ファイル処理プログラムの開発を支援する仮想ディレクトリ機構

Virtual Directory Mechanism that supports File Manipulation Scripts Development

一杉 裕志†
Yuuji ICHISUGI

１．背景

従来のプログラミング言語よりも生産性の高いスクリプト言語として、 perl, python, ruby, groovy などが注目を浴びている。これらの言語は、文字列処理のための組み込み言語機能を持つなど、実用的な処理を行うプログラムが手早く簡単に書けるよう、言語仕様やライブラリに多くの工夫がある。

スクリプト言語の主要な用途の１つに、ファイル処理がある。例えば大量のファイルのフォーマットを一括で変換する、メールボックスの形式を変換する、古いファイルを他の場所に移動してアーカイブする、などの処理である。

一般にファイル処理プログラムの開発には、下記の困難が伴う。

- プログラムにバグがあると大事なファイルを破壊してしまう危険性がある。
- テスト実行の際に、前回のテスト実行の結果が残っていると、意図しない動作が起きてプログラマを悩ますこともある。

このようなことを避けるためには、開発時にはテスト用のフォルダ以外は操作しないこと、テスト実行のたびにテスト用のフォルダを初期状態に戻すことが必要である。 しかしこれは面倒な作業であるし、この作業自体のミスで大事なファイルを失う可能性もある。

以上のような本質的問題を解決しない限りは、スクリプト言語におけるファイル処理の利便性向上には限界がある。そこで本論文では、ファイル処理を行うプログラムが安全に開発できるように、言語のライブラリレベルで支援する**仮想ディレクトリ機構**を提案する。

２．基本アイデア

仮想ディレクトリ機構は、プログラムが操作しようとするディレクトリを自動的に別の作業領域にコピーする。（これを**仮想化**と呼ぶ。）すべてのファイル操作はそのコピーに対して行われる。プログラム開発の最中には、ユーザが明示的に指示を出さない限り、仮想化された領域以外は決して変更されない。

この機構は下記の利点を持つ。

- テスト用のディレクトリを用意する必要がない。本来の処理対象であるファイルシステムの中身そのものを使ってテストできる。
- 開発中にバグがあっても大事なファイルを破壊することがない。
- 起動のたびに仮想領域が初期化されるので、前回のテスト実行の結果が実行に影響しない。

仮想ディレクトリ機構はほぼ完全にプログラムから透過な形で提供される。すなわち、あたかも本当のファイルシステムを操作するかのようなプログラムを書けば、自動的に仮想ディレクトリ上で処理が行われる。この機構は実行環境の設定あるいはプログラムの初期化時の設定でオフにすることができるが、その際、プログラム本体の変更は全く必要ない。したがって、十分にデバッグがすんだ後、機構をオフにして本番の処理を行えばよい。また、仮想フォルダは実行時のオーバーヘッドを伴うが、それが問題になる場合は、小さいテストデータで十分にデバッグをすませた後、オフにして実行するようにすればよい。

† 産業技術総合研究所 脳神経情報研究部門

３．詳細設計における前提

本機構の詳細設計をするに当たり、以下のことを前提とした。

1. 移植性を考慮し、ＯＳの改変は行わず、言語ライブラリレベルで実装する。
2. 処理対象となるファイルの総容量は大きくないと仮定する。（ファイルシステム全体に渡って読み書きするような場合は対象としない。その場合はプログラマは仮想ディレクトリ機構をオフにして実行するものとする。）
3. 処理対象となるファイルには局所性があると仮定する。（少数のディレクトリ内にあるファイルだけを処理すると仮定する。）
4. プログラムからは、原則として仮想ディレクトリ機構が提供するファイル処理プリミティブだけを使ってファイル処理を行うものとする。
5. プログラムから外部コマンドを呼び出す場合や、仮想ディレクトリ機構と無関係に書かれたレガシーコードを利用する場合についても、ある程度考慮する。
6. ファイルおよびディレクトリのハードリンク・シンボリックリンクはないものと仮定する。
7. 並行動作する他のプロセスによってディレクトリが書き換わることはないと仮定する。

４．動作概要

プログラムがカレントディレクトリを移動すると、移動先のディレクトリの中身を再帰的に、**仮想領域**にコピーする。仮想領域は、ＯＳが提供する一時ファイル置き場の下に取る。例えば一時ファイル置き場を /tmp/vdm とすると、/a/b というディレクトリの仮想領域は、 /tmp/vdm/a/b である。

カレントディレクトリの２度目以降の移動に伴う仮想化には注意を要する。新たに仮想化しようとするディレクトリを D とすると、

1. D がすでに仮想化ずみであるか、すでに仮想化ずみのディレクトリの中に含まれるディレクトリであれば、中身のコピーは行わず、カレントディレクトリの変更だけを行う。

2. D がすでに仮想化ずみのディレクトリの先祖ディレクトリであった場合は、 D を仮想領域にコピーする際に、すでに仮想化ずみのディレクトリは上書きしないようにする。
3. いずれでもない場合は、単純に D の中身を仮想領域にコピーする。

例えば、カレントディレクトリを /a/b から /a に移動する場合は、 /a/c というディレクトリがあれば /tmp/vdm/a/c にコピーするが、 /a/b はすで仮想化ずみなので、 /tmp/vdm/a/b は上書きせずに残す。

以上の方針により、ファイルシステム中のディレクトリが必要に応じて整合性を保ったまま仮想領域にコピーされる。

仮想ディレクトリ機構は、カレントディレクトリ(Current Working Directory: **cwd**) のパス名を保持し、ユーザプログラムがカレントディレクトリを問い合わせた場合は cwd を返すが、内部的なファイル処理プリミティブは、cwd に対応する仮想領域のパス名(Virtualized cwd: **vcwd**) を用いる。

ファイル処理のプリミティブが絶対パスを指定して呼び出された場合は、そのパスが指すディレクトリも仮想化する必要がある。例えば moveFile(“a”, “/b/c”) というプリミティブが呼び出されたときは、 “/b/c” というディレクトリをまず仮想化し、ファイル a はその仮想化された領域に移動する。パス名に “..” が含まれている場合は、それを含まない絶対パスに変換して処理をする必要がある。

なお、仮想化アルゴリズムの詳細については付録の擬似コードを参照されたい。

５．外部コマンドおよびレガシーコード

プログラム中から外部コマンドを呼び出す場合は、あらかじめ起動されるコマンドのカレントディレクトリを vcwd に設定してから呼び出せばよい。外部コマンドが、相対パスを使って vcwd 内のファイルだけを操作する限りにおいては、仮想ディレクトリ機構のオン・オフに関わらず、プログラムは正しく動作する。

同様のことは、仮想ディレクトリ機構とは無関係に書かれたレガシーコードの利用についても当てはまる。プログラムがカレントディレクトリを変えるたびに、言語の実行環境のカレントディレクトリを vcwd に設定すればよい。レガシーコードが絶対パスを一切使わずにファイル処理を行うならば、仮想ディレクトリ機構が提供するファイル処理プリミティブを使っていなくても、正しく動作する。ただし、 Java 言語のように実行環境のカレントディレクトリを変更する標準的手段が提供されていない場合は、この方法は使えないので注意が必要である。

６．実装

本機構は、我々が開発したスクリプト言語チャミー[1]上に実装されている。

チャミーはプログラミングに不慣れなエンドユーザでも扱えることを目的としたスクリプト言語であり、仮想ディレクトリ機構はその目的に大きく貢献する。チャミーは他にも、日本語化された構造エディタを持つ統合開発環境、可視化された実行トレースの表示機構、実行時エラーリカバリ機構といった言語機能により、エンドユーザによる開発を支援している。

なお、チャミーのデモ版は [2] においてオープンソースで公開中である。

７．関連研究

危険を伴うソフトウエア開発を容易にするために、実行環境を仮想化することは、以前から行われている。従来の仮想化のレベルには、マシンレベル、カーネルレベル、ミドルウエアレベルなどがある。

VMware[3] などの仮想マシンを提供するソフトウエアは、ＰＣのハードウエアをエミュレーションすることにより、１台のマシン上で任意のＯＳを複数動かすことができるようにする。仮想マシンを用いることにより、ＯＳの開発や分散ソフトウエアの開発が容易になる。

User Mode Linux[4] はユーザモードで実行できるように改造された Linux のカーネルである。カーネル開発の大部分において、通常の Linux のアプリケーション開発と同じデバッグ環境を利用でき、開発効率が格段に向上する。

SoftwarePot[5] はミドルウエアレベルでシステムコールの処理を仮想化して Sandbox を実現するシステムである。任意のバイナリをファイルシステムの一部と共に他のホストに転送し、転送先で安全に実行することができる。

本研究は従来研究とは違い、言語ライブラリのレベルで、ファイル処理に的をしぼって仮想化するものであり、実装が極めて容易であるため既存の言語上に実装しやすく、移植性も高いという利点がある。

８．まとめ

ファイル処理プログラムの安全な開発を支援する仮想ディレクトリ機構について述べた。

本機構は既存のプログラム言語上のライブラリとして容易に実現可能である。また、外部コマンド呼び出しやレガシーコードの利用も、絶対パスを用いていなければ可能である。

謝辞

本機構のアルゴリズム策定と実装にあたっては、尾崎 竜史氏に協力していただきました。感謝いたします。

参考文献

[1] 一杉裕志、古川浩史：“エンドユーザ向けのスクリプト言語：チャミー”, 第 7 回プログラミングおよびプログラミング言語ワークショップ (PPL2005)

[2] チャミー, http://staff.aist.go.jp/y-ichisugi/chummy/

[3] VMware, http://www.vmware.com/

[4] “User Mode Linux Kernel Home Page”, http://user-mode-linux.sourceforge.net/

[5] K. Kato and Y. Oyama: “SoftwarePot: An Encapsulated Transferable File System for Secure Software Circulation”, Proc. of Int. Symp. on Software Security, Springer, LNCS-2609, 2003. pp. 112-132.

付録 仮想化アルゴリズムの擬似コード

```
// 仮想化されたファイルを置く一時ファイル置き場
String tempPath = "/tmp/vdm";

// 大域変数
String cwd;       // カレントディレクトリ
String vcwd;      // cwd の仮想化先
Set<String> vset; // 仮想化ずみディレクトリの集合


// カレントディレクトリを変更するプリミティブ。
void changeDirecotry(String absolutePath){
  // ".." や "." を含まない正規化されたパス名に変換。
  String normalizedPath
       = normalizePath(absolutePath);
  virtualizeDirectory(normalizedPath);
  // エラーがなければ cwd を更新。
  cwd = normalizedPath;
  vcwd =  virtualizedPathName(cwd);

}

// パス名を対応する仮想化領域のパス名に
// 変換する関数。
// 例 :  "/a/b" -> "/tmp/vdm/a/b"
String virtualizedPathName(String path){
  return tempPath + path;
}

// ディレクトリの仮想化を行う。
String virtualizeDirectory(String path){
  bool alreadyVirtualized = false;
  for (v in vset){
    if (v is ancestor of path || v equals path){
      alreadyVirtualized = true;
      break;
    }
  }
  String vpath = virtualizedPathName(path);
  if (alreadyVirtualized){
    if (vpath exists?){
      return;
    } else {
      error(path + ": directory not found.");
    }
  } else {
    if (path exists?){
      virtualizeDirectoryRec(path);
      vset = vset + path;
      return;
    } else {
      error(path + ": directory not found.");
    }
  }
}

// すでに仮想化された領域を上書きしないように、
// ディレクトリを再帰的にコピーする。
void virtualizeDirectoryRec(String path){
  if (path in vset){
    vset = vset – path;
    return;
  }
  String vpath = virtualizedPathName(path);
  if (vpath does not exists?){
    create directory  vpath;
  }
  for each file in path {
    copy  file to vpath;
  }
  for each directory in path {
    virtualizeDirectoryRec(directory);
  }
}


//----------------------------------------
// ファイル操作プリミティブの例

// ファイルの削除
void deleteFile(String relativePath){
  rawDeleteFile(vcwd + "/"+ relativePath);
}

// ファイルの移動
void moveFile(String fileName,
                    String absoluteDirPath){
  String from = vcwd + "/"+ fileName;
  String originalCwd = cwd;
  String to;
  try {
    // 移動先を仮想化する
    changeDirecotry(absoluteDirPath);
    to = vcwd;
  } finally {
    changeDirecotry(originalCwd);
  }
  rawMoveFile(from, to);
}
```