システムソフトウェアと
オペレーティング・システム 87−13
（２００１．６．２９）

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

佐藤 友隆† 中山 健 小林 良岳 前川 守

電気通信大学 大学院 情報システム学研究科

要旨

LAN 上の他の計算機の周辺機器 (デバイス) を、あたかも直接接続されているかのように利用する汎用の枠組みを提案し、Linux カーネルを改造してプロトタイプを実装した。

カーネルレベルでデバイスのネットワーク透過性を実現しているので、デバイスドライバが提供するすべての機能を透過的に利用でき、たとえば他計算機に接続されているハードディスクをフォーマットしたりマウントしたりできる。また、ローカルな周辺機器用に作られたアプリケーションを再コンパイルや再リンクする必要が無い。

A Kernel-Level Framework for Network-Transparent Peripheral Control

Tomotaka Sato† Ken Nakayama Yoshitake Kobayashi
Mamoru Maekawa

Graduate School of Information Systems, University of Electro-Communications

Abstract

A general framework for using peripheral devices on a remote machine is presented. The framework provides the kernel-level transparency throught the network. A prototype system is built on the Linux kernel.

1 はじめに

最近は個人で複数の計算機を所有することも珍しくなくなっている。これらの複数の計算機を有効に利用するために、計算機同士をイーサネット等で相互に接続してLANを構築するケースが増えてきている。また、周辺機器の性能が向上し、これらの性能をフルに引き出すために、計算機と周辺機器をつなぐためのインタフェースとしてUSBやIEEE1394などの新しいインタフェースが登場してきている。これらの新しいインタフェースは、従来のものに比べてデータ転送速度が速く、ホットプラグも可能であるなど、非常に高機能で、計算機のユーザの多様化に伴って登場してきた、さまざまな周辺機器もこれらのインタフェースを利用している。

しかし、当然のことながら、計算機に新しいインタフェースが装備されていないと、これらの高性能な周辺機器や新しい機能は使用できないし、インタフェースが装備されている計算機であっても、周辺機器接続用のコネクタは計算機の背面に装備されている場合が多く、一旦、周辺機器を外してつなぎかえるのはかなりの手間となる。

これらの状況を考えたとき、LAN内で別の計算機に接続されている周辺機器を利用できれば、このような問題は解決できる。こうすることによって、手元の計算機に直接接続不可能な周辺機器も利用可能となり、物理的なつなぎかえの必要がなくなる。

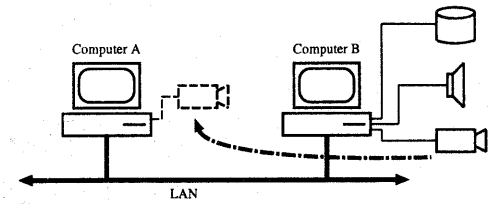

図 1: 仮想的に周辺機器を直接利用

2 目標

本研究の目標はLAN越しに、他の計算機に接続された周辺機器を利用するための汎用的なフレームワークを提供することである。具体的には、他の計算機に接続されている周辺機器を仮想的に手元の計算機に接続されているかのように扱えるようにすること、つまり、ネットワーク透過に周辺機器を利用できるようにすることである（図1）。

これまでにもネットワーク透過な周辺機器の利用方法にはさまざまなものがあった。だが、それらには以下のような問題点があって我々の目的には適当でない：

1. 特定の周辺機器専用である
2. 特別なライブラリを用いなくてはならない

上記1は主に、プリンタやディスクを共有するためのNFSやlpdなどについて言える。これらは特定用途の周辺機器専用に作られたものであり、周辺機器全般を含んだものではない。また、上記2はX Window System [1]について言える。Xではネットワーク上に存在する他の計算機のマウスやキーボードを抽象化した形でネットワーク越しに利用で

きるが、そのためにはXが提供する専用の関数を使用しなくてはならない。

これらの方式では、ある計算機に直接接続されている周辺機器と他計算機の周辺機器とでは利用方法に違いが生じて汎用性が下がり、アプリケーションの書き換えが必要となる。

本研究では、専用のライブラリを使用することなく、さまざまな周辺機器を利用できることを目標とする。このためネットワーク越しに周辺機器を利用する方法に、手元の計算機に接続された周辺機器を利用するための方法と互換性を持たせる。

これにより、既存のオペレーティングシステムで蓄積されたアプリケーションからネットワーク越しに周辺機器を利用する際も、アプリケーションを再コンパイル、再リンクする必要がない。これはアプリケーションの配布形態としてバイナリ形式でアプリケーションが配布される場合が少なくないことを考えると大きな利点となる。

また、プログラムをする側にも利点がある。特別なライブラリを使用して周辺機器をネットワーク越しに利用する場合、プログラマは新たな関数を憶える必要があるが、手元の計算機の周辺機器を扱う方法と同一の方法が使えるのであれば、そのような問題は出てこない。

3 周辺機器のネットワーク透過性の実現

本研究では実行環境としてLinuxを用い、他計算機に接続されている周辺機器も仮想的なデバイスファイルとして扱うことによって周辺機器を操作する方式を提供する。そのためには、デバイスファイルを扱うシステムコールに互換性をもたせればよい。システムコールレベルの互換性を保つだけならば、図 2の点線1の階層、つまりユーザ空間でネットワーク透過性を実現することもできる。

Linuxでは、デバイスファイルを操作するときでも通常のファイルを操作するときと同じシステムコールを利用し、プログラムからはファイルの種類を区別する必要がない。ただし、通常のファイルを操作するときと違って、デバイスファイルを操作するためのシステムコールを使用すると、カーネル内ではデバイスドライバが呼び出される。デバイスドライバにはシステムコールに対応した周辺機器を操作するための関数が用意されており、システムコールに対応した関数が呼び出される。

しかし、デバイスドライバが提供する機能には、アプリケーションがシステムコールという形で利用できる機能のほかに、アプリケーションから直接には利用できない機能もある。例えば、ブロックデバイスのデバイスドライバが持つ機能には、バッファキャッシュ向けに用意された機能が存在し、これらの機能はアプリケーションが直接利用することは出来ない。

このように考えると、システムコールレベルの互換性を実現しただけでは、ブロックデバイスのような、カーネル内で使用されることもあるデバイスドライバには対応できない。よって、図 2の点線2のレベルでネットワーク透過性を実現し、デバイスドライバが仮想的に存在するように見せる方がバッファキャッシュやファイルシステムに対しても、周辺機器のネットワーク透過性を提供できる

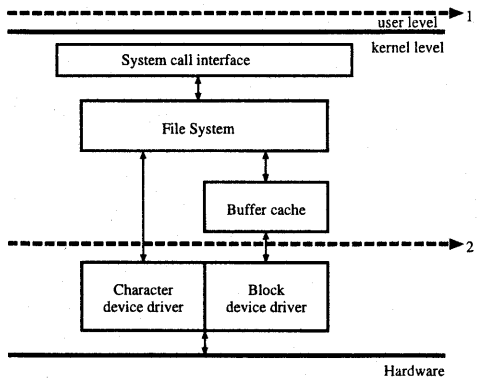

図 2: カーネル内の構成

ようになる。

カーネルレベルで実現することの利点として、上記のファイルシステムのようなカーネル内で周辺機器を利用する機能に対してもネットワーク透過性を実現できることが挙げられる。たとえば、他計算機に接続されているハードディスクをフォーマットした後にマウントするといったことが、手元の計算機に接続されているかのごとく行える。つまり、NFS がファイルシステムというディスクの機能を抽象化したものだけを扱えるようにしたのとは異なり、ディスクのデバイスドライバが提供するすべての機能を利用できるようになる。このようにして、LAN 越しに周辺機器を利用するときでも、仮想的に周辺機器が接続されているかのように扱えるようになる。

4 関連研究

ネットワーク越しに他の計算機に接続されている周辺機器を利用するための研究はいくつか存在する。ここでは、それらの研究を紹介する。

4.1 Solelc

Solelc [2] は立命館大学で開発中の分散オペレーティングシステムである。LAN でつながった複数の計算機に対して，位置透過な資源管理を可能とする環境を構築し、この環境上でカーネルを動作させることによって、オペレーティングシステムの各機能を位置透過に動作させている。

Solelc では OS を抽象化層とカーネルの 2 層に階層化している。抽象化層は各計算機ごとに動作しており、計算機の持つ資源 (ハードウェア) を管理、操作する。抽象化層は計算機資源を抽象化し、カーネルはこの抽象化層の上で動作する。

抽象化層は他の計算機と協調動作して、位置透過性を実現している。つまり、ハードウェアに近い階層で位置透過性を実現しているので、カーネルより上の階層で動作するソフトウェアはどの計算機かに依存せずに、動作できる。

4.2 Wapplet

Wapplet [3] は慶応大学で研究中の、ウェアラブルコンピュータのような可搬性が高く、計算機資源の乏しい端末に適したアプリケーションフレームワークである。このフレームワークの利点はネットワークにさえ接続できれば計算能力が低いウェアラブルコンピュータでも高度なアプリケーションを実行できることである。

Wapplet では、アプリケーションの一部または全部がウェアラブルコンピュータとプリ

ンタやディスプレイなどサービスを提供するホストに移動し、連携動作する。

このシステムはJavaを用いて実装されており、アプリケーションの一部の移動は、Javaオブジェクトの移動として実現されている。

5 フレームワーク

本研究で提案するフレームワークは、従来の周辺機器利用のためのインタフェースと互換性を保ちつつ、カーネル内の機能にもネットワーク透過性を提供する。そこで、本研究ではカーネル空間内でネットワーク透過性を実現し、ネットワーク越しに周辺機器のコントロールする機構を設け、LAN上の他計算機に接続された周辺機器を利用できるようにする。

5.1 前提

本研究ではLAN環境を前提とし、データ送受信の大幅な遅延やデータの損失は無いとする。また、個人が管理できる範囲で複数の計算機を使用しているものとし、セキュリティについては考慮しない。

以下の説明では、周辺機器を利用するソフトウェアが動いている計算機を**呼び出し側計算機**、実際に周辺機器が接続されている計算機を**周辺機器側計算機**と呼ぶ。

なお、これらの故障は役割を表しているだけで、特定の計算機に固定されるものではない。ユーザはどの計算機に使いたい周辺機器が接続されいるかを把握しているとし、それを呼び出し側計算機から使用することになる。

5.2 システムの概要

本研究ではデバイスドライバのレイヤでネットワークを通して通信するために、呼び出し側計算機に**スタブドライバ**、周辺機器側計算機に**デバイスデーモン**を設けた(図 3)。基本的には、クライアント・サーバ方式のシステムとなっている。

呼び出し側計算機のアプリケーション(ユーザプロセス)が`write( )`システムコールを呼び出すと、そこで渡されたデータは、ファイルシステムを経由してスタブドライバへ渡される。スタブドライバはこのデータとドライバ内の要求された関数の情報を周辺機器側計算機のデバイスデーモンに転送する。

スタブドライバは渡されたデータをLAN上の周辺機器側計算機となっている計算機に渡されたデータと要求されている周辺機器の操作、つまり、デバイスドライバが用意している周辺機器の操作の情報を周辺機器側計算機に送信する。

周辺機器側計算機ではデバイスデーモンがスタブドライバからの要求を待っていて、要求があったら対応するデバイスドライバの関数を呼び出す。呼び出し側計算機から渡されたデータをこの関数の引数として渡す。周辺機器からの返り値などの情報は、呼び出し側計算機のスタブドライバに返される。

呼び出し側計算機のスタブドライバはデバイスデーモンからの返り値をアプリケーションに返す。

6 実装

プロトタイプシステムはAT互換機上で動作するLinuxカーネル(バージョン2.2.16)を

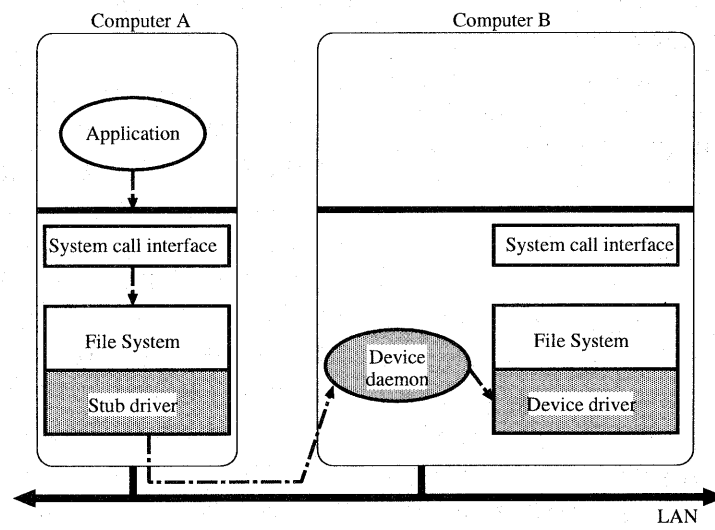

図 3: フレームワーク

改造して実装した。Linux にはカーネルの動作中にカーネルの一部を導入および削除できる機能があり、この機能を利用した。

TCP を用い、現段階では特別な通信制御は行なっていない。

6.1 スタブドライバ

呼び出し側計算機では、アプリケーションからの要求を受けるためのスタブドライバを実装する。スタブドライバ内では、アプリケーションから渡されたデータを周辺機器側計算機に転送する、あるいは周辺機器側計算機から来たデータを受信しアプリケーションに渡す機能が必要である。

スタブドライバはカーネル内では通常のデバイスドライバとして実装する。なお、実装には Linux が提供するモジュール機能を利用した。通常、Linux では周辺機器の操作は該当するデバイスファイルの操作ということになるが、デバイスドライバはこれらのデバイスファイルの操作（システムコール）に対応した関数（Linux ではメソッドと呼ばれる）で構成されている。

デバイスドライバという形で実現してあるのは、アプリケーションから通常のデバイスドライバように見せるためである。これによって、アプリケーションは従来通りのシステムコールを用いて、あたかも、自分の計算機に周辺機器があるように、周辺機器をコントロールできる。

データの送受信が発生するのは、アプリケーションがデバイスファイルへの操作を行ったときである。ファイルへの操作を行うときに主に使用されるシステムコールは open( )、close( )、read( )、write( ) である。今回の実装では、これらのシステムコー

ルに対応するデバイスドライバ内のメソッドにネットワーク越しのデータの送受信ができる機能を実装した。

各メソッドにおいて実装した機能は次のとおりである。

open( ) デバイスデーモンに接続に接続する

release( ) デバイスデーモンとの接続を切る

read( ) デバイスデーモンにデータの読み込みを要求しデータを受信する

write( ) デバイスデーモンにデータの書き込みを要求しデータを送信する

これらのデータ送受信の機能は、カーネル内に用意されているネットワーク接続機能(socket)を利用して実現した。送受信されるデータはアプリケーションがシステムコールを呼び出したときに渡す引数を介してやりとりされる。

以上のメソッドを用意すれば、周辺機器を扱う分には十分であると思われる。しかし、ブロックデバイスのような周辺機器には対応できていない。ブロックデバイスは上記のメソッド以外にファイルシステムとブロック単位のデータのやり取りを行う関数が用意されており、ブロックデバイスのデータの読み書きを行っている。今回の実装でブロックデバイスに関しては、この関数にread( )とwrite( )に相当する機能を実装した。

6.2 デバイスデーモン

周辺機器側計算機に用意するものはデバイスデーモンである。このデーモンはカーネルスレッドとして実装されるもので、特権モードで動作するのでカーネル空間内の全ての関数や変数にアクセスできる。デバイスデーモンは呼び出し側計算機のスタブドライバからの要求を受け付ける窓口となるデーモンとなる。デバイスデーモンはスタブドライバから要求のあったメソッドを呼び出して、付随するデータを渡す働きをする。

通常、デバイスドライバのメソッドやシステムコールなどのカーネル内の関数は、それを実行したプロセスのコンテキストで動作する事になっている。デバイスデーモンはカーネルスレッドというプロセスとして動作しているので、デバイスドライバのメソッドを実行するときは、デバイスデーモンのコンテキストで動作している。

本研究では実装を進めるにあたって、できる限りカーネル内の他の機能と分離することを目指していた。しかし、Linuxの仕様上、デバイスドライバのメソッドはファイルシステム経由で呼び出されることが前提となっており、ファイルシステムで管理している`file`構造体や`inode`構造体を渡す必要がある。

よって、デバイスデーモンはデバイスドライバのメソッドを呼び出すのだが、このために、カーネル内のファイルシステムが管理している`file`構造体を利用してメソッドを呼び出す方法を取っている。これは、linux-2.2.16の仕様で、`file`構造体にデバイスドライバのメソッドが登録されているのを利用したためである。

7 議論

ここでは、関連研究で挙げた研究と本研究との比較を行い、それぞれの長所と短所を述べる。関連研究で挙げた研究はいずれも周辺

機器がネットワーク透過に利用できるものであるが、ネットワーク透過性を実現したレベルに違いがある。以下では実現したレベルの違いごとに、議論を進めてゆく。

Wapplet は Java バーチャルマシン上で動作するシステムであり、ある意味、ユーザ空間のレベルでネットワーク透過性を実現しているといえる。この研究の利点として、Java バーチャルマシンの上で実行されるシステムなので、Java の持つ豊富な機能を十分に利用できることである。また、本研究のようにカーネルに変更を加えるといった、利用時に特別な権限が必要になることもない。しかし、Java バーチャルマシン上に特化しているので、本研究のフレームワークのよりも、汎用性は低くなると考えられる。

Solelc はネットワーク透過性を、カーネルよりも低いレベルである抽象化層で実現している。よって、CPU やメモリのような計算機資源もネットワーク透過にしている。本研究のフレームワークは周辺機器に特化している。これは、Solelc のように新しく OS を作成するのではなく既存の OS で利用できる形にしたかった事と、他計算機の計算機資源で主に使いたくなるのは周辺機器であると想定したので、このようなフレームワークとなった。

8 まとめ

本論文では、ネットワーク透過に周辺機器を操作するフレームワークを提案し、その実装および評価を行った。提案したフレームワークはカーネルレベルでネットワーク透過性を実現しているので、ユーザ空間で実現されたものと違い抽象化された周辺機器の機能を利用するのではなく、周辺機器が持つ全ての機能をネットワーク透過に利用できるようになっている。

今後は、ホットプラグへの対応、USB デバイスや SCSI デバイスのデバイスドライバのような、階層化されたデバイスドライバへの対応などを行う予定である。

参考文献

[1] Robert W. Scheifler and Jim Gettys. The X Window System. ACM Transactions on Graphics (TOG), Vol.5, No.2, pp. 79-109, April 1986.

[2] 芝 公仁, 大久保 英嗣. 分散オペレーティングシステム Solelc の構成. 信学技報 CPSY2000-38, pp. 57-64. 電子情報通信学会 コンピュータシステム研究会, May 2000.

[3] 村瀬 正名, 若山 史朗, 権藤俊一, 永田智大, 西尾 信彦, 徳田 英幸. 計算機環境に応じてサービス提供方式を適応させる通信用ツールキット. 情報処理学会研究報告 2000-OS-84, pp. 31-38. 情報処理学会 システムソフトウェアとオペレーティング・システム研究会, May 2000.