多 OS に対応したユーザーモードファイルシステムフレームワーク UniversalFUSE

池嶋 俊† SURÁNYI, Péter † 星月 優佑‡ 古橋貞之‡ 加藤 和彦† ‡

本論文では、多 OS に対応したユーザーモードファイルシステムフレームワークである UniversalFUSE を提案する。UniversalFUSE はファイルシステムをユーザーモードで実装できるようにし、かつ、一度のプログラミングで多数の OS に対応させる事を可能にする事ができる。本論文では、この UniversalFUSE の設計と実装について述べる。

1. はじめに

旧来、ソフトウエアは、一台のホスト内で完結して動作する事が一般的であった。しかし、近年のインターネットの普及により、複数のホストにまたがって動作する形のシステムが増えてきた。さらに、SOAP や XML-RPC といった標準化された規約によって異なるサービス提供者によって設置されているホスト間で連携が行われる事や、広くインターネットで RPC インターフェースを公開し、不特定多数のホストが連携する形を取るシステムなど、さらに複雑な形のマルチホストソフトウエアも出現している。

このように、複数のホストが連携したマルチホストシステムの場合、それぞれのシステムで使用される各ホストで動作するソフトウエアは、ホスト内で動作が完結するシングルホストソフトウエアとは違い、他のホストと連携して動作する事が求められる。そのため、既存のシングルホストソフトウエアをこのような複数のホストが連携するようなシステムでも使用する事はできず、新たに開発をしなおす必要がある。しかし、既存のソフトウエアはかなりの数が存在するがこれらは全てかきなおす必要がある。

一方、このように他のホストとの連携が必要な場合でも、既存の単一ホスト内で動作するように書かれたソフトウエアを変更せず、または、わずかな変更のみで他のホストとの連携機能を利用できる方法がいくつか提案されている。そのうちの一つに、

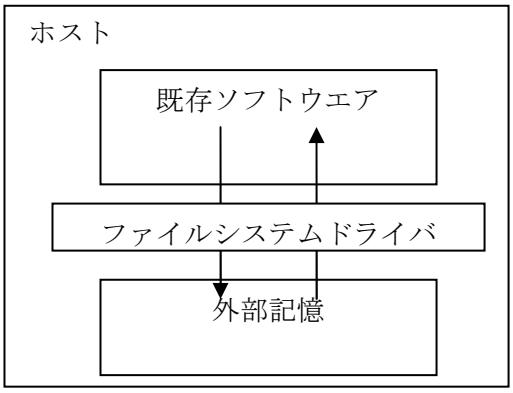

図 1 シングルホストソフトウエア

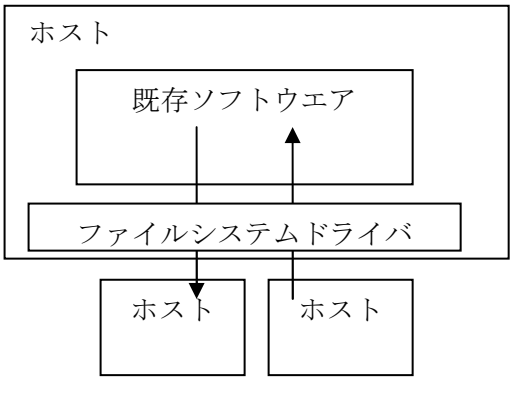

図 2 ファイルシステムを利用したマルチホスト化

† 筑波大学大学院 システム情報工学研究科 コンピュータサイエンス専攻
‡ 筑波大学 第三学群 情報学類

ファイルシステムを利用する方法がある。

シングルホストソフトウエアは、図 1 のように自ホストが持つ外部記録にファイルシステムを介してアクセスを行う。ファイルシステムを利用する方法では、図 2 のようにファイルシステムへのアクセスを他のホストへのアクセスに変換することで、他のホストとの連携を行う。この方法では、既存のソフトウエアに対して他のホストとの連携を行う部分を隠蔽する事で、既存のソフトウエアはそのまま利用でき、かつ、他のホストとの連携を可能にしている。

このような方法を利用している物に、ファイルを自動的に同期する Microsoft 社の Live Messenger[1] のファイル共有機能や、Google 社のメールサービスへのアクセスを行う GmailFS[2]、Blog への投稿を既存のエディタで行える BloggerFS[3]などがあげられる。

そこで、このようなファイルシステムを手軽に実装するために、FUSE[4]が提案されている。

一般的な OS が準備しているファイルシステムのための仕組みではファイルシステムドライバはカーネル内で実行される。そのため、ファイルシステムを実装するためにはカーネルモードでプログラミングを行う必要がある。また、他の OS との通信などを行うという複雑なソフトウエアをカーネルモードで行わなければならない。しかし、カーネルモード用の開発ツールはユーザーモード用の物に比べて少なく、限定された作業しか行えない。

FUSE を使用したファイルシステムの概観を図 3 に示す。FUSE では、カーネルモードに FUSE のファイルシステムドライバを入れる、しかし FUSE のファイルシステムドライバは実際のファイルシステムの仕事はせず、ユーザーモードで実装された FUSE 用ファイルシステムに呼び出しを転送する。これによってユーザーモードでファイルシステム本体を記述する事ができる。

しかし、ファイルシステムドライバは各 OS ごとに作成する必要がある。OS ごとにファイルシステムの開発方法は異なるため、ある OS 用にファイルシステムを作成しても他の OS ではそのファイルシステムは利用できない事が一般的である。

しかし、FUSE は Linux など数種類の OS しかサポートしておらず、UNIX 系 OS 以外はサポートしていない。また、UNIX 系 OS でも同一コードで動作する保証はない。

もし、多数の OS で利用できるファイルシステムが必要である場合、各 OS 用にファイルシステムを作りなおす必要がある。

本研究では、多 OS に対応したユーザーモードファイルシステムフレームワーク UniversalFUSE を提案する。

UniversalFUSE は、ファイルシステムを開発す

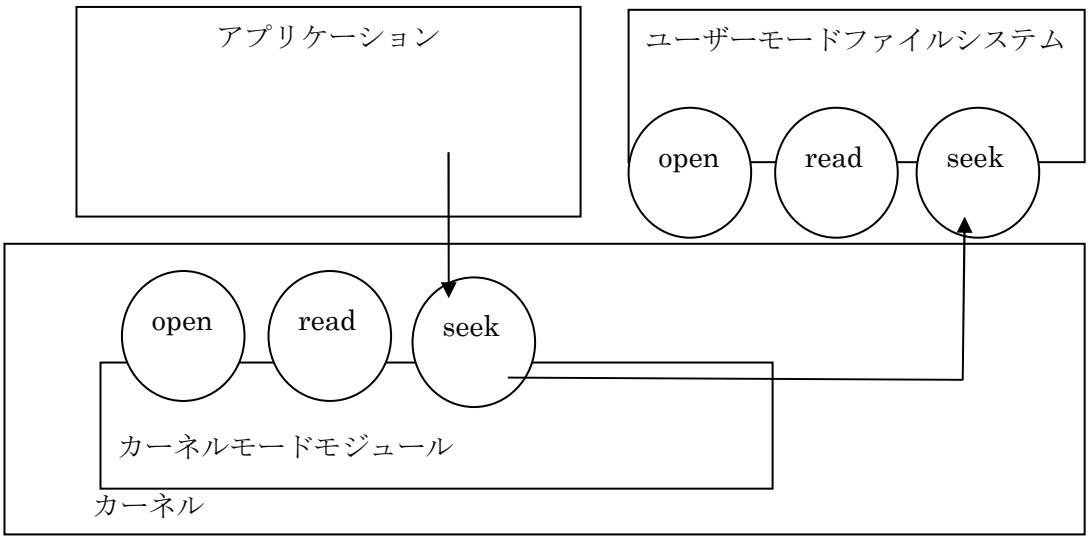

図 3 FUSE によるファイルシステムのユーザーモード化

る手間を削減するために、ファイルシステムをユーザーモードで開発し、かつ、OS に依存しない形でファイルシステムを開発するためのフレームワークである。

そのために、UniversalFUSE では、FUSE と同じようにカーネルモードからユーザーモード呼び出しを転送する、さらに呼び出しを OS に依存しない形にする事で各 OS に対応させる。

本研究では、UniversalFUSE を Windows と Linux 用に実装し、実験を行った。

2. 提案システムのアプローチ

前章で述べた通り、提案システムでは、ユーザーモードでファイルシステムを開発できるようにし、かつファイルシステムを OS に依存しない形で開発できるようにする。UniversalFUSE では次のようなアプローチでこの課題を解決する。

2.1 ユーザーモードファイルシステム

ファイルシステムは一般的には、カーネルモードで実装を行わなければならない。しかし、カーネルモードでの実装は、通常のユーザーモードでのプログラミングと違うところが多々ある。まず、よく使用されているほとんど使っているライブラリが使えない事があげられる。既存のライブラリはユーザーモードでの使用を前提としている物がほとんどであるため、カーネルモードで利用する事ができない。そのため、カーネルモードで開発を行う場合、これらのライブラリを使えず、使用するライブラリはカーネルモードでも動作するライブラリに制限される。また、カーネルモードでのプログラミングでは、デバッグ環境が制限され、通常使っているデバッガなどの開発支援ツールを使う事ができない。また、カーネルモードは、一般的に CPU の特権モードで動作しているため、プログラムのミスがシステム全体のクラッシュを呼ぶ原因になる。システムがクラッシュしてしまった場合、その原因を調べる事は難しい。また、このような環境で開発を行うためには、仮想マシンの中で開発を行うなどの工夫を行う必要がある。

このように、カーネルモードでの開発は沢山の手間がかかる。そこで、提案システムでは、カーネルモードではなく、ユーザーモードでファイルシステムを実装する。

しかし、一般的な OS では、ファイルシステムはカーネルモードで実装しなければならない。そこで、FUSE を利用したファイルシステムでは、カーネル

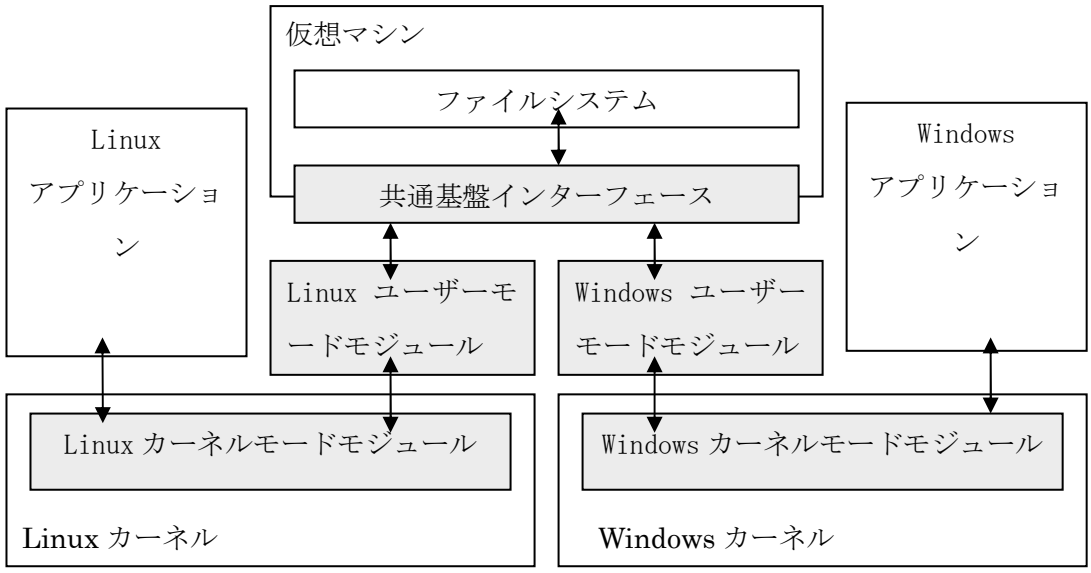

図 4 提案システムの概観

内にカーネルモードで動作するスタブを用意し、ファイルシステムの代わりにカーネル内で動作させる。そして、OS からの読み書き要求をユーザーモードに転送し、ユーザーモードで実際の処理を行う。提案システムでもこの手法を利用する。

2.2 ファイルシステム共通化基盤

一般的にファイルシステムは、OS ごとに作りなおす必要がある。これは OS ごとにファイルシステムのセマンティックが違うためであるが、それ以前にインターフェースが OS ごとに違うためである。

OS によってファイルシステムの詳細は大き異なる、しかし、大多数の OS では、ファイルシステムは木構造になっており、入れ子になっているディレクトリにファイルが入っている構造をしている。また、各ファイルは先頭から順に、またランダムな位置から読み書きなどのアクセスする事がきる。このような OS を越えてファイルシステムに共通な機能が多く、また、実際の使用に関しては、これらの機能をサポートすれば十分である。

しかし、サポートする機能は同じでもファイルシステムをカーネルがどのように呼び出すかといった規約や関数の引数などは OS 毎に異なっており、互換性はない。

提案システムでは、これらの OS 間の差異を吸収する共通基盤ライブラリを作成する。共通基盤ライブラリは、単一の UniversalFUSE API を提供する。ファイルシステム作成者は、この UniversalFUSE API を利用してファイルシステムを開発する。この UniversalFUSE API を UniversalFUSE がサポートするどの OS でも同じ API にする事で、各 OS で同じファイルシステムを使用する事ができる。

3. 設計

本章では、UniversalFUSE の設計について述べる。

3.1 概観

UniversalFUSE の概観を図 4 で示す。

UniversalFUSE は大きく分けてユーザーモードモジュールとカーネルモードモジュールからなる。さらにユーザーモードモジュールは、ユーザーモードライブラリと共通基盤ライブラリからなる。さらにユーザーモードの共通基盤ライブラリは、提案システムを使用して作成されたファイルシステム本体が接続されている。

カーネルモードモジュールはカーネル内で動作している。ユーザーモードモジュールはすべてユーザーモードで動作している。

これらのモジュールはほぼ全てが、各 OS が動作しているアーキテクチャのネイティブコードで動作しているが、ファイルシステム本体は、各 OS で共通のバイナリが動作するようにするため、Java などの仮想マシン上で動作する必要がある。

3.2 カーネルモードモジュール

カーネルモードモジュールは各 OS のカーネル内で動作するモジュールである。各 OS カーネルは、このモジュールをファイルシステム本体であると認識し、このモジュールに対してファイルシステム呼び出しを発行する。カーネルモジュールは、受け取った呼び出しを実行せず、ユーザーモードライブラリに転送する。

カーネルモードモジュールがユーザーモードライブラリに呼び出しを転送する場合、単純な関数呼び出しなどでこれを行う事はできない。カーネルモードモジュールはカーネルモードで動作しており、ユーザーモードで動作するユーザーモードライブラリとは動作モードが違うためである。そのため、カーネルモードモジュールはブロック型デバイスインターフェースや名前付きパイプなどを利用して、ユーザーモードと通信を行う。この時、ブロック型デバイスインターフェースや名前付きパイプは単純な通信路であるため、呼び出しをシリアライズして送る必要がある。また、カーネル内の処理を邪魔しないように、キューイングして送るなどの処理が必要である。

3.3 ユーザーモードライブラリ

カーネルモードライブラリでは、ブロック型デバイスインターフェースや名前付きパイプなどを使ってユーザーモードへ呼び出しを伝えている。その呼び出しを受け取るのがユーザーモードライブラリである。ユーザーモードライブラリでは、カーネルモードモジュールが関数呼び出しを通信路に入れるためにシリアライズした物を、再度関数呼び出しにデシリアライズする処理を行う必要がある。こ

れらの処理は使用するOSによって異なっているため、各 OS 用に実装を行う必要がある。

3.4 共通基盤ライブラリ

カーネルモードモジュールやユーザーモードライブラリは、各 OS ごとに異なるインターフェースを持っている。これらのインターフェースを統一するが共通基盤ライブラリである。

OS ごとにファイルシステムの持つ機能が違うため、各 OS の持つインターフェースを全て実装する事は難しい、しかし、一般的なファイルの読み書きは様々な OS でほぼ同じであるため、共通化する事ができる。また、ファイルシステムで良く使われる機能はこれらだけであるため、共通化できる部分だけでも十分実用的であると考えられる。

また、パス表記の変換や、ファイル名の文字コードを統一するなども、このライブラリで行う。

3.5 仮想機械

本提案システムでは、ファイルシステム本体は対応する各 OS で同じ物が動作するようにしたい。そこで、提案システムでは、ファイルシステム本体を仮想マシン内で動作させる事で、OS や CPU アーキテクチャに関係無く動作させる。

今回は多数のプラットフォームで同じバイナリを実行できるようにするために、Java を利用している。

3.6 ファイルシステム本体

ファイルシステム本体は、提案システムを用いて実装される。ファイルシステムはユーザーモードで実行されるため、既存のライブラリおよび開発手法を用いて効率的な開発を行う事が可能である。また、ファイルシステム本体は一度プログラミングを行えば、後は UniversalFUSE がサポートする OS で同じコードを実行する事が可能である。

4. 実装

次に具体的な提案システムの実装について述べる。

4.1 Windows カーネルモードモジュールの実装

Microsoft 社の Windows オペレーティングシステムでは、ファイルシステムドライバをインストールする事で、使用するファイルシステムを追加する事が可能である。提案システムでは、Windows 用のファイルシステムとして、UniversalFUSE を利用して書かれたファイルシステムを利用可能にするために、Windows 用のファイルシステムドライバと、ユーザーモードライブラリを実装した。

Windows カーネルへのドライバのインストールは、ドライバオブジェクトと呼ばれる構造体を用いて行う。ドライバオブジェクトは巨大な関数ポインタの集りであり、このドライバオブジェクトに自分が実装したいドライバが利用する関数群を入れカーネルに登録する事で、ドライバが利用されるようになる。今回は、ファイルシステムを実装するために、CREATE、CLOSE、READ、WRITE などのシステムコールを入れて登録を行った。

カーネルモードからユーザーモードへの通信は DeviceIoControl を用いた。この DeviceIoControl は、ドライバとユーザーモードの相互通信を行うための WindowsAPI(システムコール)である。

本来、DeviceIoControl は、カーネルモードのドライバから情報を抜き出したり、カーネルモードドライバのコントロールを行ったりするために存在しているが、提案手法では逆に、カーネルモードモジュールがユーザーモードファイルシステムに対して情報を要求するために使用されている。

提案システムでは、カーネルから要求を受けたドライバはその要求をユーザーモードに届け、返事を待ち、返事をカーネルに向けて返す。しかし、この間、要求元のスレッドに対してブロックを行うとシステム全体のパフォーマンスを下げる事になりかねない。そのため、呼び出された提案システムのカーネルモジュールは一度呼び出し元に対して、ペンディングステータスを返す。そして、ユーザーモードから来た返事の順にこれらの呼び出しにさらに通史を行う。

4.2 Windows 用ユーザーモードライブラリ

Windows 用のユーザーモードライブラリは、カーネルモードから DeviceIoControl で受けとる。しかし、DeviceIoControl は、カーネル内のデバイスを操作するための仕組みであり、カーネルからデータを受け取る事は簡単にはできない。本システムでは、常にユーザーモードライブラリが DeiceIoControl を呼び出すようにしておき、カーネ

ルモードモジュールはこれをブロックしておく。カーネルモードモジュールがユーザーモードへ通信を行いたい時は、このブロックを解除しデータを返す。こうする事によって必要に応じてカーネルモードからユーザーモードへデータを送信できる。

4.3 Linux カーネルモードモジュールとユーザーモードモジュールの実装

Linux オペレーティングシステムでも UniversalFUSE を実装した。Linux の場合、FUSE 用の既存のカーネルモードモジュール及び、ユーザーモードモジュールが存在しているため、これを利用して実装を行った。

UniversalFUSE を FUSE 上で実装する事で、カーネルモードモジュールを開発する手間が省けると共に、FreeBSD などの他の UNIX 系 OS でも FUSE が移植されているシステムでは、UniversalFUSE を簡単な手間で使用する事が可能にできる。

4.4 共通基盤ライブラリの実装

共通基盤ライブラリは各 OS の差を吸収するためのライブラリである。各 OS で差はあるものの、大体の OS は同じようなファイルアクセスのための手続を持っているため、引数の形を整える程度の変更で済む。

一方、ファイルシステムのセマンティックは各 OS により異なり単純な方法では解決できない物もある。

簡単な例として、ファイルシステムのディレクトリ階層の区切り文字がある。Linux など UNIX 系 OS では、"/"(スラッシュ)がディレクトリの区切り文字として利用されている。一方、MS-DOS からの流れがある Windows オペレーティングシステムで

UniversalFUSE のファイルシステムが実装する関数一覧

OPEN	ファイルを開く
CREATE	ファイルを作成する
CLOSE	ファイルを閉じる
READ	ファイルから読み込む
WRITE	ファイルに書き込む
OPENDIR	ディレクトリを開く
CREATDIR	ディレクトリを作る
CLOSEDIR	ディレクトリを閉じる
READDIR	ディレクトリ内のファイルを調べる
UNLINK	ファイルを削除する
RMDIR	ディレクトリを作成する
STAT	ファイルの状態を調べる
STATFS	ファイルシステムの状態を調べる

表 1 関数一覧

は、区切り文字として”¥”（バックスラッシュ）が利用されている。また、本論文では対応としてないがマッキントッシュでは”:”（コロン）が区切り文字として使用されている。共通基盤ライブラリはこれらの区切り文字を統一する必要がある。本提案では、区切り文字として”/”を利用する。

さらに問題になる物として、ファイルシステムの文字コードの問題がある。特に日本語の場合は、多数の文字コードが存在しているため、どの文字コードを利用しているかは各コンピュータによってまちまちである。提案手法では、文字コードはUTF-16で統一する物とし、それ以外の文字コードを利用するシステムの場合は文字コードをコンバートする物とする。

4.5 仮想機械

提案システムは、各OS上で同一コードを動作させるためにJava VMを利用し、ファイルシステムはJava VM内で実行させる事としている。そのため、共通基盤ライブラリとJavaで書かれたファイルシステムの相互接続を行う必要がある。

4.6 ファイルシステム本体

提案システムを利用して実装したファイルシステムはいずれかのOSのインターフェースを持つわけではなく、UniversalFUSEのAPIを利用してファイルシステムを開発する。そのため、多数のOSで簡単に使用する事が可能になっている。

UniversalFUSEが用意しているファイルシステムインターフェースは図4に示す。

5. 実験

本提案システムの有用性を確認するために、速度比較を行った。

Windows用のNULデバイスと等価なファイルを持つファイルシステムをUniversalFUSEを用いて作成した。これを用いて速度を調べる。

ベンチマークは富士通社製のサーバーで行い、CPUはIntel Xeon 3.6GHz、メモリ2GBの環境で行った。またOSとしてはWindowsServer2003を使用した。

5.1 ベンチマーク

ベンチマークでは、指定したブロックサイズで、10MBのデータを書き込んだ。また、ハードディスクなどの速度による影響を受けないように読み込んだデータはディスクに書かない。さらにWindowsに標準でありディスクへ書き込まないNULデバイスへの書き込みを行った。

比較結果を図5に示す。これによると、ほぼファイルシステムの呼び出し回数に応じた時間がかかっている。この時間は今後最適化によって減らす事が考えられる。

6. 関連研究

関連研究としては、FUSEがあげられる。FUSEはファイルシステムドライバのユーザーモード化を行うライブラリである。しかし、FUSEはUNIX系ファイルシステムに特化した実装を行っており、他のOSに対応する事は難しい。

また、Windows用の物として、FIFS[5]があげられる。FIFSはWindows用のユーザーモード化ファイルシステムであるが、Windowsに既に存在しているSMBクライアント機能を利用して開発されている。

7. まとめ

本論文では、多OSに対応したユーザーモードファイルシステムフレームワークであるUniversalFUSEを提案した。UniversalFUSEはファイルシステムを多数のOSに対応させる事を可能にする事ができ、さらに、ユーザーモードで実装できるようにしている。本論文では、この

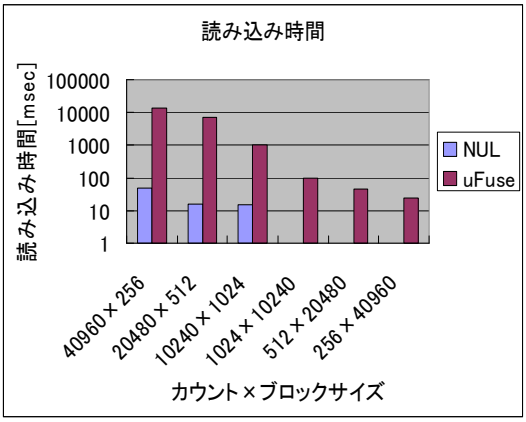

図 5 データの書き込みと時間

UniversalFUSE の設計と実装について述べ、さらに提案システムの有用性を確認した。

8. 今後の課題

今後の課題として、まず、呼び出しの高速化があげられる。

また、アクセスコントロールなどは、OS によって異なっているため、現在の UniversalFUSE では共通化していない。これを今後抽象化し共通化する事が考えられる。

また、現在は、Read Write などの関数そのものである。今後これを抽象化し、使いやすい形にすることを行いたい。

さらに、現在の実装は Linux および Windows に対応しているが、今後さらに別な OS にも対応する事が考えられる。

9. 謝辞

本研究の一部は、魅力ある大学院教育イニシアティブ「実践 IT 力を備えた高度情報学人材育成プログラム」による。

参 考 文 献

---

[1] Microsoft Live! Messenger
http://messenger.live.jp/

[2] GmailFS
http://richard.jones.name/google-hacks/gmail-filesystem/gmail-filesystem.html

[3] BloggerFS
http://mundau.blogspot.com/2006/12/bloggerfs-last-night-i-was-playing.html

[4] FUSE: Filesystem in Userspace
http://fuse.sourceforge.net/

[5] FIFS: A Framework for Implementing User-Mode File Systems in Windows NT
http://www.usenix.org/publications/library/proceedings/usenix-nt99/almeida.html