意識しないで自然に使えるデータ管理システム Decas

Decas: A User-friendly Data Management System

荒川 淳平* 浅川浩紀*
Jumpei Arakawa Hiroki Asakawa

概要

「あの時バックアップを取っておけば...」という経験をした人は少なくないだろう．データ管理の技術自体はバックアップ，バージョン管理，暗号化などすでに確立している．しかし，それが「めんどくさい」ために実行されなければ意味をなさない．そこで我々は，新しいコンセプトのデータ管理システム Decas を開発した．Decas では OS 上に仮想ドライブを実現し，その仮想ドライブ上での利用者によるファイルやディレクトリへの操作をフックし，自動的に一連のデータ管理処理を行う．これにより，利用者は特別な操作はもちろん，意識すらする必要がない．我々は，仮想ドライブによるデータ管理を実現するため，カーネル空間で動作する Windows 用仮想ファイルシステムドライバ Dokan とその上のユーザ空間で動作するデータ管理システム Decas を開発した．Dokan により独自のファイルシステムが定義でき，Decas はデータ管理機能を持つ複数のモジュールを組み合わせて一つのファイルシステムを構築することができる．つまり，Decas は優れた開発フレームワークにもなっている．

1 はじめに

情報化社会といわれる現代は，未だかつてないほどに多種多様な情報が溢れている時代といってよいであろう．情報化社会における狭義での「情報」とはいわゆる電子データを指す．実際，文章にはじまり画像，音楽，映像などのコンテンツがデジタル化され，電子データとして扱われるようになった．それらのデータを扱うコンピュータプログラムもまた電子データの一つである．そして，今この瞬間も，インターネットに代表されるコンピュータネットワークを通じて，膨大な量のデータが世界中を飛び交っている．それらのデータは私たちの社会の所産であり，価値をもつ大切なものである．まさに，現代社会におけるデータは「財産」といってよいであろう．しかし，その大切な「財産」は現代社会においてきちんと管理されているのであろうか．データ社会の現在には，多くのリスクが存在している．装置の故障によるデータの損失や誤った操作やウィルスなどによるデータの削除・上書き，さらにはコンピュータの盗難やトロイウィルスやファイル交換ソフトを悪用したデータの盗聴・流出などである．しかしながら当然，そのようなリスクに対しては，技術的な対策は用意されている．

- データの対故障性を実現するバックアップ
- データの復元性を実現するバージョン管理
- データの機密性を実現する暗号化

などが代表的である．そして，これらは大規模な商用のソリューションから数 KB の無料のプログラムまで，さまざまな形で私たちに提供されている．

にもかかわらず，私たちの日常生活の中で，データ管理がきちんとされているとは言い難い．ここで，「私たちの日常生活の中で」としたのは，

* 東京大学大学院 情報理工学系研究科
Graduate Scrool of Information Science and Technology, The University of Tokyo

一部の大きな組織などでは徹底したデータ管理がされているからである．販売記録や顧客情報，デジタルコンテンツなど，組織にとって大変重要なデータは，程度の差はあれ，きちんと管理されている[*1]．しかし，問題の本質は「私たちの日常生活の中」にあると考える．なぜならば，組織が管理するに至るデータも，最終的に利用するのは個人であり，そのデータも元をたどれば，私たちひとり一人が生み出しているデータだからである．

「めんどくさい」問題

では，なぜ「データ管理がきちんとなされない」のか．それは「めんどくさい」からに他ならない．ただこれは「私たちの怠惰のせい」という単純な意味ではない．この「めんどくさい」には現状のデータ管理における潜在的な問題が含まれていると考える．ひとつは，データ管理を「意識して」「実行する」必要があることである．私たちは一日に数え切れないほどたくさんのデータを扱う．もし，そのたびにいちいちデータ管理を意識しなければならないとしたら，それは大変なコストで，生産性を著しく低下させるであろう．また，データ管理を人手で明示的に「実行する」ことは，人為的なミスの可能性を常に孕んでおり，このリスクも無視できない．もうひとつは，様々な種類のデータ管理が独立に行われていることである．データ管理の機能の概念はそれぞれ独立したものであるが，利用する際には，組み合わせて利用されるものである（例えば，暗号化したものをバックアップする）．しかし，現状では別々のシステムを別々に利用する必要があり，乗算的に「意識」と「実行」の頻度を増加させてしまっている．私たちは，こういったコストやリスクをおそらく直感的に理解しているので，「面倒なことになりそうだ」と感じ，結果としてデータ管理を行うことを敬遠してしまうのである．

2　目的とシステム概要

そこで我々は，この「めんどくさい」問題を解決するために，利用者が意識することなく，一連のデータ管理を自動的に行い，自然な形でその恩恵を享受できるようなシステム Decas の提案と開発を行った．Decas は，「私たちの日常生活の中」で使われるようにするため，高価で複雑なソリューションではなく，簡単に導入できる普段使いのソフトウェアとしての実現を目指した．

Decas は動作するオペレーティングシステム上に仮想ドライブ[*2]を作成する．そして，そのドライブ上での利用者のファイルやディレクトリへの操作（保存や削除など）をフックし，自動的に一連のデータ管理処理（バックアップ，バージョン管理，暗号化）を行うものである（**図 1**）．

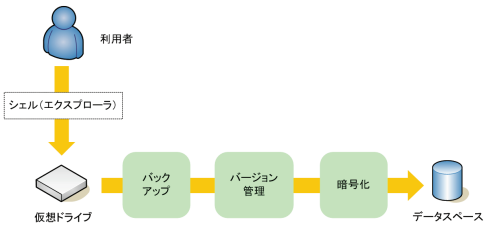

図 1　Decas の利用イメージ

ここで重要なことは，利用者はこれらの処理が実行されることを意識しなくてよく，ただ普通に，通常の物理ドライブを使う場合と変わりなく利用すればよい点である．また，複数のデータ管理の処理を組み合わせて一連の流れで実行している点も重要である．この 2 点により，データ管理を意識するコストとそれを実行するコストとリスクをなくすことができ，「めんどくさい」を解消することができる．

Decas は，以下の**図 2** のようなアーキテクチャを持つ．我々が実装した部分は FUSE 部分を除

[*1] と信じたいが，データ流出などで大企業や国の省庁などの名がニュースを賑わせることも珍しくない．

[*2] Linux では仮想ディレクトリとなる．

く全体である．システム全体としては，Windows においては後述する Dokan，Linux においては FUSE[4] を使うことで複数の OS を跨いで機能を提供することが可能になっている[*3]．以下で個々の開発内容について述べる．

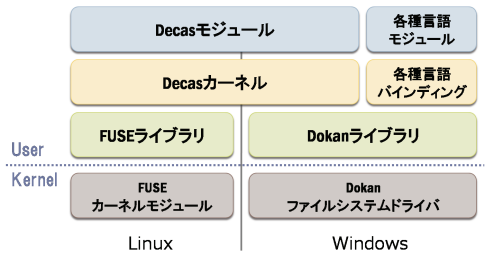

図 2　ソフトウェアの全体アーキテクチャ

3　仮想ファイルシステムライブラリ Dokan

Dokan は Windows のカーネル内部で動作する仮想ファイルシステムドライバとユーザ空間で動作するライブラリからなる，Windows 上で仮想ドライブを実現する仕組みである．Dokan を使うことで，仮想ドライブから利用できるユーザ定義のファイルシステムを実現することができる．

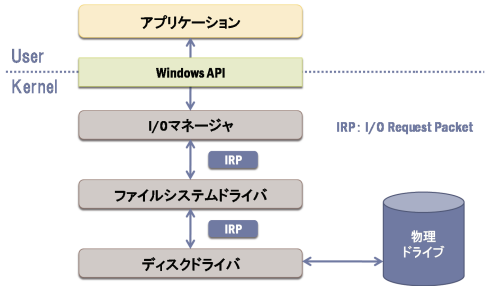

図 3　Windows ファイルシステムのアーキテクチャ

通常，Windows のファイルシステムはカーネル内部では IRP（I/O Request Packet）と呼ぶパケットを単位として I/O マネージャやファイルシステムドライバ，ディスクドライバがやり取りを行い，最終的に物理ドライブに格納される形で実現されている（**図 3**）．

一方，Dokan では，ファイルシステムドライバのところで IRP を一時的に中断させ，その処理を，ライブラリを通じて，ユーザ空間で実装されているファイルシステムに委譲する．これにより，NTFS や FAT32 など Windows が標準的に提供しているファイルシステム以外の独自機能を持ったファイルシステムをユーザ空間，つまり通常のプログラムの形で実現できる．また，ファイルシステムドライバに加えて仮想のファイルディスクドライバも作成した．仮想のファイルディスクドライバが仮想ファイルシステムをマウントしているかのように見せかけることで，正常な動作が行えるようになっている．

次に，Dokan ファイルシステムと，Dokan ライブラリにおいて，アプリケーションからユーザモードで動作しているファイルシステムを呼び出す流れを説明する（**図 4**）．

Windows 上で動作しているアプリケーションがファイルを開くときには，WindowsAPI である `CreateFile` を呼び出す (1)．`CreateFile` は Windows のカーネルレベルでは `NtCreateFile` 呼び出しとなる．Windows カーネルは，`NtCreateFile` を実行するために，IRP の `IPR_MJ_CREATE` を生成する (2)．`IRP_MJ_CREATE` は，I/O マネージャにより Dokan ファイルシステムドライバに渡される (3)．ファイルシステムドライバは，その IRP をいったん中断させ，ユーザモードで動作している Dokan ライブラリに知らせる (4)．Dokan ライブラリは，ユーザモードファイルシステムが実装している `CreateFile` を呼び出す (5)．Dokan ライブラリは，ユーザモードファイルシステムの `CreateFile` の実行結果を受け取り (6)，Dokan ファイルシステムドライバに渡す (7)．Dokan ファイルシステムドライバは，先ほど中断していた `IRP_MJ_CREATE` を完了させ，I/O マネージャに

[*3] Mac OS-X では FUSE の移植版が利用できる．

知らせる (8)．I/O マネージャは NtCreateFile を完了させ (9)，最終的にユーザモードアプリケーションが実行した CreateFile が完了する (10)．

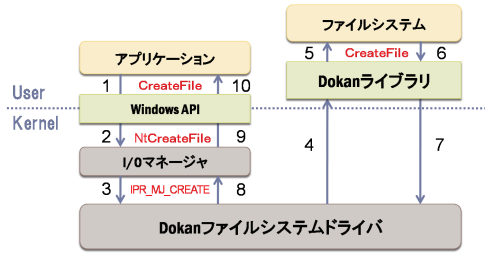

図 4　Dokan とユーザモードファイルシステムの動作の流れ

ユーザモードファイルシステムを実装するには，ユーザモードアプリケーションが呼び出すファイル操作 API に対応する，ディスパッチルーチンを実装すればよい．先の例では，Windows-API の CreateFile に対して，CreateFile を実装する．そのほかに，ReadFile，MoveFile，LockFile 等，現在の Dokan ライブラリでは，22 種類のディスパッチルーチンが定義されている．

4　フレームワークとしての Decas

Decas はデータ管理システムであると同時に，Dokan（と FUSE）のファイルシステムアプリケーションとして振舞うための，開発フレームワークでもある．Decas は OS 間の差異を吸収して，OS 非依存のデータ管理機能を実現する基盤を提供する．図 5 に Decas のアーキテクチャを示す．

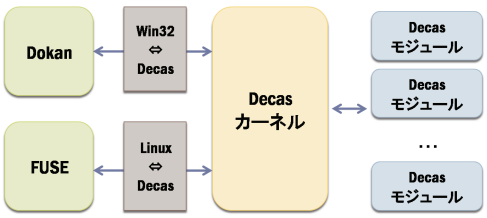

図 5　Decas のアーキテクチャ

Decas はデータ管理機能をモノリシックな構造ではなく，個別の独立した機能をモジュールとして別々に実装し，それらを重ね合わせて，一つのファイルシステムとして提供することを可能にしている．つまり複数のモジュールがデータをやりとりして，全体として統合された一つの機能を提供している．

このような連携が必要な場合，各モジュールをレイヤーに見立てて，モジュール間のやりとりに明示的なインターフェイスを定義し，モジュールの入出力をそれに従わせることでモジュールを実装するのが一般的である（**図 6**）．

図 6　一般的なモジュール間の関係

これに対して Decas では，フレームワークの実装としてはより複雑であるが，仮想ドライブを経由してモジュールが連携する仕組みを考案し実現した（**図 7**）．この方式では，モジュールの入出力には決まったインターフェイスは定義せず，自由な入出力を許す．その代わりに，モジュールに対するリクエストの段階でパスにコンテキスト情報（モジュールやセッションの表す識別子）を埋め込んでおく．そして，そのリクエストのパスによって生じるモジュールの入出力を仮想ドライブ（より正確には仮想ドライブ経由で Decas カーネル）で検出し，コンテキスト情報を解析し，応答すべきモジュールに処理を委譲している．

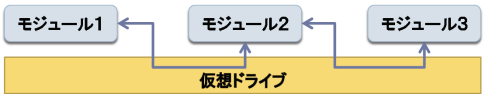

図 7　Decas におけるモジュール間の関係

この仕組みの最大のメリットは，既存システムの利用にある．ここでいう既存システムとは様々なデータ管理機能を提供している既存のプ

ログラムやソフトウェアのことである．それらのシステムは，当然ではあるが，ファイルシステムとしてではなく，あくまでファイルシステムを前提に普通のプログラムとして作られている．したがって，明示的なインターフェイスを規定する方式では，既存システムの入出力を行うすべての箇所について，そのインターフェイスに沿う形に書き換えなければならない．これはほとんどシステムを最初から書き直すのと同じぐらいコストのかかる行為である．

これに対して，Decas で採用した方式では，こうした書き換えを一切行う必要がなく，既存システムの機能をほぼそのまま利用することができる．現在，数えきれないほどのデータ管理を行うプログラム／システムがあることを考えるとこのメリットは計り知れない．また，このモジュールの実装構造は Apache で採用されている DSO（Dynamic Shared Object）方式を採用することで，開発の容易性とポータビリティを両立している．さらに，Decas カーネルは標準的なファイルシステム処理などの必要な機能をモジュールに提供している．これにより，OS に依存しないデータ管理モジュールを低コストで実現することができる．

5　アプリケーション

以下で Decas や Dokan を用いて我々が開発したアプリケーションを紹介する．

バージョン管理モジュール

バージョン管理モジュールは，データの変更や削除によって生じる差分を管理することで，データをいつでも過去の任意の状態に復元する機能を提供する．また，データの複製をブランチ（バージョン管理上の分岐処理）として扱うことで，効率のよいデータ保存も可能にした．データを過去の状態に戻す機能などはシェル拡張を用いることで，利用者に違和感なく自然な形でバージョン管理の恩恵を受けられるようにしている．この「元に戻す」機能はファイルだけでなくディレクトリに対しても有効で，誤ってファイルを削除してしまった場合などはディレクトリを過去の状態に戻すことで，ファイルを復元できる．また当初は「元に戻す」機能のみが考えていたが，「戻す」のではなく単純に過去の状態を「見る」機能も追加した．これにより，「元に戻す」作業を手軽に「試す」ことや過去データの取得や現状データとの比較が容易になった（**図 8**）．

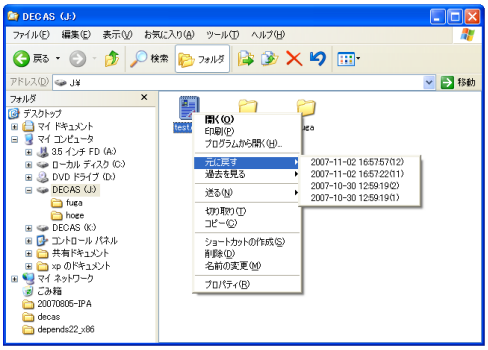

図 8　バージョン管理モジュールの実行イメージ

バックアップモジュール

バックアップモジュールは、データの保存時に指定された複数のスペースにデータを複製して保存することで対故障性を実現する．また，このモジュールでは，複数ある保存スペースのいくつかが利用不可能になった場合でも，利用可能なスペースのみを使うことでそのまま利用すること（故障に対する透明性）を実現にしている．このことにより，ディスク破損などによって利用者は作業を中断されることを防ぎ，意識せずともバックアップの恩恵を享受できる．

暗号化モジュール

暗号化モジュールは，データの保存時にデータスペースに暗号化を施してデータを保存し，逆にデータの利用時には復号化を施すことで，データの機密性を提供する．保存時にデータが暗号化されていることで，データスペースを含む機器（たとえばノート PC や USB メモリ）が盗難に遭い，悪意ある第三者がデータを盗み出そうと

```
require 'dokanfs'
class Hello
  def initialize
   @msg = "hello, world"
  end
  def contents path
    ["hello.txt"]
  end
  def file? path
    path  =~ /hello.txt/
  end
  def directory? path
    path == "/"
  end
  def read_file path
    @msg
  end
  def size path
    @msg.length
  end
end
DokanFS.set_root(Hello.new)
DokanFS.mount_under("r")
DokanFS.run
```

図 9　Dokan Ruby バインディングによる HelloFS

しても、正しい情報を読み取ることができない．

Ruby バインディング

Dokan や Decas ライブラリを使用することで，ユーザモードで手軽にファイルシステムを実装することが可能になったが，より簡単にファイルシステムが実現できるように Dokan の Ruby バインディングを実装した．Ruby バインディングを使用することで，数十行のプログラムを記述するだけでファイルシステムが作成できる（**図 9**）．また，FUSE の Ruby バインディングである fusefs 互換インターフェイスを作成した．これにより，Ruby で書かれたプログラムを修正することなく Linux，Windows 両方で動作させることが可能である．

.NET バインディングと SSHFS

C#や VB.NET など.NET 環境で手軽にファイルシステムが作成できるように.NET バインディングを実装した．.NET Framework の豊富なライブラリ群が使用できるため，多機能なファイルシステムを簡単に作成できる．.NET バインディングを利用して，SSHFS を実装した（**図 10**）．SSHFS は SSH で接続できるサーバー上のファイルシステムをローカルのドライブとしてマウントするためのプログラムである．SSHFS を使用することで，たとえば Linux サーバー上のファイルを Windows のメモ帳などで直接編集することが可能になる．また，Samba の代わりとしても利用できる．

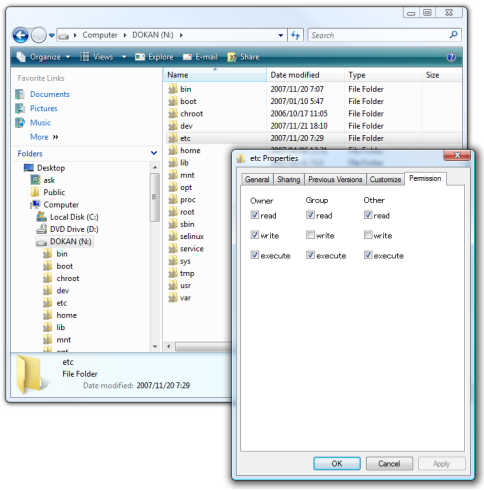

図 10　SSHFS の実行イメージ

6　考察

意識しない／自然な形での利用

Decas の最大の特徴は，システムを意識しない，もしくは非常に自然な形で利用できる点である．実際，仮想ドライブは通常のドライブと利用するにおいて何ら差異はなく，一般的な利用者には区別がつかない．また，明示的に機能を利用する場合（たとえば「元に戻す」機能の利用）においても，機能をシェルに統合した形で提供するため，極めて自然な形で利用できる．これにより利用者の本システムの利用にかかるストレスや学習コストを極めて小さくすることができたと考える．

```
int caeser_read(const char *path, char *buf, size_t size, off_t offset,
                struct decas_file_info *fi)
{
    int i;
    int res;

    res = delegate->read(path, buf, size, res_size, offset, fi);
    for (i = 0; i < res; i++)
        buf[i] += 3;

    return res;
}

int caeser_write(const char *path, const char *buf, size_t size, off_t offset,
                 struct decas_file_info *fi)
{
    int i;
    char *newbuf;

    newbuf = strndup(buf, size);
    for (i = 0; i < size; i++)
        newbuf[i] -= 3;

    return delegate->write(path, newbuf, size, offset, fi);
}
```

図 11　Decas でのシーザー暗号モジュールの定義

実装コストの低減

これまで Windows でファイルシステムを実装するには，Windows やデバイスドライバに関する非常に高度な知識が必要であった．このため，Linux 等のオープンソースの OS と比較して，Windows のファイルシステムの研究や実装は皆無に等しい．今回我々が開発した Dokan や Decas を利用することで，Windows のファイルシステムや，デバイスドライバに関する知識がなくても，比較的簡単にファイルシステムを作成することが可能になる．Decas や Dokan を利用して，Windows でも Linux のようにさまざまなファイルシステムが提案され，利用されることが期待できる．また，特定の処理にフォーカスを当てたデータ管理機能を実現する場合，Decas が標準的なファイルシステムの機能を提供するため，必要最小限の機能のみを実装すればよいので実装コストを更に低くすることができる．例えば，暗号化はデータの読み込みと書き込みのみに注目すればよい（**図 11**）．さらに，Decas が OS ごとの差異を吸収してくれるため開発者は機能のみに集中することができ，一つのモジュールを作るだけで複数の OS で機能を提供できる．

7　関連システム

機能としての関連

LFS[1] などのログ構造ファイルシステムやジャーナリングファイルシステムは，メタデータと呼ぶ管理情報を扱う点でバージョン管理モジュールと似ているが，その目的は，過去のデータの復元ではなく，ファイル書き込みについての無矛盾性を保証することである[*4]．また，ファイルシステムとして実装されているため，拡張や組み合わせが困難である．

[*4] 一部の実装はデータの復元機能を持つが，バージョン管理と異なり履歴は一定期間で破棄される．

darcs[2] などの分散バージョン管理システムは，バージョン管理に加えて分散化が行われている点で Decas に似ている．しかし，ここでの分散化とはリポジトリが利用者別に複製されているだけであり，Decas のような冗長性を目的としているものではなく，またその機能（例えば故障に対する透明性など）もない．

最近発売された Microsoft 社の Windows Vista や Apple 社の Mac OS-X（Leopard）にはそれぞれバージョン管理と似た機能が搭載されている[*5]．しかし，これらの機能はいわば頻度の高いバックアップであり，厳密なバージョン管理とは異なる．また，各 OS に組み込まれた機能であり，OS を跨いでの構築や利用はできない．

基盤としての関連

一部の BSD 系の OS で利用できる Stackable FS[3] を用いることで，Decas のように複数の機能を持ったファイルシステムを組み合せてひとつのファイルシステムとして利用することが可能である．しかし，ぞれぞれのファイルシステムはカーネル空間で実装しなければならず，ユーザ空間で通常のプログラムと同様に開発できる Decas に比べて開発が困難である．

FUSE[4] は Linux で動作する仮想ファイルシステムライブラリである．FUSE を用いることで，ユーザ空間でファイルシステムを作成することができる．この機能は Decas で Windows 上に仮想ファイルシステムを作成する Dokan に相応するものである．Decas では Linux においては構成要素の一部として FUSE を利用する．いずれの基盤も特定の OS に依存しているため他の OS で利用することはできない．

8 今後の予定

現在, Decas および Dokan は, Web サイト[*6]上で公開・開発が続けられている．現状で，バージョン管理，バックアップ，暗号化の 3 つのモジュールがあるが，その他の様々なモジュールの開発を計画している．また，Ruby と .NET 用に加えて新たな言語バインティングも作成し，より手軽にファイルシステムが作成可能になる環境を提供したい．また，現行のシステムは主に 1 台コンピュータを対象にしているモデルだが，今後は 1 つの保存領域に対して複数の仮想ドライブが接続する「ネットワークモデル」，さらには複数の保存領域に対して複数の仮想ドライブが接続する「分散モデル」へとシステムを発展させていき，より多くの利用状況に対応したい．

[*5] Vista ではシャドウコピー，Leopard では Time machine と呼ばれる．

[*6] http://decas-dev.net/

謝辞

Decas は IPA（情報処理推進機構）2006 年度下期未踏ソフトウェア（ユース）採択プロジェクト『データ管理システム』の開発成果です．プロジェクトマネージャの筧捷彦先生，プロジェクト管理企業のオープンテクノロジーズ様には大変お世話になりました．感謝申し上げます．

参考文献

[1] M. Rosenblum and J. K. Ousterhout. Design and Implementation of Log-Structured File System Proceedings of the 13th ACM SOSP, pp.26-52, 1992.

[2] Roundy David. Darcs: distributed version management in Haskell. Proceedings of the 2005 ACM SIGPLAN workshop on Haskell Tallinn, Estonia, 2005.

[3] John S. Heidemann and Gerald J. Popek. A layered approach to file system development. Technical Report CSD-910007, University of Carifornia, Los Angeles, 1991.

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