Copy On Write を用いたオーバーレイブロックデバイスの実装

丹 英之[†]，須崎 有康[‡]，飯島 賢吾[‡]，八木 豊志樹[‡]

株式会社 アルファシステムズ[†]，独立行政法人 産業技術総合研究所[‡]

E-mail: tanh@alpha.co.jp, k.suzaki@aist.go.jp, k-iijima@aist.go.jp, yagi-toshiki@aist.go.jp

概要 Linux Kernel にあるファイルシステムレイヤと物理ブロックデバイスドライバの間に位置する，オーバーレイブロックデバイス(Overlay Block Device, OBD)を実装した．この OBD は，ファイルシステムレイヤからの物理ブロックデバイスへの書き込み要求をトラップし，Copy On Write によって書き込みデータを元の物理ブロックデバイスからの差分データとしてファイルに保持する．実装した OBD について簡単な性能測定を行った結果，ファイルシステムのイメージファイルを loop デバイス経由で参照した場合と同程度のスループットを得ることができた．また，この OBD のドライバをライブ CD に組み込んだことで，書き込み可能で，且つ，変更分を再利用できるファイルシステムをユーザに提供できた．

Implementation of the Overlay Block Device using Copy on Write

Hideyuki Tan[†], Kuniyasu Suzaki[‡], Kengo Iijima[‡], Toshiki Yagi[‡]

Alpha Systems Inc.[†]

National Institute of Advanced Industrial Science and Technology[‡]

E-mail: tanh@alpha.co.jp, k.suzaki@aist.go.jp, k-iijima@aist.go.jp, yagi-toshiki@aist.go.jp

Abstract We have developed the Overlay Block Device (OBD) located between the filesystem layer and the physical block device driver in Linux Kernel. OBD carries out the trap of the write request to the physical block device from the filesystem layer, and save written data as difference from an original physical block device at a file by using copy-on-write. The measuring performance of OBD results nearly same throughput as accessing filesystem image via loop device. We have succeeded in implementing OBD in Live CD with the rewritable filesystem in which difference file is reusable.

1. はじめに

近年，KNOPPIX[1]に代表される CD や DVD から起動する GNU/Linux(以下，ライブ CD)が広まってきた．ライブ CD には，OS(kernel からアプリケーションソフトウェア)のシステム一式が CD-ROM 一枚に格納されている．そして，この CD-ROM からシステムを起動するので，他の OS がインストールされている PC でも簡単に Linux Box として使用することができる．また，書き込み不可能であるメディアの特性から，ユーザの操作によるファイルシステムの破壊はありえない．これらの理由により，高いロバスト性を要求する用途に向いている．特に，学生による想定外の操作に対処しなければならない教育機関の実習用 PC や，不特定多数の人が触れる公衆 PC などでの利用が考えられる．

しかし，ライブ CD に収録されているアプリケーションを単に道具として利用するのではなく，ライブ CD を IT 系専門分野での実習用教材とするには相応しくない．なぜなら，書き込みできないメディアにシステムが構築されていることにより，ユーザの自由度が制限されてしまうからである．よって，本番同様の環境下でのサーバ・クライアントシステムの構築や，ライブラリなどシステム内部に手を入れる実験環境としての用途には不向きである．

そこで，ライブ CD の長所である手軽さを損なうことなく自由にユーザランドを変更できるライブ CD を企図し，書き込み不可能なメディアを用いるストレージ上のファイルシステムを，擬似的に書き込み可能とする機能を追加することにした．

擬似的書き込みの機能は，その元となるストレージへの書き込み要求をトラップし，他の書き込み可能なストレージへ転送することで実現できる．そして，データを参照する場合は，書

き込み可能なストレージを先に参照し，このストレージの該当する箇所にデータが存在しない場合のみ元のストレージを参照すればよい．

我々は，kernel 内部のファイルシステムレイヤと実際のストレージであるブロックデバイスを制御するドライバとの間に仮想レイヤを用意し，このレイヤでデバイスへの読み書き要求を制御する，オーバーレイブロックデバイス(Overlay Block Device, OBD)を実装した．

本稿は 6 章で構成されており，第 2 章では関連研究，第 3 章では OBD の詳細，第 4 章では OBD 実装，そして第 5 章では実装物の簡単な性能評価，そして第 6 章でまとめと今後の展開について述べる．

2. 関連研究

ブロックデバイスを仮想化する先行研究として，OBD 実装の参考にした loop デバイス，そして Device-mapper[2]を挙げる．また，書き込み不可のディレクトリツリーに対する擬似的書き込みを実現する機能の先行研究として，unionfs[3][4]に代表される仮想ファイルシステムを紹介する．そして，User-mode Linux[5][6]に実装されている UML Block Driver についても触れる．

2.1. Loop デバイス

これは，任意のファイルをブロックデバイスとして扱うために Linux へ実装された仮想ブロックデバイスであり，BSD 系 OS では vnode ディスクと呼ばれている．ファイルシステムレイヤからの読み書き要求をブロックデバイスとしてマッピングされたイメージファイルに対して行う．このため，他のファイルシステム上に置かれたイメージファイルを，別のファイルシステムとして利用することができる．しかし，イメージファイルに格納されたファイルシステムが ISO9660 のような書き込み不可のファイルシステムであれば，イメージファイルのアクセス権が読み書き可能な場合でも，この仮想ブロックデバイスのマウントポイントは書き込み不可になる．この仮想レイヤでデータの暗号化を行うことによって，通常のファイルシステムやストレージにあるスワップ領域を暗号化する機能の実装もある[7].

2.2. Device-mapper

これは，実際のブロックデバイスを仮想して汎用的に扱えるようにするレイヤで，Linux2.6 から実装された．LVM2[8]や EVMS[9]と共に用いることで，複数の物理ボリュームをまたがった一つの論理ボリュームの構成や，ストライピング，動的なボリュームサイズの変更，そして Copy On Write を用いたボリュームのスナップショット機能を実現している．Device-mapper は実際のブロックデバイス上にある仮想レイヤで，ファイルシステムレイヤから来たブロックデバイスに対する要求を処理し，あらかじめストレージに用意された物理エクステント(Physical Extent, PE)と呼ばれるユニットに対してデータの読み書きを行う．

loop デバイスではファイルシステムからの要求をファイルに対して行うが，Device-mapper を用いた LVM では物理ストレージを細かく区画したパーティションブロックである PE に対して行う．また，loop デバイスと同様に，このレイヤでの暗号化処理によって，暗号化ファイルシステムや暗号化スワップ領域を提供するための実装もある[10].

Device-mapper 単体でも dmsetup で指定したブロックデバイスに対し Copy On Write によって差分データを他のブロックデバイスに保持できる．

2.3. Unionfs

書き込み不可であるディレクトリツリーを擬似的に書き込み可能にする機能には，ファイルシステムレイヤでの実装がある．Linux での代表的な実装が，unionfs である．これは，Virtual Filesystem Switch(VFS)と実際のファイルシステムとの間に，機能を提供する仮想的なファイルシステムとして用意される．この仮想的なファイルシステムの配下に，読み込みで参照するファイルシステムと書き込み分のデータが保存されるファイルシステムが置かれる形になる．参照するファイルシステム上のディレクトリツリーに対するユーザのファイル変更は，書き込み可能なファイルシステム上のディレクトリツリーに反映される．機能を実現する仮想的なファイルシステムは，これら二つのディレクトリを一つのディレクトリとして束ねてユーザに提供する．

この複数のディレクトリを結合する概念のファイルシステムには，3-D File System[11]，Translucent File System[12]，そして，Plan 9 の union directory[13]と，4.4BSD に実装された Union Mounts[14]がある．Linux では，ユーザ空間で動くプロセス経由で仮想ファイルシステムがあり[15][16]，この仕組みを利用したものや[17]，

unionfs のようにカーネル空間での実装が幾つかある[18][19][20].

2.4. User-mode Linux の UML Block Driver

User-mode Linux(UML)は，ユーザプロセスで動作する Linux カーネルで，仮想 OS の一つである．この UML の内部で使用されるブロックデバイスは，UML Block Driver(UBD)で提供される．

UBD は，UML を動かすホスト OS のファイルシステム上にあるイメージファイルを UML 内部からブロックデバイスとして参照できる機能を提供し，UML 起動時に指定したイメージファイルを内部から UBD 経由でマウントできる．そして，Greg Lonnon によって追加された Copy On Write 機能[21]は，UML がイメージファイルへ書き込もうとしたデータを，元のイメージファイルとの差分データとして，COW と呼ばれるファイルに保存する．よって，一つのイメージファイルを複数の UML 間で共有して使用することが可能になる．本研究の OBD では，差分データファイルを UML の COW ファイルと同じ形式を採用したので，UML と差分データファイルの相互利用が可能である．この UBD を応用することでライブ CD のカスタマイズを容易に行えるようにした試みもある[22].

3. OBD の詳細

kernel のソフトウエアレイヤにおける OBD の位置を，図 1 に示す．OBD は，ファイルシステムレイヤと，物理ストレージをブロックデバイスとして利用するためのドライバとの間に置かれる．ユーザ空間で動くプロセスが発行したファイル操作のシステムコールは，VFS を経由しそれぞれのファイルストアに渡される．ファイルストアでは，デバイスに対する原始的な読み書き命令に置き換えられ，デバイスドライバに渡される．OBD は仮想的なデバイスのドライバとして，ファイルストアと実際のブロックデバイスとの間で読み書きの要求をトラップすることになる．

・書き込み要求の場合

ユーザプロセスがファイル作成や編集などファイルシステムに変更を加えると，そのファイルシステムのファイルストアが OBD へ書き込み要求を出す．OBD は，そのファイルシステムが格納されているブロックデバイスの変更する該当部分のデータを，再び VFS を経由して別のブロックデバイスにあるファイルシステム上の差分ファイルへ書き込む．この書き込みの際，ブロックデバイス上のどの位置に変更が発生したのかを示すフラグを記する．

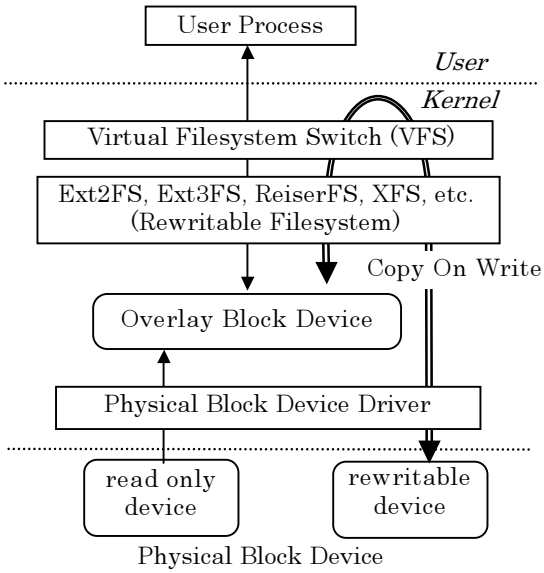


図 1. kernel 中のソフトウェア階層における OBD の位置

・読み込み要求の場合

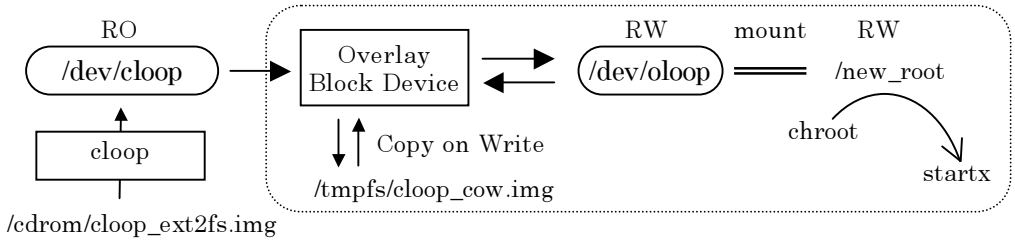
ユーザプロセスがファイルやディレクトリエントリを参照すると，そのファイルシステムのファイルストアが OBD へ読み込み要求を出す．OBD は，そのファイルシステムが格納されているブロックデバイスの参照部分が変更されているか変更フラグの確認を行う．過去に変更が有る場合，読み込むデータは差分ファイルに存在するので，VFS を経由して別のブロックデバイスにあるファイルシステム上の差分ファイルからデータを読み込む．そして変更が無い場合には，実際のブロックデバイスを参照する．

これら読み書きのデータは，セクタ単位(512B)で扱い，変更フラグはセクタ当り 1bit のビットマップを用いて保持する．

第 2 章で述べた通り，OBD の差分データを保持するファイルは，UML の UDB で用いられる COW ファイル(v3)と同じ形式を採用した．この COW ファイルのヘッダには，

- COW ファイルの識別のための，マジックナンバーとバージョン．
- 元のイメージファイルに変更が無いことを確認するための，イメージファイルのパス，最後に更新した時刻，サイズ．

図 2. OBD を適用したライブ CD のデバイスノードとマウントポイントの関係



・COW ファイルで扱うセクタサイズ.

が格納され，その後にブロックの変更状況についてのビットマップデータ，そして実際のブロックのデータが続く．このビットマップとブロックの部分はスパースとなっており，未使用部分は零のデータとして論理的なディスク領域を確保するが，物理的なディスク領域を消費しない．そして，変更のあったブロックは，元のイメージファイル内の位置と，相対的に同じ位置に書き込まれる．つまり，変更のあったブロックの分だけ実際のディスク領域を消費する．

UML の COW ファイルと同じ形式にしたことで，UML utilities に含まれる，新規 COW ファイルを作成する uml_mlcow，そして COW ファイルに保持されたデータと元としたイメージファイルを合成する uml_moo をそのまま利用できる．

OBD は，Device-mapper のブロックデバイスに対する Copy On Write 機能の差分データ保存先として，loop デバイスのようにファイルを対象とする．言い換えると，UML の UBD をそのまま実機で動く kernel で利用できるようにしたものである．unionfs とはユーザへ提供する機能面では似ているが，実装レイヤが異なる．

4. OBD の実装

OBD の実装と，ライブ CD への組み込みについて述べる．

4.1. OBD ドライバ

OBD はカーネル空間で仮想ブロックデバイスとして動き，指定されたブロックデバイスと COW ファイルのデータを読み書きする．そこで，第 2 章で挙げた loop デバイス (kernel/device/block/loop.c) を元に機能を実装した．具体的には，主機能である COW によるブロック読み書き制御の他に，COW ファイルを設定するための ioctl の追加，procfs での状況表示である．デバイスのマイナー番号は 0〜7 まで準備したので 8 個のデバイスを同時に利用できる．このドライバは，Kernel Module として linux2.6.8.1 (2.6.11.7)で動作するように実装した．

4.2. 差分ファイル設定コマンド

COW ファイルを OBD ドライバへ設定するため，loop デバイスの設定と制御を行う losetup コマンドに，上記で追加した ioctl を発行する機能を追加した．このコマンドの実装には，util-linux2.12b を元に行った．

4.3. ライブ CD への組み込み

OBD ドライバを，ライブ CD の KNOPPIX3.7 日本語版[22]へ組み込んだ．KNOPPIX では，ルートファイルシステムを圧縮 loop デバイス (cloop)に載せている．この cloop は，圧縮されたイメージファイルを扱うことができる loop デバイスである．扱うイメージファイルは，元となるファイルシステムイメージをブロック毎 (64KB)に分け，zlib で圧縮されている．cloop ではデバイスの参照時に該当ブロックを透過的に伸張するので，ユーザは圧縮されていることを意識することなくイメージファイル中のファイルシステムをマウントできる．ただし，cloop デバイスドライバ自体には圧縮機能が無いので，読み出し専用のデバイスである．今回はこの cloop デバイスに OBD を適用することで，ユーザランドを自由に変更できるライブ CD を作成した．

KNOPPIX ではブートローダに isolinux を用いており，起動時に kernel と miniroot を読み込む．そして，この miniroot にある初期化スクリプト linuxrc の処理で CD-ROM 自体をマウントし，この CD に格納された cloop の圧縮イメージファイルをマウントする．その後，システムに依存してファイルが更新されるディレクトリ，/etc，/var を tmpfs に用意する．linuxrc 終了後，kernel は init を起動し，AutoConfig でのデバイス検出などライブ CD 特有の処理を行う．後は，通常の HD インストールされた Linux と同等の起動シーケンスで，X が起動しユーザが利用できる状態になる．

OBD の導入では，ユーザが行った変更を次回

表 1. OBD のベンチマーク結果

	Sequential Output			Sequential Input		Random
	putc (KB/sec)	put_block (KB/sec)	rewrite (KB/sec)	getc (KB/sec)	get_block (KB/sec)	seeks (/sec)
OBD	20,052	22,219	10,445	17,242	27,946	123
direct	27,454	36,246	12,849	21,276	29,446	223
bd-loop	20,752	28,620	12,392	18,360	28,670	117
fs-loop	22,828	21,417	11,588	20,992	28,943	123

のライブ CD 起動時でも流用できることを企図した．そこで COW ファイルを取り出す手順を踏めるよう，chroot(1)下でユーザ環境を提供するようにした．

図 2 に，この OBD を適用したライブ CD のデバイスノードとマウントポイントの関係を示す．CD には，Ext2FS の圧縮イメージファイルが格納されている．この圧縮イメージファイルを参照する cloop デバイスは，参照ファイルが格納されているメディアと cloop ドライバの特性によって書き込み不可である．この cloop のデバイスノード/dev/cloop に，OBD ドライバを適用することで擬似的に書き込みが可能になる．そして，この OBD のデバイスノード/dev/oloop をマウントすることで，書き込み可能なルートツリー/new_root が実現できる．このツリー配下の変更は全て，tmpfs 上に用意した COW ファイルに反映される．そして，/new_root 環境下でもプロセスが正常に動くように実際の/proc，/dev，/sys を/new_root 配下へ再マウントした後，このルートツリーへ chroot しユーザ環境である X を起動する．ユーザのセッション終了後，/new_root のマウントを解除することで，ユーザが行ったファイルシステムの変更は全て COW ファイルに残る．

つまり，ユーザが X のセッションを終えた後，tmpfs にある COW ファイルを取り出し，次回起動時の OBD 処理前に COW ファイルを準備できれば，その COW ファイルの差分が反映されたルートツリーを利用できることになる．

5. 評価と考察

実装した OBD，そして，OBD を組み込んだライブ CD についての評価を述べる．

5.1. OBD の評価

OBD では，物理的なブロックデバイスに対する読み書きをトラップし，Copy on Write によって差分データを保持するファイルを読み書きする．OBD を適用しない通常のブロックデバイスとでデータ読み書きの性能を比較した場合，Copy On Write の処理によりスループットの劣化が予想される．そこで，通常のブロックデバイス，実装した OBD，そして OBD と似たレイヤ構造でのブロックデバイスの参照について，ベンチマークを行い性能評価した．

ベンチマークでは，ブロックデバイスに対してキャラクタベースとブロックベースの連続書き込みと読み込み，そして，ランダムシークについて測定した．この結果を表 1 に示す．

表中，OBD は物理デバイスを OBD 経由で参照，direct は実際のブロックデバイスである物理デバイスを直接参照，bd-loop は物理デバイスを loop デバイス経由で参照，fs-loop は物理デバイス上のファイルシステムに置かれたイメージファイルを loop デバイス経由で参照した場合である．また値は，連続して 3 回計測した結果の平均値である．OBD では，まだ COW ファイルに差分データが無い場合，物理デバイスを loop デバイス経由で読み込む bd-loop とレイヤ構造が同じ状態になる．また，COW ファイルに差分データが有る場合には，ファイルシステム上に置かれた COW ファイルを読み書きするので，イメージファイルを loop デバイス経由で参照する fs-loop と同じレイヤ構造になる．キャラクタベースでの書き込みには putc()，読み込みには gets()，ブロックベースではデータサイズを 8KB として書き込みには write()，読み込みには read()を用いた．また，rewrite も 8KB のデータサイズで read()，lseek()，write()を繰り返した．ランダムシークでは，ランダムに生成したファイルの位置へのシークを 8192 回行った．OBD を適用するブロックデバイスのサイズは 4GB とし，fs-loop で参照するファイルシステムイメージは 2GB とした．また，物理デバイス及び，loop デバイスで参照したイメージファイル上のファイルシステムは Ext2FS にした．これらベンチマークには，bonnie++ 1.03a を用いた．また，ハードウェアには，CPU:P4-2GHz, MEM:512MB, HD:MAXTOR 5T040H4(40.9GB, 7200RPM, 2MB CACHE, ATA-100)，IDE transfer mode が UltraDMA Mode 2 であ

表 2. OBD と Unionfs の比較

	適用ディレクトリ上			適用前ディレクトリ上
	既存ファイルの消去 (sec)	ファイルの生成 (sec)	生成したファイルの消去 (sec)	ファイルの生成 (sec)
OBD	0.83	17.44	0.17	17.51
Unionfs	33.23	23.27	0.58	

る PC を用いた．ベンチマークで生成するファイルサイズはメモリのキャッシュに収まらないよう，物理メモリの 2 倍である 1GB とした．

結果から，レイヤを重ねるに従いスループットが悪くなる傾向が確認できる．これは，レイヤ間でのリクエストに伴うメモリコピーなど，余計な処理が増えるからである．HD へのデータ転送速度の論理ピークが 33.3MB/sec であるにもかかわらず，ブロックデバイスを直接マウントした場合のブロック書き込みでは，35.4MB/sec の値を得た．これは，Ext2fs の遅延書き込みに拠るものであると考えられる．

OBD と物理デバイスの直接参照を比較すると，連続書き込みではキャラクタベースで 27%，ブロックベースで 39%と性能低下が見られた．しかし，連続読み込みのキャラクタベースでは約 19%，ブロックベースでは約 5%と，書き込み時に比較して性能劣化の度合いが低い．この劣化度も，Ext2fs の遅延書き込みによって，書き込みに対する劣化度から相対的に低くなると考えられる．rewrite では約 19%の低下となった．ブロックベースの書き込みで 39%，ランダムシークで 45%と低下の度合いが大きいが，ブロックベースでの読み込みでは 5%の低下の度合いであることを考えると，rewrite では，I/O が局在化するためファイルシステムのキャッシュ効果が効いていると考えられる．

OBD と fs-loop，bd-loop を比較すると，連続書き込み，連続読み込み，ランダムシーク，共に極端な差がみられなかった．これは，OBD の挙動が，書き込みに対する操作では fs-loop と同じ，読み込みに対する操作では，データの更新具合によって bs-loop と fs-loop が切り替わるという振る舞いをするからである．

5.2 OBD と Unionfs との比較

OBD は，デバイスレベルで書き込み不可のブロックデバイスを疑似的に書き込み可能とする．これは，適用するブロックデバイス上に書き込み可能なファイルシステムがあれば，そのディレクトリツリーも擬似的に書き込み可能となる機能を提供すると言える．この機能に注目し，ファイル生成及び消去に要する時間を unionfs と比較した．この結果を表 2 に示す．

表中，適用ディレクトリ上とは，OBD，unionfs を適用したマウントポイントのことを指す．また，適用前ディレクトリは，OBD，unionfs それらを適用しないディレクトリのことであり，詳細は後述する．ファイルの生成・消去で扱ったファイル数は 50,000 個で，OBD，unionfs のマウントポイント配下で creat()，unlink()をファイル数分，繰り返した．OBD，unionfs 共に，tmpfs 上に用意した 100MB の Ext2fs イメージファイルを loop デバイス経由で参照し，計測の対象となるディレクトリを提供するブロックデバイスとした．適用前ディレクトリは，このブロックデバイスのマウントポイントのことである．OBD では，この loop デバイスと関連付けたデバイスノードのマウントポイントを計測するディレクトリとした．unionfs では，上記 loop デバイスのノード二つを読み込み専用，読み書き専用のディレクトリとしてマウントし，このディレクトリ二つを unionfs で束ねたマウントポイントを計測するディレクトリとした．計測には unionfs-1.0.11，linux2.6.11.7 を用いた．またハードウェアには，OBD の評価と同じマシンを用いた．

OBD では，ファイル生成のスループットが適用しない場合とほとんど変わらず，2860 個/sec であった．計測対象のディレクトリに既存するファイルの消去では，新規生成したファイルの消去よりも時間を要した，既存ファイルの i-node ブロックについて，一旦元のブロックデバイスを参照した後，COW ファイルへファイル情報の書き込みを行うからである．

unionfs では，ファイルストアレイヤでのディレクトリ情報の処理に時間を要するため，OBD と比較すると全体的に遅い．ファイル生成では 1.3 倍，生成したファイルの消去では 3.4 倍の時間であるが，既存ファイルの消去では，OBD の 40 倍もの時間を要した．unionfs では，読み込み専用ディレクトリ配下に有るファイルが消去さ

れた，という情報を読み書き専用ディレクトリ配下に，特殊なファイル名を持つファイルを生成することによって保持する．それ故，消去される既存ファイルの確認と消去情報保持のファイル生成により，OBD と比べて遅くなる．

5.3. OBD を組み込んだライブ CD

OBD を組み込んだライブ CD では，パッケージの追加や削除など，HD にインストールされた Linux ディストリビューションの様に扱うことができた．また X のセッション終了後，COW ファイルをネットワーク上のストレージへ保存し，PC の再起動後，デスクトップ環境起動前にその COW ファイルを取り寄せる事で，前回の更新内容が反映されたルートツリーの環境を利用できた．

差分データを格納している COW ファイルはスパースファイルなので，物理的にストレージを消費していない．しかし，論理的には OBD を適用する cloop の伸張後のイメージサイズ，約 1.8GB 相当のファイルサイズになる．つまり，ネットワーク経由での転送時に行った COW ファイルの圧縮・伸張では，1.8GB の比較的大きなデータを扱うことになり，処理に時間を要した．

6. まとめ

本稿では，ファイルシステムレイヤからの物理ブロックデバイスへの書き込み要求をトラップし，Copy On Write によって書き込みデータを元の物理ブロックデバイスからの差分データとしてファイルに保持する，オーバーレイブロックデバイスの実装，簡単な評価，そしてライブ CD への組み込みについて述べた．

実装物の評価では，loop デバイスと同程度のスループットが出ることが判った．また，OBD を組み込んだライブ CD では，HD インストールの Linux のように，アプリケーションの追加・削除ができた．そして，そのルートファイルシステムの更新分を，システムの再起動を跨いで利用できた．

この OBD を組み込んだライブ CD の評価中，新しくリリースされた KNOPPIX3.8 では，unionfs が搭載されルートファイルシステムを変更できるようになった．しかし，ファイル生成・消去について OBD との比較の結果から，unionfs ではディレクトリ情報の処理に時間を費やすので，一つのディレクトリに大量のファイルを保持しているパッケージ管理システムでは，アプリケーションの追加や削除は非常に遅くなることが予想される．一方，OBD ではブロックデバイスレベルで処理するので，unionfs の様な問題は発生しない．ところが，unionfs では更新のあったファイルは，別ディレクトリにファイル単体で保持される．よって，ユーザにとっては unionfs を適用していない場合でも更新したファイルを容易に参照できる．OBD で更新分を参照する場合には，その元となっているブロックデバイスが必要になる．故に，ライブ CD へ unionfs を適用するかそれとも OBD を適用するのかという問題は，ユーザがライブ CD を利用する目的次第であると思われる．

先行研究で挙げた Device-mapper でも，loop デバイスと併用する事によって，差分データをファイルの形で保持できることが後の調査で判明した．これにより本研究の当初の目的が解決できる．しかし今回の実装では，ファイルフォーマットを UML の COW ファイルと同じ形式にしたことによって，UML と実機間とで差分データを流用できる．

ライブ CD 以外には，HD への適用が考えられる．例えば HD のパーティションへ OBD を適用することによって，ユーザに対し LVM のように差分データ専用領域の用意を要求することなく“HD への書き込みを無かった事にする”機能を提供できる．

今回用いた COW ファイルフォーマットではスパースファイルであり，物理的にはファイルサイズが小さくても論理的には大きく，差分データの取り扱いに時間を要すこととなった．今後利用していくにあたっては，効率よく扱えるフォーマットの検討が必要である．また，差分情報を時系列で管理することによるブロックデバイスのスナップショット機能なども検討したい．これには，COW ファイル形式の拡張が必要であろう．そして，cloop と統合することで，透過的に再圧縮される書き込み可能な圧縮 loop デバイスが実現できると考えている．

本研究で実装した OBD は，kernel への patch として Linux-kernel ML への投稿を予定している．

参考文献

[1] Klaus Knopper, “Building a self-contained auto-configuring Linux system on an iso9660 filesystem”, the Annual Linux Showcase, October 2000.

[2] Device-mapper Resource Page. http://sources.redhat.com/dm/

[3] C. P. Wright and J. Dave and P. Gupta and H. Krishnan and E. Zadok and M. N. Zubair. “Versatility

and Unix Semantics in a Fan-Out Unification File System.", Stony Brook U. CS TechReport FSL-04-01b, October 2004.
[4] A stackable Unification File System Web Page. http://www.fsl.cs.sunysb.edu/project-unionfs.html
[5] Dike Jeff, "A user-mode port of the Linux kernel", Proceedings of the 4th Annual Linux Showcase & Conference (2000), October 2000.
[6] The User-mode Linux Kernel Home Page. http://user-mode-linux.sourceforge.net/
[7] Loop-aes Project Web Page. http://loop-aes.sourceforge.net/
[8] LVM2, Logical Volume Manager, Resource Page. http://sourceware.org/lvm2/
[9] EVMS, Enterprise Volume Management System, Project Web Page. http://evms.sourceforge.net/
[10] dm-crypt, a device-mapper crypto target, Web Page. http://www.saout.de/misc/dm-crypt/
[11] D. G. Korn and E. Krell. "A New Dimension for the Unix File System.", Software-Practice and Experience, pages 19-34, June 1990.
[12] D. Hendricks. "A Filesystem For Software Development.", In Proceedings of the USENIX Summer Conference, pages 333-340, Anaheim, CA, June 1990.
[13] AT&T Bell Laboratories. Plan 9 Programmer's Manual, March 1995.
[14] J. S. Pendry and M. K. McKusick. "Union mounts In 4.4BSD-Lite." In Proceedings of the USENIX Technical Conference on UNIX and Advanced Computing Systems, pages 25-33, December 1995.
[15] LUFS (Linux Userland File System) Project Web Page. http://lufs.sourceforge.net/lufs/
[16] FUSE (Filesystem in Userspace) Project Web Page. http://fuse.sourceforge.net/
[17] LUFS-based unionfs for Linux Web Page. http://alumnus.caltech.edu/~muresan/projects/lufs_unionfs.html
[18] Overlay Filesystem Home Page http://mywebpages.comcast.net/artn/ovlfs/ovlfs.html
[19] Overlay Filesystem Project Web Page. http://ovlfs.sourceforge.net
[20] translucency lkm Project Web Page. http://translucency.sourceforge.net
[21] Dike Jeff. "User-mode Linux", Proceeding of the 5th Annual Linux Showcase & Conference, November 2001.
[22] 須崎，飯島，八木，丹, "Copy On Write を適用した UML-KNOPPIX", FIT2004 第 3 回情報科学技術フォーラム，2004 年 9 月.
[23] KNOPPIX Japanese edition. KNOPPIX Japanese edition Web Page. http://unit.aist.go.jp/itri/knoppix/