスファイルはデーモンプロセスに接続要求を出す。

3. デーモンプロセスはデバイスファイルからの接続要求を受け、接続を確立する。

4., 5. アプリケーションプログラムがデバイスファイルに対して `read` システムコールや `write` システムコールを発行すると、デバイスファイルはデーモンプロセスとあらかじめ決められたプロトコルを用いてデータのやりとりを行う。

6., 7. アプリケーションプログラムがデバイスファイルに対して `close` システムコールを発行すると、デバイスファイルはデーモンプロセスから切断される。

5.4 プロトコル

デバイスファイルとデーモンプロセス間でデータのやりとりを行うとき、転送するデータの量などの情報をあらかじめ決めておく必要がある。ここでは、その通信方法を定めるプロトコルについて述べる。デバイスファイルとデーモンプロ

セス間でデータのやりとりするとき、そのプロトコルはデバイスがキャラクタデバイスかブロックデバイスによって異なる。本システムが提供するコンソール機構はキャラクタデバイスで、ディスク機構はブロックデバイスである。

5.4.1 キャラクタデバイス

コンソール機構のようなキャラクタデバイスの場合、デバイスファイルは次のようなプロトコルでデーモンプロセスと通信する。

`read` のとき

デバイスファイルは必要なデータサイズを伝え、データの受信を待つ。

`write` のとき

デバイスファイルは必要なデータサイズを伝え、データを送信する。

5.4.2 ブロックデバイス

ディスク機構のようなブロックデバイスの場合、デバイスファイルは次のようなプロトコルでデーモンプロセスと通信する。

`read` のとき

デバイスファイルはアクセスするディスクの位置、ブロック単位でのデータサイズを送信する。デーモンプロセスはその指定を受け、指定されたサイズのデータを送信する。デバイスファイルはデーモンプロセスに指定したデータサイズ分のデータを受信する。

`write` のとき

デバイスファイルはアクセスするディスクの位置、ブロック単位でのデータサイズを送信する。デーモンプロセスはその指定を受け、指定されたサイズ

のデータを受信するまで待つ。デバイスファイルはデーモンプロセスに指定したデータサイズ分のデータを送信する。

5.5 デバイスファイル

5.5.1 カーネル関数

デバイスファイルはカーネル空間で動作するため、ユーザ空間で利用できる関数は、デバイスファイルを実現するプログラムからは利用することができない。具体的には、socket、close、connect、read、write といった関数を利用することはできない。しかし、Linux では上記の関数に対応するカーネル関数が用意されており、これを用いることで同様の機能を実現できる。

また、デバイスファイルとアプリケーションプログラムでは存在するメモリ空間が異なるため、データのやり取りの際、カーネル関数を用いてバッファのコピーを行う必要がある。

本システムの実装では、表 5.2 のカーネル関数を用いた。

5.5.2 インタフェイス

デバイスファイルはアプリケーションプログラムから利用するためのインタフェイスを提供する。このインタフェイスは一般的なデバイスファイルにアクセスするためのインタフェイスと同等のものである。デバイスファイルに対する open、close、read、write のシステムコールはそれぞれに対応するデバイスファイルの関数に対応している。以下にデバイスファイルで定義している関数について示す。

open

open 関数はデバイスファイルを利用可能な状態にする。ターゲット上のプログラムはホスト上の資源を利用するために、まず、デバイスファイルに対して open システムコールを発行する。デバイスファイルは open システムコー

表 5.2: 利用したカーネル関数

関数名	機能
sock_create	ソケットを作成する
sock_release	ソケットを解放する
connect	ソケットを接続する
	(sock_create の戻り値が示すオブジェクトのメンバ関数)
sock_sendmsg	ソケットにデータを送信する
sock_recvmsg	ソケットからデータを受信する
copy_from_user	ユーザ空間からカーネル空間へバッファをコピーする
copy_to_user	カーネル空間からユーザ空間へバッファをコピーする

ルを受けるとホストのデーモンプロセスに対してソケット接続を試みる。このとき、デバイスファイル内では、まずカーネル関数 sock_create を用いてソケットを作成し、戻り値のオブジェクトの ops メンバに格納されている connect 関数を用いて接続している。

close

close 関数はデバイスファイルの利用を終了する。ターゲット上のプログラムがホスト上の資源の利用を終了するとき、デバイスファイルに対して close システムコールを発行する。デバイスファイルは close システムコールを受けると、カーネル関数の sock_release を用いて、デーモンプロセスとのソケット接続を切断し、ソケットを終了する。

read

read 関数はデバイスファイルからデータを読み込む。ターゲット上のプログラムはホスト上の資源からデータを読み込むときに read システムコールを発行する。read 関数は引数として、読み込んだデータを格納するバッファと読み込むデータ数をとる。このバッファはユーザ空間に存在している。read 関数は前述したプロトコルに基づいて、カーネル関数 sock_recvmsg を用いてデーモンプロセスと通信を行う。得られたデータはカーネル空間にあるため、カーネル関数 copy_to_user を用いて引数で指定されたユーザ空間のバッファへ書き込む。read 関数の戻り値として読み込んだデータ数を返す。

write

write 関数はデバイスファイルへデータを書き込む。ターゲット上のプログラムはホスト上の資源へデータを書き込むときに write システムコールを発行する。write 関数は引数として、書き込むデータが格納されているバッファと書き込むデータ数をとる。このバッファはユーザ空間に存在している。まず、write 関数はカーネル関数 copy_from_user を用いて引数で指定され

たバッファからデータをカーネル空間へ移す。次に、前述したプロトコルに基づいて、カーネル関数 sock_sendmsg を用いてデーモンプロセスと通信を行う。write 関数の戻り値として書き込んだデータ数を返す。

5.6 デーモンプロセス

デーモンプロセスはターゲットのデバイスファイルからの要求に対して返答する。デーモンプロセスの処理はコンソール機構とディスク機構で共通する部分と異なる部分がある。ここではそれぞれの機構に対しての、デーモンプロセスの振る舞いを述べる。

5.6.1 共通部分

デーモンプロセスを起動すると、まず、デバイスファイルと通信するためにソケット通信待ち状態に入る。デバイスファイルのソケットと接続されると、デーモンプロセスは fork システムコールを用いて子プロセスを生成し、データの読み込みを担当するプロセスと、データの書き込みを担当するプロセスに分かれる。このようにプロセスを分割して、役割を単純化する実装により、デーモンプロセスの実装を容易にすると同時に、読み込みと書き込みを平行して行うことができるようになり、システム全体のパフォーマンス向上が見込める。また、プログラム全体の見通しも良くなる。子プロセス生成後は各機構によって異なる処理が行われる。ターゲット上のプログラムがデバイスファイルに対して close システムコールを発行し、デバイスファイルがデーモンプロセスとのソケット通信を切断すると、デーモンプロセスは再び接続待ち状態に入る。

5.6.2 コンソール機構

処理の流れを図 5.3 に示す。前述したキャラクタデバイスのプロトコルに基づいて、データの読み書きを行う。読み込み時、すなわちキーボード入力のときは、ホストでキーボード入力を待ち、入力されたデータをデバイスファイルが指定した分だけ送信する。書き込み時、すなわちビデオ出力のときは、デバイスファイルから送信されたデータをホストの画面に出力する。

5.6.3 ディスク機構

処理の流れを図 5.4 に示す。前述したブロックデバイスのプロトコルに基づいて、データの読み書きを行う。本システムの実装では、ホスト上のファイルシステム上に大きな一つのファイルを用意し、それをディスクに見立てて読み書きする。ディスクの読み込みのときは、デバイスファイルが指定したディスク位置へファイルポインタを移動し、デバイスファイルが指定した分だけその位置からブロック単位でデータを読み込み、デバイスファイルに送信する。ディスクの書き込みの時は、デバイスファイルが指定したディスク位置へファイルポインタを移動し、デバイスファイルが指定した分だけブロック単位でデータを受信し、その位置へデータを書き込む。今回の実装ではディスク機構ではディスクのセクタ長を 512 Byte として、ブロック単位も同じ大きさとする。

5.7 システムの使い方

本システムはそれぞれ次のようにして利用する。

コンソール機構

本システムのコンソール機構は remoteconsole.o というカーネルモジュールと remoteconsoled というユーザプログラムで提供される。

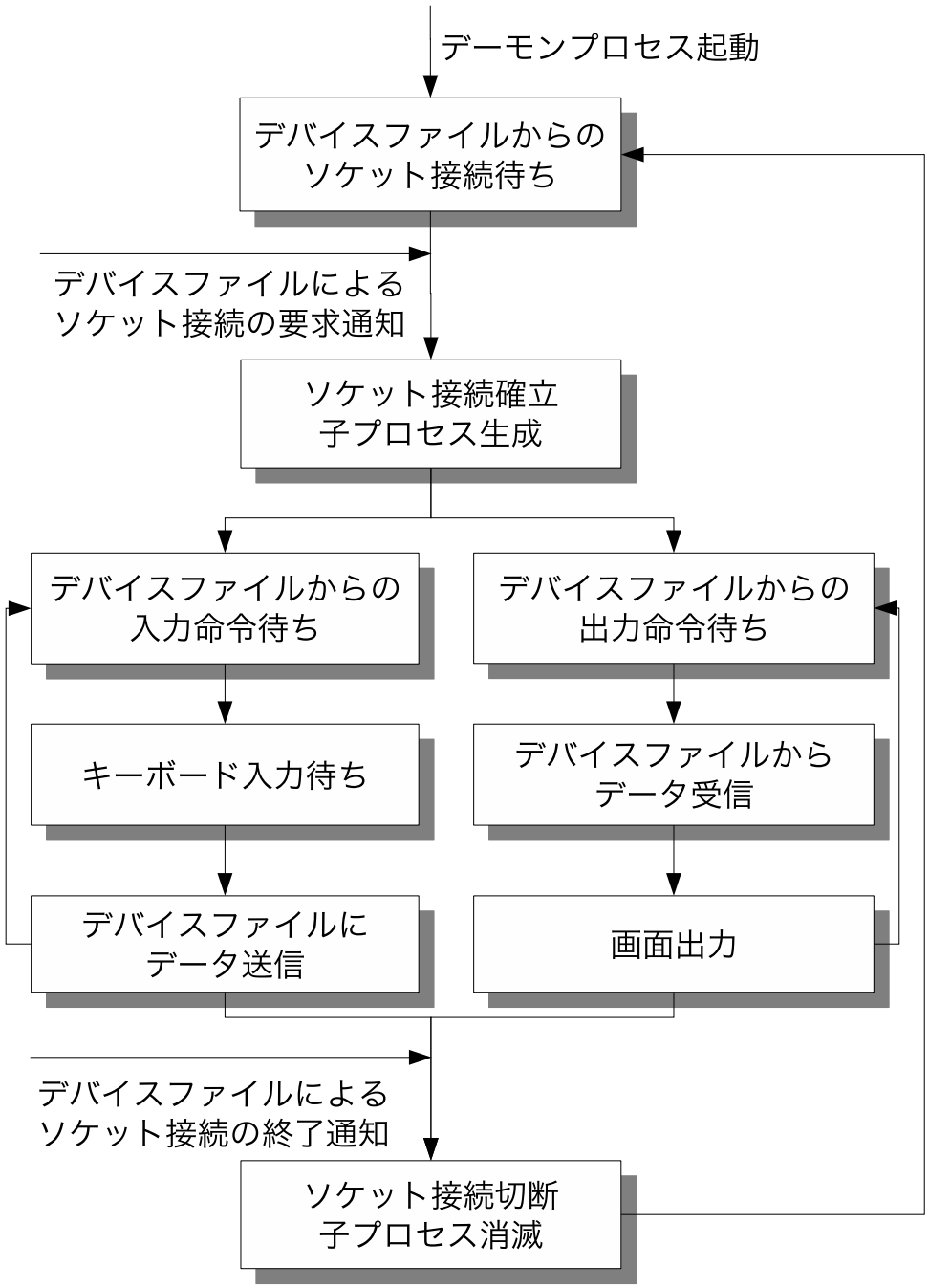

図 5.3: コンソール機構の処理の流れ

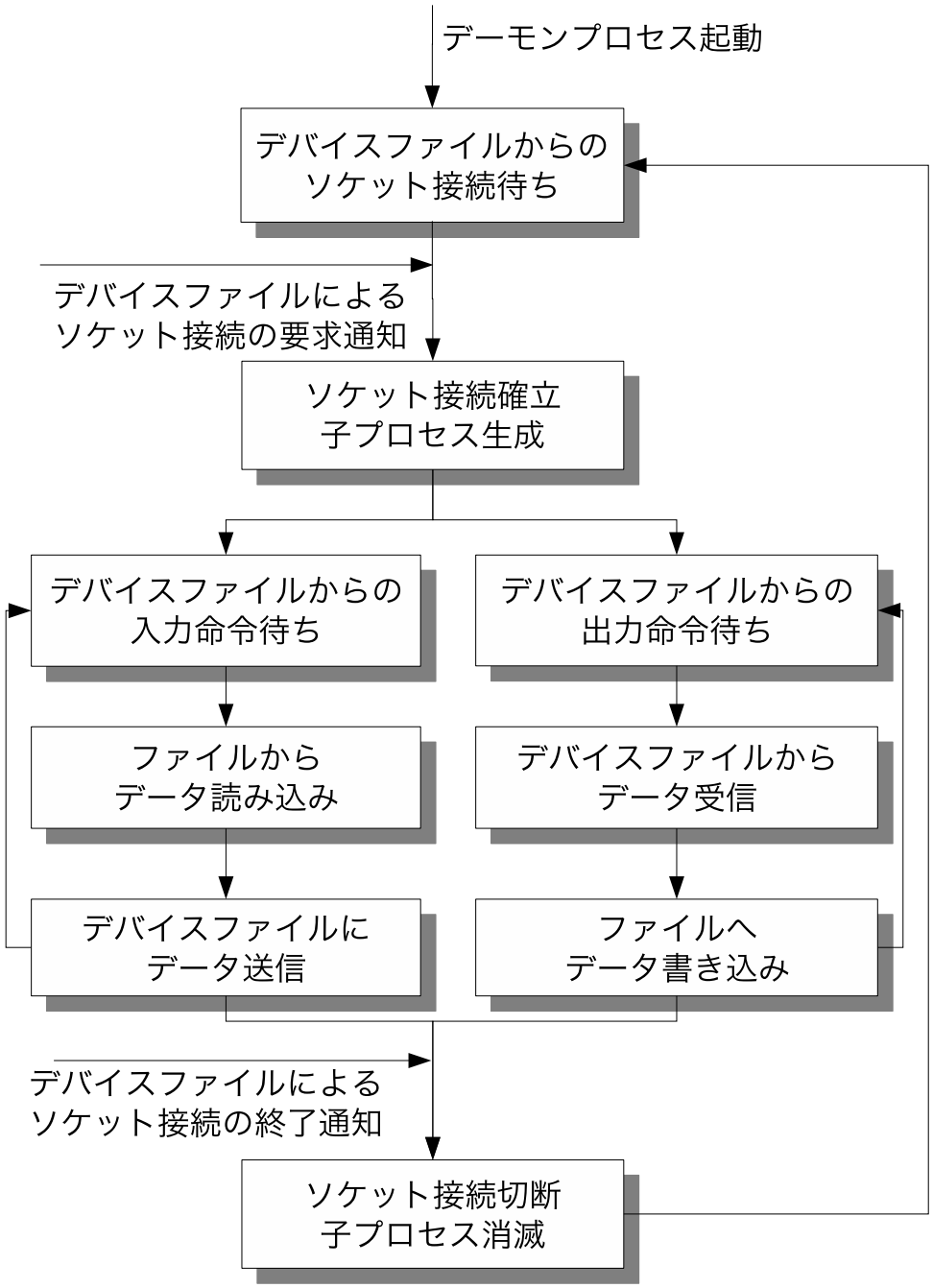

図 5.4: ディスク機構の処理の流れ

1. ターゲットで `insmod remoteconsole.o` として OS に本システムを組み込む。
2. `mknod /dev/remoteconsole c 60 0` としてデバイスファイルを作成する。
3. ホストでデーモンプロセス remoteconsoled を起動する。
4. ターゲットで `getty 9600 /dev/remoteconsole` を実行し本システムを通じてターゲットにログイン可能な状態にする。
5. ホストの画面にログインプロンプトが表示され、ターゲットへアクセス可能となる。

ディスク機構

本システムのディスク機構は remotedisk.o というカーネルモジュールと remotediskd というユーザプログラムで提供される。

1. ターゲットで `insmod remotedisk.o` として OS に本システムを組み込む。
2. `mknod /dev/remotedisk b 254 0` としてデバイスファイルを作成する。
3. ホストでデーモンプロセス remotediskd を起動する。
4. ターゲットで `mke2fs /dev/remotedisk` を実行しディスクのフォーマットを行う。
5. ターゲットで `mount -t ext2 /dev/remotedisk /mnt` を実行する。以後 /mnt へのアクセスはデーモンプロセスを通じてホスト上のディスクへのアクセスとなる。

第 6 章

システムの評価

本章では本システムと既存手法を比較する。本システムを既存手法のソフトウェアと比較して、本システムが組込み計算機に対して妥当であるかを検討する。特にここで検討する妥当性とは、本システムのハードウェアに対する要求の度合いと、既存システムとの性能比である。

6.1 評価環境

本システムの評価環境について述べる。本システムを図 6.1 のような環境で評価を行う。ホストとターゲットは表 6.1 のものを用いた。今回ターゲットには実験を簡単にするために、一般的なノート計算機を用いた。組込み計算機ではスワップ領域のための十分な補助記憶装置が用意できないことを想定し、ターゲットにはスワップ領域を確保しなかった。

6.2 コンソール機構の評価

ここでは本システムが提供するコンソール機構と既存手法のソフトウェアシステムである telnetd の評価を行う。

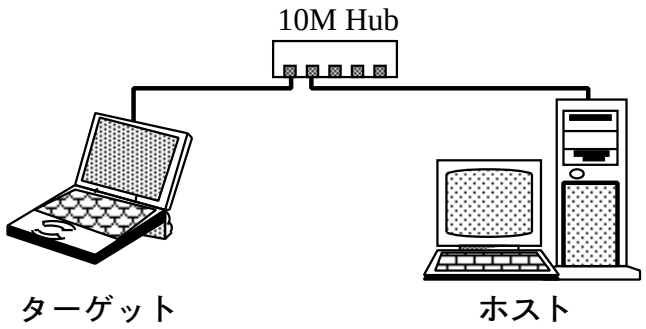

図 6.1: 評価に用いた計算機環境

表 6.1: 評価に用いた計算機環境

	ホスト	ターゲット
OS	Linux 2.4.27	Linux 2.4.27
CPU	Intel Pentium 4 3GHz	Mobile Intel Pentium 4 1.8GHz
メモリ	1024 MByte	512 MByte
ネットワーク	10/100/1000 Ethernet	10/100 Ethernet

6.2.1 測定

測定する内容は以下の通りである。

メモリ使用量

OS が起動してからそれぞれのシステムを利用可能にするために使用したメモリ量を測定する。

プログラムサイズ

それぞれのシステムのプログラムサイズを測定する。

6.2.2 結果

メモリ使用量

表 6.2 に以下の状態のメモリ使用量を示す。

- OS 起動直後の状態
- 本システムのプログラムを組み込んだ直後
- telnetd のプログラムを組み込んだ直後

既存手法のソフトウェアと比較して約 200 KByte の削減が達成された。

表 6.2: コンソール機構のメモリ使用量の比較

	本システム	telnetd
起動直後	23,476 KByte	23,476 KByte
プログラム組込み後	23,656 KByte	23,868 KByte
メモリ使用量	180 KByte	392 KByte

プログラムサイズ

次に、表 6.3 にそれぞれのシステムのプログラムサイズを示す。既存手法のソフトウェアと比較して約 31 KByte の削減が達成された。

表 6.3: コンソール機構のプログラムサイズの比較

本システム		telnetd	
プログラム名	サイズ	プログラム名	サイズ
remoteconsole.o	5 KByte	telnetd	36 KByte

6.2.3 考察

前節で述べた結果から、本システムのコンソール機構が既存手法のソフトウェアと比較して組込み計算機に適しているかを評価する。

メモリ使用量に関しては十分な使用量の削減を達成できた。本システムはカーネル空間で動作するのに対して、telnetd はユーザ空間で動作する。そのため、telnetd はスワップアウトが可能なため、メインメモリを圧迫しないと考えられる。しかし、組込みシステムではスワップ領域を確保することは困難なため、そのような事態を想定する必要はない。

プログラムサイズに関しては、あまり効果が得られなかった。これは比較対象である telnetd 自体が十分に小さいためである。

以上のことから、本システムは既存手法のソフトウェアと比較して、コンパクトなプログラムという観点から組込み計算機に適していると言える。

6.3 ディスク機構の評価

ここでは本システムが提供するディスク機構と既存手法のソフトウェアである NFS の比較を行う。ディスク機構の評価にはファイルシステムのベンチマークソフトである iozone [4] を用いた。また、本システムのディスク機構が提供するディスクについては、あらかじめ ext2 ファイルシステムでフォーマットする。

6.3.1 測定

測定する内容は以下の通りである。

転送速度

ターゲットからそれぞれのディスクにアクセスしたときの転送速度を測定する。測定する項目はシーケンシャルリード、シーケンシャルライト、ランダムリードとランダムライトである。転送速度を比較することで、本システムが性能面において既存手法のソフトウェアと代替可能であるか検証する。

メモリ使用量

OS が起動してからそれぞれのシステムを利用可能にするために使用したメモリ量を測定する。メモリ使用量を比較することで、本システムが既存手法のソフトウェアと比べて、ハードウェアに対する要求が低いことを検証する。

プログラムサイズ

それぞれのシステムのプログラムサイズを測定する。メモリ使用量を比較することで、本システムが既存手法のソフトウェアと比べて、ハードウェアに対する要求が低いことを検証する。

6.3.2 結果

転送速度

図 6.2〜図 6.5 に本システムと NFS の転送速度のグラフを示す。横軸は実際に転送したファイルサイズで、縦軸はそのときの転送速度である。OS やハードウェアのキャッシュ機構のため、ファイルサイズによって正確な転送速度が得られず、システムの性能の比較ができない。ここではキャッシュの効果が出やすい 100KByte 以下のファイルを除き、それよりも大きなファイルサイズで測定を行った。

書き込み速度については、本システムはランダムライト (図 6.2)、シーケンシャルライト (図 6.3) 共に安定しており、ファイルサイズが大きくなるほど速度が上がっている。対して、NFS はシーケンシャルライトについては小さなファイルに対しては高速であるが、2 MByte あたりから急激に速度が落ち込んでいる。また、ランダムライトは全体的に低速である。

図 6.4 と図 6.5 が示すように、読み込み速度は、全体的に書き込み速度より低速である。本システム、NFS ともにファイルサイズによる速度の大きな変化はみられない。本システムは NFS の 3 割程度の速度だった。

メモリ使用量

次に、表 6.4 に以下の状態のメモリ使用量を示す。

- OS 起動直後
- 本システムのプログラムを組み込んだ直後
- NFS のプログラムを組み込んだ直後

既存手法のソフトウェアと比較して約 400 KByte の削減が達成された。

プログラムサイズ

最後に、表 6.5 にそれぞれのシステムのプログラムサイズを示す。既存手法

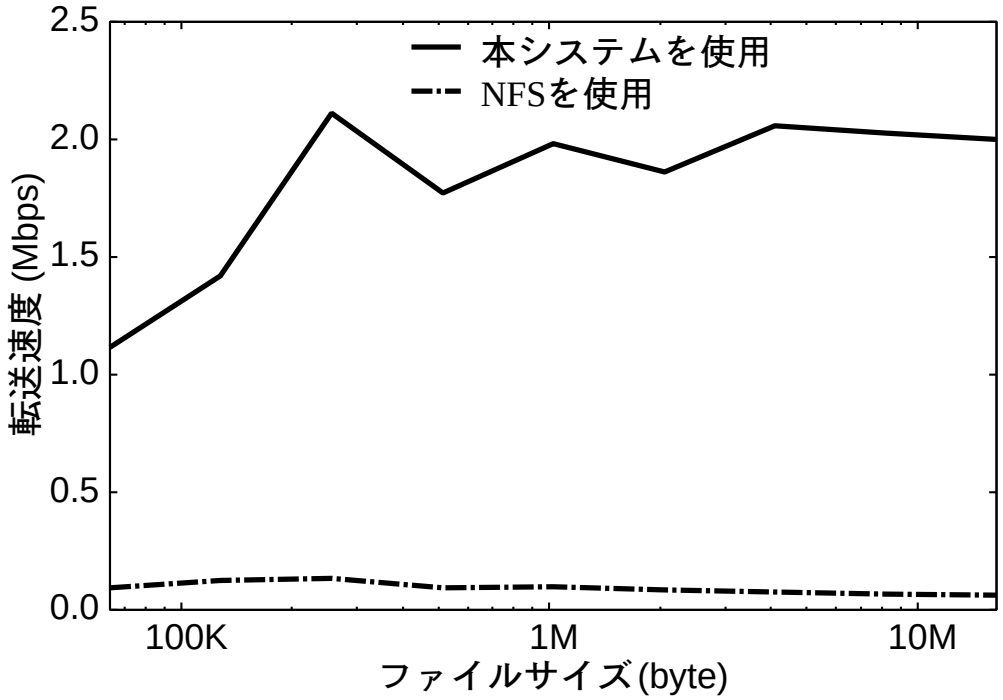

図 6.2: ランダムライトの転送速度

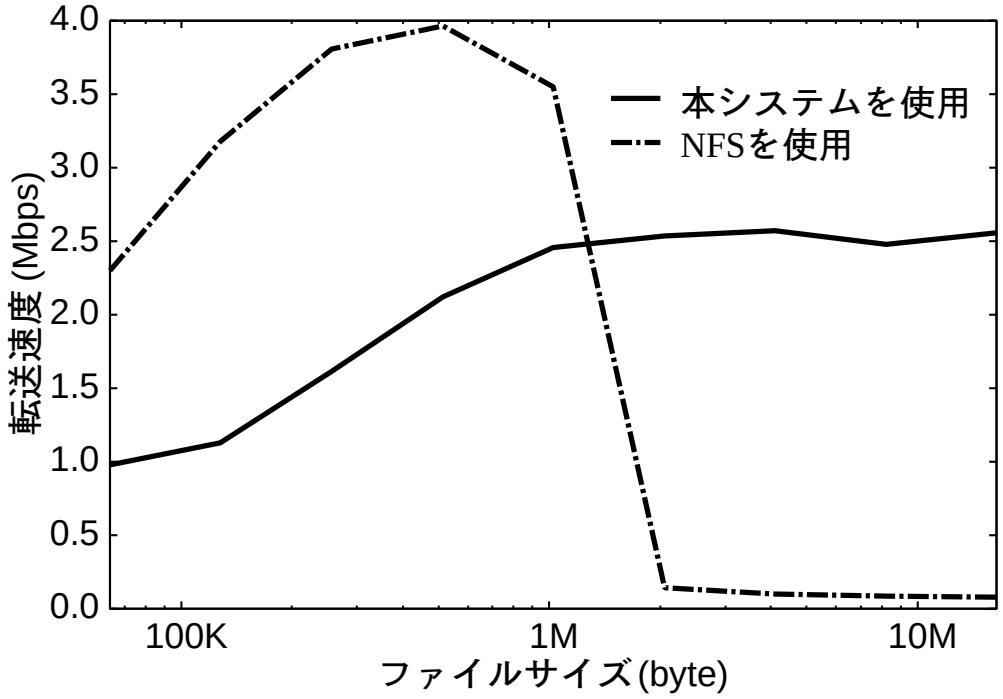

図 6.3: シーケンシャルライトの転送速度

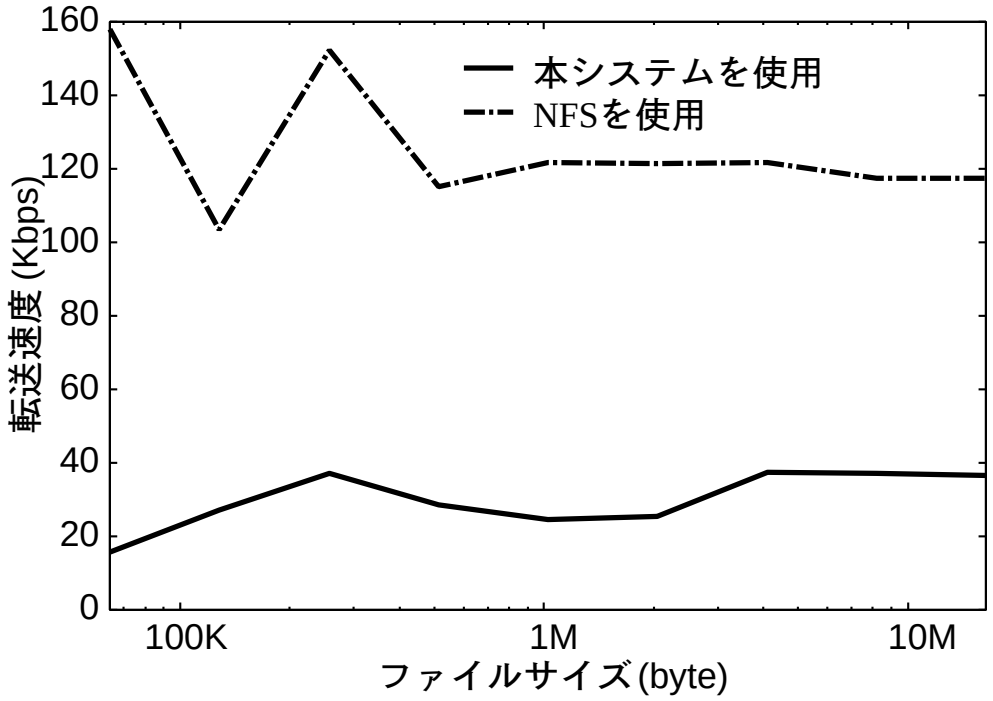

図 6.4: ランダムリードの転送速度

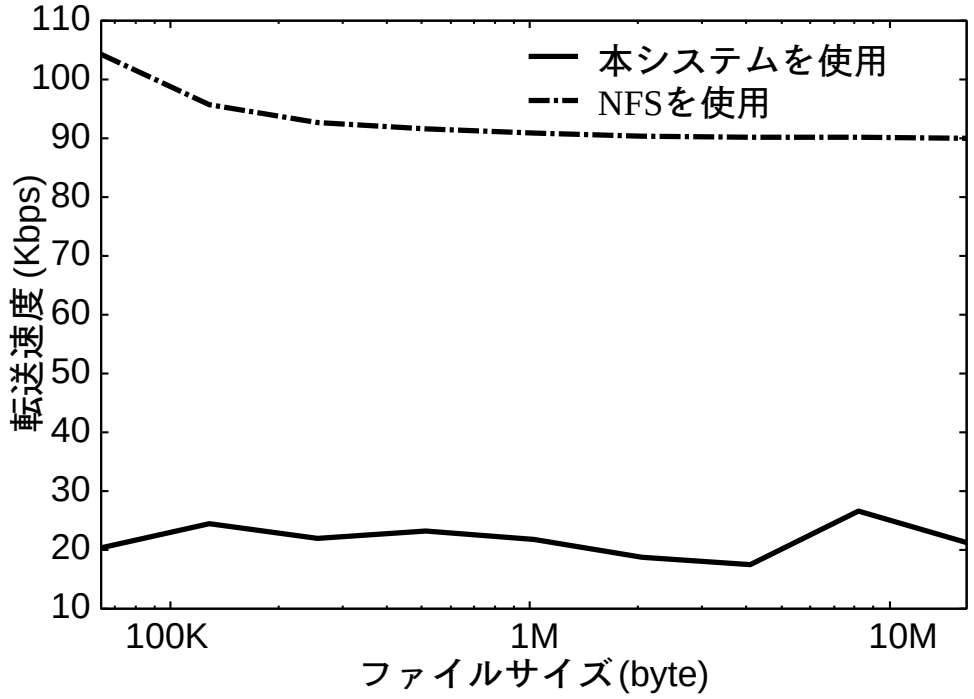

図 6.5: シーケンシャルリードの転送速度

表 6.4: ディスク機構のメモリ使用量の比較

	本システム	NFS
起動直後	23,448 KByte	23,448 KByte
プログラム組込み後	23,536 KByte	23,952 KByte
メモリ使用量	88 KByte	504 KByte

のソフトウェアと比較して約 237 KByte の削減が達成された。

表 6.5: ディスク機構のプログラムサイズの比較

本システム		NFS	
プログラム名	サイズ	プログラム名	サイズ
remotedisk.o	13.42 KByte	nfs.o	74.88 KByte
		sunrpc.o	107.08 KByte
		lockd.o	56.86 KByte
		portmap	12.44 KByte
合計	13.42 KByte	合計	251.25 KByte

6.3.3 考察

前節で述べた結果から、本システムのディスク機構が既存手法のソフトウェアと比較して組込み計算機に適しているかを評価する。転送速度に関しては、書き込み速度については既存手法のソフトウェアと比較して遜色なく利用可能である

といえる。また、読み込み速度については、既存手法のソフトウェアより 3 割程度に減少したが、この程度であれば十分許容範囲であると考えられる。

メモリ使用量に関しても組込み計算機においては十分効果的であると考えられる。たとえば、3 章で挙げたような組込み計算機では 16MByte のメインメモリを実装している。本システムによって節約された約 400KByte のメインメモリは、全体の約 3% に相当する。

また、プログラムサイズについては、3 章で挙げた組込み計算機では 8MByte の記憶装置を実装しているため、約 250KByte のプログラムサイズの削減は約 3% に相当する。そのため本システムによって多少の改善は見られる。

3 章で挙げた BusyBox では、数百 KByte から数 MByte の削減を実現している。本システムでは数百 KByte 単位での削減を実現しているため、ほぼ同程度の効果を挙げていると考えられる。

以上のことから、本システムは既存手法のソフトウェアと比較して、コンパクトなプログラムという観点から組込み計算機に適していると言える。

第 7 章

関連研究

この章では本研究の関連研究について述べ、比較を行う。

- USB over IP

  ネットワークに接続されたデバイスや、ネットワーク上の計算機に接続されたデバイスをネットワークを介して利用する手法として USB over IP [5–7] がある。この手法は Linux の USB ドライバスタックを用いて実装されている。そのため、利用できるデバイスが USB デバイスに限定される。本研究では USB ドライバスタックによらず、設計、実装しているため利用できるデバイスは USB デバイスに限定されない。

- Etherconsole

  Etherconsole [8] は Linux 上でのコンソールへのアクセスをイーサネットインタフェイスへリダイレクトする。対象はキーボード、ビデオ、マウスであるため、それ以外のデバイスには対応していない。そのためリモートにあるファイルシステムやディスクを利用することはできない。

- リモートデバイス

  ネットワークに接続された別の計算機上の資源（リモートデバイス）を利用するために、佐藤らはネットワーク透過な周辺機器制御の枠組みを提案し [9]、Linux 上で実装している。ターゲットにデバイスファイルを実装している点で、本研究と類似している。しかし、佐藤らによる実装では、ホスト側のプ

ログラムはカーネル空間で動作するように実装されているが、本研究のシステムではユーザ空間で動作するように実装している。ユーザ空間で動作するプログラムはカーネル空間で動作するプログラムに比べて実装が容易である。実装が容易で柔軟な対応が行えるため、たとえば、ホストの OS 環境が変わっても比較的容易に実装できるという利点や、新たなハードウェア資源への対応が容易であるという利点がある。また、佐藤らによる実装ではホットプラグには対応していない。本研究でのホットプラグへの対応策は次章で述べる。

- Personal Server

  身の回りにある様々な入出力デバイス、ディスクデバイスを無線ネットワークを介して借り受けて動作する機器として、Personal Server [10] が提案されている。Personal server は UPnP の技術を必要とし、これは計算機内部のデバイスを利用することは想定されていない。本研究ではホストが計算機内部に持つデバイスを利用することを目的としている。

第 8 章

今後の課題

8.1 ホットプラグへの対応

本システムはターゲットとホストの間のネットワークは常に接続されていることを前提としている。そのため、ネットワークが切断されたときや再接続されたときの本システムの挙動については保証しない。

そこで、以下の設計を追加することで上記のことに対応できると考えられる。

- 本システムがネットワークの切断を検知する。
- ネットワーク切断中、デバイスファイルやデーモンプロセスは転送すべきデータをバッファにためておく。
- ネットワーク切断中、デバイスファイルはアプリケーションプログラムに対して、デバイスがビジーであることを伝える。
- ネットワークが再接続されると、バッファに蓄積されたデータを転送する。

8.2 複数ターゲットへの対応

利用するターゲットが複数の場合、単一のホストでそれらを管理することでコストを削減できる。しかし、本システムの実装ではターゲットとホストは一対一対応を前提としており、多対一の状況は考慮されていない。そのため、ホスト側の

デーモンプロセスを改良し、接続してきたターゲットごとに対応できるようにする必要がある。

8.3 転送速度の向上

本研究で実装したディスク機構は NFS と比較して、特に読み込みが低速であった。この原因はディスク機構のデーモンプロセス側の処理にあると考えられる。図 5.4 のディスク機構の処理の中で、読み込みの処理はデバイスファイルからの入力命令を受けて、ファイルからデータを読み込み、デバイスファイルにそのデータを送信する、となっている。今回のディスク機構のデバイスファイルの実装では、ディスク I/O が 1 回発生するたびにデーモンプロセスにデータの要求を行っていた。今回の実装では、ディスク機構のディスク I/O ではセクタサイズが 512 Byte であり、ディスク I/O が発生するたび、デーモンプロセスは 512 Byte のデータをファイルから読み込み、デバイスファイルに送信していた。これらの処理は全てブロックするため、ディスクの読み込み、データの受信にそれぞれ待ち時間が生じる。例えば 100 KByte のデータを読み込むときは、上の処理を最低でも 200 回行わなければならず、その間に生じる処理の待ち時間は無視できないものであると予想される。

転送速度を改善するためにいくつかの方法が考えられる。まず、データを送信するときに、ディスク I/O 1 回に対して毎回すぐに転送するのではなく、複数のディスク I/O をまとめて処理するという方法がある。これによってデータの読み込み、データの受信を待つ回数が減り、結果として全体の転送速度の向上が見込まれる。このとき、いくつのディスク I/O をどのタイミングでまとめるかを調整するパラメータの選定が重要となると予想される。また、別の方法としてデータの読み込み、データの受信を非同期にして待ち時間をなくす方法も考えられる。

8.4 コンソール機構の充実

本研究で実装したコンソール機構では、open、close、write、read しか実装していない。そのため、完全なコンソールとは言えず、たとえば、ログインプロンプトでパスワード入力を隠すといった処理はできない。これらはコンソール機構に ioctl を実装することで解決できると考えられる。

第 9 章

まとめ

組込み計算機には資源が乏しく拡張性が低いという問題があり、組込み計算機自身での開発環境や保守環境が不十分であった。また、資源の乏しさや拡張性の低さを補うための既存手法のソフトウェアは、組込み計算機に対してハードウェア要求が高いという問題があった。そこで本研究では既存手法のソフトウェアに比べてハードウェア要求が低いコンパクトなシステムとして、デバイスファイルとデーモンプロセスを実装した。本システムは一般的なデバイスファイルと同じセマンティクスを提供しており、既存のアプリケーションプログラムから容易に利用できる。また、ローダブルカーネルモジュールとして提供しているため、既存の OS 環境を変更することなく利用できる。

本研究では、組込み計算機に対してディスクを提供する機構とコンソールを提供する機構を Linux 上で実装して評価を行った。既存手法のソフトウェアである NFS と telnetd と比較した結果、メモリ使用量とプログラムサイズの双方において十分にコンパクトであることを確認した。また NFS との比較においても代替可能な性能を得られた。

本システムを組込み計算機で利用することで、既存手法に比べて、ハードウェア要求が低くなるためハードウェアにかかるコストの低減が見込まれる。また、ソフトウェアによる資源拡張手法のため、資源拡張のためにハードウェア増設を行う必要がなく、ハードウェアにかかるコストが低減され、追加のハードウェアによって省スペース性が失われることもない。

謝辞

本研究を遂行するにあたっては、いろいろな方々にお世話になりました。

まず、指導教員の多田好克先生には日頃から熱心なご指導、そしてご鞭撻を賜わりました。ご多忙中にもかかわらず論文の草稿を丁寧に読んで下さり、大変貴重なご助言をいただきました。ここに厚く御礼申し上げます。また、貴重なご意見を頂き、適切なご助言を頂いた佐藤喬助手に感謝いたします

そして、本研究が行なえたことは、研究方針や方法論について議論をし、共に研究生活をおくってきた多田研と村山研の学生諸氏おかげでもあります。最後に、これらの皆さんに感謝いたします。

参考文献

[1] 独立行政法人 情報処理推進機構 2005 年版組込みソフトウェア産業実態調査, URL: http://sec.ipa.go.jp/download/200510ei.php (2006/2/14 参照).

[2] BusyBox, URL: http://www.busybox.net (2006/2/14 参照).

[3] 明神智之, 佐藤喬, 多田好克：組込み計算機を拡張するためのリモートデバイスの実現, 第 4 回情報科学技術フォーラム講演論文集, pp. 133–134 (2005).

[4] IOzone Filesystem Benchmark, URL: http://www.iozone.org (2006/1/31 参照).

[5] 佐藤純次, 河合栄治, 中村豊, 藤川和利, 砂原秀樹：リモート・デバイス利用に関する汎用的なフレームワークの設計と実装, 情報処理学会研究報告, 2002-OS-092, pp. 115–122 (2003).

[6] 広渕崇宏, 河合栄治, 中村豊, 藤川和利, 砂原秀樹：USB ドライバスタックを拡張したリモートデバイス利用方式, 情報処理学会研究報告, 2003-UBI-2, pp. 41–48 (2003).

[7] 広渕崇宏, 河合栄治, 中村豊, 藤川和利, 砂原秀樹：IP デバイス空間実現にむけた USB over IP によるデバイス制御機構の提案, 情報処理学会研究報告, 2003-OS-93, pp. 117–122 (2003).

[8] Kistler, M., Hensbergen, E. V. and Rawson, F.: Console over Ethernet, in *Proceedings of the FREENIX Track:2003 USENIX Annual Technical Conference*, pp. 125–136 (2003).

[9] 佐藤友隆, 中山健, 小林良岳, 前川守：カーネルレベルで実現したネットワーク透

過な周辺機器制御の枠組み, 情報処理学会研究報告, 2001-OS-087, pp. 97–104 (2001).

[10] Want, R., Pering, T., Danneels, G., Kumar, M., Sundar, M. and Light, J.: The personal server:Changing the Way We Think About Ubiquitous Computing, in *Proc. 4th International Conference on Ubiquitous Computuing(Ubicomp 2002)*, pp. 194–209 (2002).