NetBSD ベースシステムパッケージ化技法の実装報告

榎本 優樹[1,a)]　深町 賢一[1,b)]

概要：OS のユーザランド構成における粒度は OS ごとに様々であるが，運用面を重視すれば，より小さな粒度でのアップデートやリスタート，ロールバックなど，こまやかなシステム管理ができるように OS ビルドシステムを再構成するべきである．歴史的経緯により，Linux ディストリビューションは粒度の細かなパッケージ群から OS が組み上げられているが，BSD Unix はシステム全体でも十数種類の分割で，かつ基本的に上書き処理という粒度の荒い構成となっており，改善が望まれる．そこで，我々は，NetBSD ベースシステムをパッケージ化するシェルスクリプトベースのシステム basepkg を開発し，より細かな粒度でのシステム管理と機能拡充を実現した．basepkg を pkgsrc（NetBSD サードパーティソフトウエア管理フレームワーク）の上に作成することで，既存の不完全なフレームワーク syspkg にくらべ，より少ないコード量で見通しの良いソフトウエアとなっている．また，実運用環境下での評価のため，ベースパッケージ配布サーバも試験運用している．本システムは，NetBSD ユーザやカスタマーにとっての利便性向上に寄与すると考えられるが，解決するべき課題も多い．本稿では，一つの大きなシステムを細かい粒度へ再構成した実践例において，以前より改善できた点と，再構成したことで新たに生じた課題について報告する．

キーワード：BSD Unix, シェルスクリプト, オープンソースソフトウエア, システム粒度, システム管理

The Implementation of the NetBSD Base System Packaging

Yuuki Enomoto[1,a)]　Ken'ichi Fukamachi[1,b)]

Abstract:
It is considered that UNIX operating system (OS) built on fine granular small parts is preferable to one built on the traditional large tarballs in order to support speedy security update, easy replacement and rollback of specific parts. In Linux distributions, the system are already divided into many small packages. On the other hand, BSD Unix variants are behind the curve on the base system packaging. To improve NetBSD base system granularity, we have developed a software "`basepkg`" by making the best use of pkgsrc framework. It is shorter than the existing base packaging framework `syspkg`, easy to understand and an almost POSIX compliant shell script for more sustainable maintenance. Also we operate an experimental base package distribution server to evaluate our software in realistic environment. It is shown that `basepkg` provides faster replacement of a few OS granular parts and extra useful functions. It must be beneficial for NetBSD users and customers.

Keywords: BSD Unix, shell script, open source software, system granularity, system management

1. はじめに

オペレーティングシステム（OS）のユーザランド構成における粒度は OS ごとに様々である．歴史的経緯により，Linux ディストリビューションは粒度の細かなパッケージ群から OS が組み上げられているが，BSD Unix はシステム全体でも十数種類の分割で，かつ基本的に上書き処理という粒度の荒い構成となっており，改善が望まれる．

Unix は，歴史的に一つのソースツリーで管理，配布されてきたが，今日， OS を粒度の小さな部品から組み上げることには利点がある．たとえば（1）高速なセキュリティ

[1] 千歳科学技術大学
758-65 Bibi, Chitose, Hokkaido, 066-8655, Japan
[a)] m2160020@photon.chitose.ac.jp
[b)] k-fukama@photon.chitose.ac.jp

アップデート（2）特定の部品だけの入れ替え（3）特定の新機能だけを運用機材へ取り込む機能（4）更新時にデーモンの再起動を確実に実行する機能（5）粒度の細かな更新履歴（6）失敗時のロールバック機能などが実装されていることが望ましい．

そこで，我々は NetBSD ベースシステムをパッケージ化するシェルスクリプトベースのシステム basepkg を開発し，より細かな粒度でのシステム管理と機能拡充を実現した．また，実運用環境下での評価のため，ベースパッケージ配布サーバも試験運用している．basepkg は，pkgsrc（NetBSD サードパーティソフトウエア管理フレームワーク）の上に作成することで，既存の不完全なフレームワーク syspkg にくらべ，より少ないコード量で見通しの良いソフトウエアとなっている．

本稿では，basepkg の設計と実装について述べる．

2. Unix のソフトウエア配布方式

FLOSS（Free Libre Open Source Software）の Unix 配布方式の粒度と区分は様々である．

tar(1) でまとめて gzip(1) などで圧縮した形式の tarball（例：something.tgz）は配布の際によくつかわれるアーカイブの例である．BSD Unix のベースシステムは大きな tarball 配布の例だ．ここで，用語 “ベースシステム” は，プロジェクトが公式に維持管理し配布している OS 基幹部分を指している．“ベース” の意味する範囲はプロジェクトによって異なり，含まれているプログラムも少しずつ異なる．歴史的に BSD Unix のベースシステムは，（現代的な利便性はともかくとしても），Unix プログラミング環境として必要十分なものが一式含まれている．ただし配布は，OS が動作する上での必須部分（base.tgz）やコンパイラ（comp.tgz），マニュアル（man.tgz）など十数種類の tarball からなる粒度の荒い構成で行われてきた．

BSD Unix では，もともとベースシステムという考え方があるため，OS はベースシステムとそれ以外つまりサードパーティからなる，つまり二つの区分からなると考えるものが多い．サードパーティはパッケージと呼ばれ，パッケージを追加・削除する管理システムがベースシステムとは別に用意されている．用語「パッケージ」は，プログラムやライブラリ，設定ファイル，マニュアルなどが一つの配布ファイルとなっているものを指している．パッケージのフォーマットや管理システムもプロジェクトにより様々である．

歴史的に，BSD Unix では，ベースは大きな tarball で，サードパーティを別途パッケージとして管理してきた．

一方，Linux ディストビューションは，もともと小さなパッケージの集合として組み上げられており[*1]，ベースやサードパーティという区別は意識[*2] させられないことが多い．Linux ディストビューションにおいても，ベースシステムといった用語を使うことがあるが，それは OS が動作する上での必須部品群を意味するメタパッケージ名と考えてよい．

3. NetBSD 配布方式の現状

本節で説明するようにベースシステムとサードパーティは明確に分かれており，操作方法も異なっている．つまり二重運用になっている．サードパーティの取扱いのほうが現代的で優れているため，開発においては，サードパーティ側にあわせていくほうが開発効率がよいと考えられる．

3.1 ベースシステム管理方式の現状

ソースツリー（典型は/usr/src，以下 src/）には，配下に 15000 余りのディレクトリがあり，make を使った伝統的なビルドシステムで OS ベースシステムの配布セットを作成することができる．具体的には src/で make release などを実行すると，最終的に base.tgz などの tarball が生成される．NetBSD の特徴は src/直下にあるシェルスクリプト build.sh[2] である．これはクロスプラットフォームなツールチェインを作成し，任意のアーキテクチャをクロスコンパイルする wrapper である．また，build.sh は，配布アーカイブやインストールメディアの作成，ベースシステムアップデートなどの操作をする際の wrapper でもある．

このビルドシステムには，ベースシステムのパッケージ化の仕組み “syspkg” が 2002 年に導入されたが，この数年 syspkg の開発は停滞[*3] いるため，syspkg は最新の NetBSD とは不整合になっている．たとえば，syspkg が作成するパッケージは情報の欠落が多く，src/distrib/sets/配下にあるメタデータ（deps や comments，attrs ファイルなど）は不完全である．また致命的な問題として，システ

[*1] パッケージという用語には，ブランドやビジネスモデル戦略の意味合いもあれば技術的な観点もある．本稿はパッケージ化する技術について論じていることに注意されたい．“OS をパッケージングする” つまり”コミュニティの作成した多くのソフトウエアの集合体として OS を組み上げる” という思想の明瞭な証言は 1995 年の Debian が初出のようで，この時代に認識されたように思われる．しかしながら，これらの部品の品質はまちまちである．作者も品質もまちまちな無数の寄せ集め部品の OS を信じることは難しい．RedHat ディストリビューションというブランドは，ソフトウエアハウスのようにソースコードそのものを売る（パッケージビジネス）ではなく，信頼を売っている．Redhat 社のビジネスモデルが証明したことは，フリーソフトウエアにおいても，ブランディングとパッケージングが重要であるという事実である [1]．これは，のちの OSI（Open Source Initiative）創設（1998 年）と，多大な成功をおさめたマーケティング用語 “オープンソース” へと間接的につながっていくと考えられる．

[*2] サードパーティか否かよりも，ライセンス区分を意識させられることのほうが多いようである．

[*3] NetBSD wiki では “There has been a lot of work in this area already, but it has not yet been finalized” [3] とされている．たとえば 2012 年の PR46937 チケット [4] は以前手づかずであるし，ソースコード管理システムのログを分析すると 2010 年 2 月以降 syspkg が保守されている形跡はない．

```
openssh-7.5.1nb1.tgz/
    +CONTENTS          +COMMENT
    +DESC              +INSTALL
    +DEINSTALL         +DISPLAY
    +BUILD_VERSION     +BUILD_INFO
    +SIZE_PKG          +SIZE_ALL
    bin/
        scp      sftp      ssh
    ...snip...
```

図 1　pkgsrc のパッケージ openssh-7.5.1nb1.tgz の中身
Fig. 1　Content of a pkgsrc package openssh-7.5.1nb1.tgz

ムの上書きや削除してはならない部分を syspkg では偶然消してしまう可能性もあるが未修正のままとなっている．

3.2 NetBSD pkgsrc(7)

NetBSD のサードパーティソフトウエア管理システムは pkgsrc[*4] と呼ばれている．pkgsrc は “NetBSD を含めた Unix クローンにおいてサードパーティソフトウエアをビルドし管理するフレームワーク” で，移植性のあるパッケージ管理システムを目指しており，多くの Unix 風プラットホーム上でビルドできるよう設計されている．

pkgsrc の使い方は src/と同様である．ここでは “openssh”（pkgsrc/security/openssh）を例にとろう．“openssh” パッケージを作成するには，pkgsrc/security/openssh ディレクトリで make package を実行する．最終的に，pkgsrc/packages/All ディレクトリに openssh-7.5.1nb1.tgz というファイルが作成される．これが “openssh” パッケージである．このパッケージは，メタデータ群とバイナリ，設定ファイル群などから構成されている．図 1 は openssh-7.5.1nb1.tgz の中身である．“+” 文字で始まる小さなテキストファイル群はメタデータである．

パッケージ群を操作する際には，pkg_で始まるコマンド群を利用する．インストールには pkg_add(1)，削除する際には pkg_delete(1) コマンドを用いる．これらのコマンドでは，実行時にメタデータ内のフックを実施するなどの拡張機能が利用でき，ベースシステムよりも利便性が高い．また，pkg_コマンド群はベースシステムに含まれているが，pkgsrc には，それら以外にも高機能なユーティリティがあり，pkgsrc を全面的に活用することで，効率的な開発が可能となる．

4. Basepkg の設計と実装

我々は NetBSD ベースシステムパッケージングのために syspkg にかわる basepkg という別のフレームワークを開発した．本節では，basepkg の動作の詳細を説明する．

[*4] https://www.pkgsrc.org/

特徴	syspkg	basepkg
言語	Makefile と Bourne shell 混在	Bourne shell
フック	なし	あり
カーネルパッケージ	なし	あり
リポジトリ	公式	pkgsrc-wip

表 1　syspkg と basepkg の比較
Table 1　Comparison between syspkg and basepkg

4.1 Basepkg の概要

basepkg は BSD ライセンスのオープンソースソフトウエアで，github から入手可能である[*5]．2017 年 5 月からは pkgsrc の開発版である pkgsrc-wip[*6] に登録されている．basepkg と syspkg の主な違いを表 1 に示した．

basepkg は 1000 行強の Bourne shell スクリプトで，可読性が高く，長期の運用に耐えられることを目指している．

basepkg はメタデータを解析し，適宜 pkg_*プログラム群を呼び出すことで処理を進める．利用するメタデータは syspkg に由来するが，basepkg の独自拡張により修正が加えられている．

4.2 Basepkg の設計方針

第一にプログラミング言語には POSIX 準拠のシェルスクリプトを選択した．

NetBSD は移植性を重視した OS で，いまでも 50 以上のプラットフォームをサポートしている．そういったクロスプラットフォームでの動作が必須である．NetBSD はリリースエンジニアリングとしても，可能なかぎり後方互換性を維持しようとしてきている．

そういった方針にあう開発方針は，シェルスクリプトであると考えた．

POSIX 規格を中心としたプログラミング手法は「POSIX 中心主義」と言われ，システムの持続性や互換性といった効果検証がおこなわれている [5]．

コンパイラ型の高級プログラミング言語は実行速度が速いと言われているが，シェルスクリプトでも awk (1) や find (1)，xargs (1) で扱える内部ループを中心とした実装をおこなえば実用的な実行速度が出せるという報告が松浦らによりなされている [5]．またシェルスクリプトはコンパイルをおこなう必要がなく，プログラムのテストをコンパイルが必要な言語より簡単におこなうことができる．

現在の実装では，basepkg は 3 つのコマンド (hostname(1)，mktemp(1)，pkg_create(1)) をのぞいて POSIX 準拠である．また，コーディングスタイルは ShellCheck[*7] によりチェックをおこない，可能なかぎり素直なコードを目指している．

[*5] https://github.com/user340/basepkg
[*6] https://www.pkgsrc.org/wip/
[*7] http://www.shellcheck.net/

第二に，可読性と理解しやすさを重視することとした．そこで，syspkg のようにファイルを分散させず，単一のファイルに処理をすべて記述している．

Unix 哲学の観点からすれば，「スモール・イズ・ビューティフル」[6] であり，「一つのプログラムには一つのことをうまくやらせる」[6] ことが Unix らしい開発手法であるといえる．また Kernighan らも『ソフトウェア作法』において，小さいプログラムを複数作り，それらをつなぎ合わせることでテキストエディタを開発するプロセスを読者に示した [7]．ファイルを複数個に分散させてそれらをつなぎ合わせる syspkg の設計は，Unix 哲学的には正しく，また実践されてきた手法であるといえる．

しかしこのような設計では，プログラムの実行の流れを上から下へと追っていくことが困難である．

小さい部品を組み上げてプログラムを書くことは重要であるが，プログラムの処理の流れを上から下へ逐次的に読めることもまた重要であると Kernighan[8] や Boswell ら [9] によって指摘されている．今回の basepkg の開発では，Unix 的な設計思想よりもプログラムの可読性に重点を置き，単一のファイルにスクリプトをすべてまとめることとした．

第三に，できるだけ pkgsrc にある機能を再利用する方針としている．パッケージを管理するために作られ，きちんと維持管理もされている pkgsrc の上に構築する方が効率的な開発が可能であると考える．

basepkg では，作成するパッケージのフォーマットを pkgsrc と合わせることで，pkgsrc の管理システムをそのまま利用できるようにしている．

なお，basepkg の利用するメタデータは，syspkg に由来するが，basepkg 側で修正を行っている．そのため，それらの差分は basepkg で独自に管理している．たとえば，削除不可のメタデータ（“+PRESERVE” ファイル）を修正するため，basepkg では独自に essential[*8] というリストを管理している．そのほか basepkg は，より正しい依存関係の確認などもおこなっている．

これらの方針により basepkg は，syspkg より簡潔で，可読性が高く，維持管理しやすくなりつつも，充実した機能を持つことが可能になっている．

4.3 Basepkg の内部処理

本節では basepkg の処理の詳細をしるす（**図 2**）．

(1) メタデータの収集と修正
basepkg は syspkg のメタデータを元にデータを準備し，次のステップへ備える．
basepkg は src/disrib/ ディレクトリ以下にある sets/lists/base/mi （アーキテクチャ非依存）ファイルおよび sets/lists/base/md.ARCH （アーキテクチャ依存）ファイルから，ファイル名，パッケージ名，各種オプション情報を収集する（sets/以下にある情報の詳細は src/distrib/sets/README を参照）．
mi および md.ARCH ファイルに obsolete と記載のあるファイル名をパッケージの対象から削除する．
その後，basepkg は収集した情報を解析し，カテゴリ（例:base，etc など）ごとに FILES ファイルを作成する．それぞれの FILES はパッケージ名とファイル名の対応関係を記録している．

(2) 一時的なメタデータの作成．
basepkg は前述の FILES を読み込み，必要なディレクトリを生成する．その後，各パッケージの PLIST ファイルを作成する．このファイルはパッケージの中身の一覧を持つ．

(3) pkgsrc が利用するメタデータの生成．
basepkg は sets/essential を元に適切な+PRESERVE ファイルを生成する．これは削除してはいけないパッケージであることを示すメタデータである．これは該当するパッケージのみで生成される．
その後，各パッケージについて，pkgsrc フォーマットに必要なメタデータを生成する．+BUILD_INFO ファイルはパッケージ作成時の環境情報，+CONTENTS はパッケージに含まれるファイル一覧，+DESC と+COMMENT は簡単な表示である．
パッケージによっては，+INSTALL と+DEINSTALL ファイルも生成する．+INSTALL と+DEINSTALL および前述の+CONTENTS はパッケージの追加・削除時に，フックとして利用できるメタデータである．

(4) pkg_create(1) コマンドを利用してパッケージを作成する．
前述のメタデータと，DESTDIR(図 2) ディレクトリにあるバイナリを元にパッケージが作成される．

(5) チェックサム（MD5 および SHA512）を生成する．

なお，basepkg の動作検証は，既存のアーカイブファイルと同じものが生成されることで確認している．つまり，既存の配布物 base.tgz と，basepkg が作成した base-*.tgz ファイル群の総計は同じものになっている．

5. Basepkg の使い方

basepkg にはベースパッケージを作成して配布するという管理側の側面と，ベースパッケージを更新するというユーザ側作業の側面がある．basepkg のインストールが必要なのは管理側の話だ．一般ユーザは特別なツールも不要で，pkgsrc と同様の追加や削除の操作をするだけでよい．

[*8] syspkg の attr ファイルが，これに相当するが，不完全な状態である．

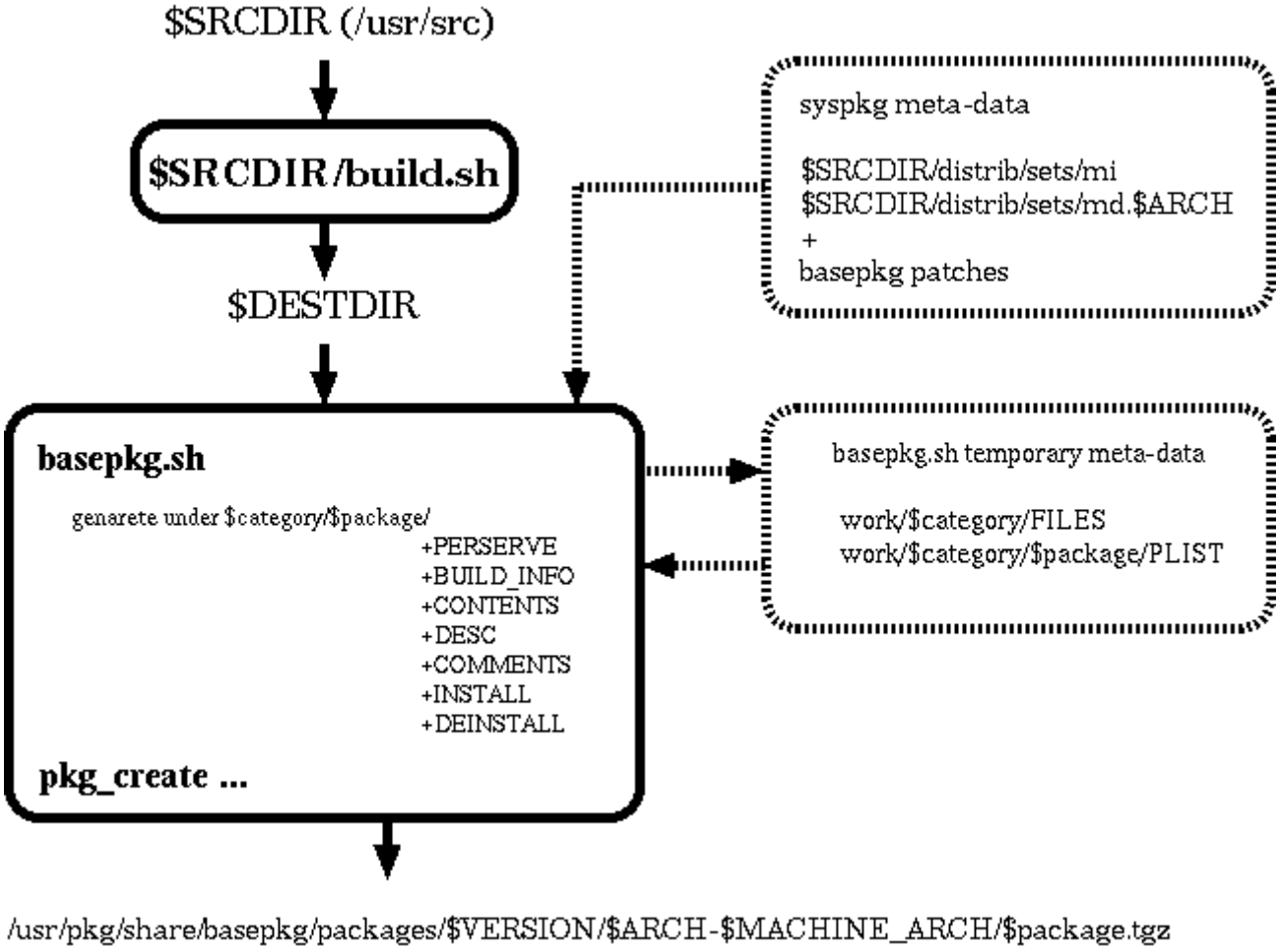

図 2 Basepkg の内部処理の様子
Fig. 2 Basepkg processing internals

5.1 Basepkg のインストール

basepkg のインストールは，github [*9] と pkgsrc-wip[*10] の二通りの方法から選択できるが，basepkg は pkgsrc (7) の pkgtools/pkg_install を前提としているため，こういった依存関係も自動解決してインストールをおこなう pkgsrc-wip の利用を推奨している．

pkgsrc-wip でのインストール方法は，3 節などと同様に make を実行すればよい．

```
# cd pkgsrc/wip/basepkg
# make install
```

5.2 ベースパッケージの作成・配布

DESTDIR にベースシステム一式が必要であるので，事前に build.sh release などを実行し用意しておく（3 節を参照）．

pkgsrc-wip を利用している場合，デフォルトで basepkg は /usr/pkg/share/basepkg ディレクトリにインストールされているので，ここに移動し，basepkg.sh を実行すれば，配下の packages/[NetBSD_version]/[MACHINE]-[MACHINE_ARCH] にパッケージが生成される．NetBSD-7.1/amd64 の場合，/usr/pkg/share/basepkg/packages/7.1/amd64-x86_64 となる．

```
# cd /usr/pkg/share/basepkg
# ./basepkg.sh pkg
# ./basepkg.sh kern
```

この作業は，各ユーザが（独自に自システムを細かい粒度で保守したいのであれば）各自のシステムで実行してもよいが，本来は，ベースパッケージを作成・配布する運用側がソースコードのアップデートのたびに一度だけ実行する作業である．各ユーザでは，次の 5.3 節のようにベースパッケージの入れ替えのみを行う運用が想定されている．

5.3 ベースパッケージの更新

basepkg によって作成されたパッケージは pkg_add (1) を用いてシステムへインストールできる．pkgsrc (7) でインストールしたパッケージと区別するため，データベースを分けてインストールすることを推奨する．たとえば

[*9] GitHub を使う場合，https://github.com/user340/basepkg/releases からアーカイブをダウンロードして使う．
[*10] git://wip.pkgsrc.org/pkgsrc-wip.git を pkgsrc/wip に git clone する．

base-sys-usr パッケージをインストールする際には，データベースを指定する-K オプションをつけて次のように実行する．

```
# cd ./packages/7.1/amd64-x86_64
# pkg_add    -K /var/db/basepkg base-sys-usr
# pkg_delete -K /var/db/basepkg base-sys-usr
```

注意点として，etc-からはじまる名前のパッケージに対しては/etc 以下のファイルの上書きを避けるために-p オプションを付け，一時ディレクトリへ中身を展開後，+INSTALL スクリプトで/etc 以下のファイルに反映させなければならない．

```
# pkg_add -K /var/db/basepkg -p /tmp etc-sys-etc
```

本来，このような生の操作を各ユーザがするべきではなく，操作を支援するユーティリティが必要である（7.2.1 節）．

6. syspkg との比較

basepkg にはベースパッケージを作成して配布するという管理側の側面と，ベースパッケージを更新するというユーザ側作業の側面がある．それぞれについて評価をおこなう．

6.1 ベースパッケージ作成・配布時における syspkg との比較

basepkg はシェルスクリプトのみによって書いたことで，コードは読みやすくなり，今後の保守運用性能がよくなったが，そのぶん syspkg に比べ実行時間が 4 割ほど余分に必要であり，コーディングには改善の余地が残されている．

basepkg は syspkg よりも少ないソースコード行数で実装できた．basepkg.sh 1.2 は 1190 行のシェルスクリプトである．一方，syspkg は，src/distrib/syspkg 内に存在するすべての Makefile の合計が 1008 行，src/distrib/sets/ 内に存在するすべてのシェルスクリプトおよび awk スクリプトの合計が 2721 行，合計 3729 行のスクリプト言語と Makefile から構成されている．

またソースツリー内のディレクトリ数も大きく異なり，basepkg の 45 個に対し，syspkg は 872 個である．したがって，ソースツリー全体の把握もしやすい．

しかしながら実行速度には差がある．Intel Xeon CPU 2.40GHz，RAM 8GB の NetBSD 7.1 amd64 にて，build.sh syspkgs と basepkg.sh pkg を，それぞれ 5 回ずつ実行した平均時間の比較が**表 2** である．basepkg は syspkg に比べ 4 割程度実行が遅いことになる．これは，現在の basepkg のコーディングスタイルがシェルスクリプトを手続き型言語のように使っているためかもしれない．ストリーミング型の記述 [5] に置き換えなどコーディングを改良していくことで，実行時間の改善が見込めると考えている．

プログラム	実行時間（秒）
build.sh syspkgs	850.4
basepkg.sh pkg	1188.4

表 2 build.sh syspkgs と basepkg.sh pkg の平均実行時間の比較
Table 2 Comparision of the average processing time betwen build.sh syspkgs and basepkg.sh pkg

6.2 ベースパッケージを利用したシステム更新時における syspkg との比較

従来の更新方式は，大きなアーカイブをダウンロードして展開（上書き）する方法だが，新方式の場合，各ベースパッケージそのものが小さく，細かい粒度での更新になるので，当然，更新も迅速になると期待される．ただし，パッケージ管理は，依存関係の解決[*11] など新たなオーバヘッドも導入するため，更新サイズに比例して高速化するとは限らない．

そこで，新旧両方式の更新速度を比較した．動作中のシステムに影響を与えないため，対象カテゴリは games である．旧方式とは，games.tgz をダウンロードしてカテゴリ全体を展開（上書き）する手法である．一方，我々の新方式では，配布サーバから該当するパッケージをダウンロードし pkg_ツールによる更新（追加・削除）を行う．パッケージ間の依存関係を擬似的に再現するため，インストール対象の数を 1 から 4 まで変化させて実効速度の変化を見る．なお，ベースパッケージの games-*.tgz ファイル群のすべてが games-games-root と games-sys-root に依存するため，この二つは除き，残りの 11 個の games-*.tgz ファイル群について総当たり戦（${}_{11}C_1$，${}_{11}C_2$，${}_{11}C_3$，${}_{11}C_4$ ）を行い評価した．

クライアント側は学内に設置された Xen で仮想化した Intel Xeon 2.4GHz，RAM 4GB の NetBSD-7.1 amd64 である．配布サーバには，われわれが試験運用しているベースパッケージ配布サーバ basepkg.netbsd.fml.org を利用した．

ネットワーク環境は次のとおりで，極めて高速かつ安定した環境といえる．大学は IIJ[*12] 専用線接続で，games.tgz のダウンロード先 ftp.jp.netbsd.org も IIJ にホスティングされている．basepkg.netbsd.fml.org は，さくらインターネット[*13] の大阪にある VPS で，さくらインターネットと IIJ は大阪で接続している．

結果を**表 3** にしめす．一つのカテゴリすべてを上書きインストールするのであれば旧方式が速い．一部を更新するだけであればパッケージ方式が速いことがわかるが，依存関係が 5〜6 あたりで旧方式と同程度になってしまうと見積もられる．しかしながら，メタデータ+CONTENTS 内の

[*11] openssh は openssl ライブラリに依存し，ビルドには perl が必要であるといったぐあいである．
[*12] https://www.iij.ad.jp/
[*13] https://www.sakura.ad.jp/

タスク	実時間（秒）
旧方式	5.3231
新方式 ${}_{11}C_1$ 通りのインストール	1.1976
新方式 ${}_{11}C_2$ 通りのインストール	1.8055
新方式 ${}_{11}C_3$ 通りのインストール	2.4476
新方式 ${}_{11}C_4$ 通りのインストール	3.2794
新方式（games-*すべて）	19.2955

表 3 各手順の平均処理時間の比較

Table 3 Comparison of the average processing time

@pkgdep 行から依存関係数を調べると，8 割以上のベースパッケージで依存関係が 3 以下となっている．よって，依存関係を解決するオーバヘッドを考慮しても，たいていの場合，新方式のほうが速度向上が見込める．それだけでなく，新方式であれば，パッケージの入れ替えと同時にサービス（デーモン）を再起動するといった旧方式にはない機能が提供可能となるため，ユーザの利便性が向上する点が重要である．

7. 議論と今後の課題

7.1 ベースパッケージ作成・配布時の課題

7.1.1 配布サーバ運用体制

多くの Linux ディストリビューションは，ビルドされたパッケージをサーバ上で配布しており，簡単な操作で最新の状態へ更新することができる．

一方，NetBSD では，そういったサービスを提供していないため，実環境での評価およびユーザの利便性のために，我々は試験的にベースパッケージ配布サーバを運用している．

しかしながら，移植性を重視する NetBSD はサポートするアーキテクチャも多く，複数ブランチの全アーキテクチャをビルド（5.2 節）するには非常に時間がかかる．セキュリティアップデートが必要な際に，現実的な時間でベースパッケージを準備するには，相当ハイスペックなマシンが必要だが，NetBSD では `build.sh` の導入以降，全自動でビルドテストしつづけるシステム（以下 daily build）が運用されているので，この daily build を最大限利用するべきだろう．最近では，すくなくとも最新の安定版[*14] は毎日ビルドされている．

正式にシステムの一部となれば daily build に統合されるかもしれないが，現状では daily build のバイナリからベースパッケージを作成する仕組の構築と運用が現実的な路線である．

6.1 節で述べたように，basepkg は syspkg に比べ 3 割程度実行が遅いが，この処理はベースパッケージを作成して配布する管理側の課題で，しかも，現実的運用では，ソースコード更新時の一度だけしか行われない処理である．その一方，ベースパッケージの追加・削除は多くのユーザが行う処理であるから，ベースパッケージ配布サーバを運用することで，NetBSD ユーザコミュニティ全体としての利便性は向上すると期待される．

7.1.2 basepkg データベースの保守・管理

`src/distrib/sets` には，従来方式で配布セットを作る情報と `syspkg` のメタデータの双方が含まれている．そして `basepkg` は `src/distrib/sets` にある `syspkg` のデータベースに依存している．ソースコード管理システムのログから判断するに，配布セット情報は保守されているが，`syspkg` 関連のメタデータは保守されているとは言い難い．実際，従来方式と `basepkg` 方式で作成物の総計が同じになることは確認している（4.3 節）ので，`basepkg` の配布情報も配布生成物も正しいと考えられるが，`syspkg` メタデータの不完全さに起因する問題は多く存在する．7.2.2 節で後述するように，一般ユーザにとって，ベースパッケージの命名規則や分類分けの妥当性は不明瞭であり，パッケージの記述（`comment` と `descr`）には欠落もある．パッケージの追加や削除動作に直接関係があるわけではないものの，ユーザの利便性を考慮するなら，これらのメタデータの修正作業も必要である．また，それらの修正作業は，`pkgsrc` の命名規則や分類分けに近づける方針がよいと考える．

7.2 ベースパッケージを利用したシステム更新時の課題

7.2.1 フールプルーフな一般ユーザ向けインタフェース

5.3 節で述べたように，`pkg_`ユーティリティを利用してベースパッケージの追加や削除をする際，オプションの指定や上書きを避ける工夫などがユーザに求められ，およそフールプルーフではない．これらの作業を自動でおこなう一般ユーザ向けインタフェースを提供するべきである．

また，デーモンプログラムのパッケージをアップデートした際には，自動でそのデーモンを再起動しなければ，古いデーモンがシステム中で動き続け，セキュリティホールが残ってしまうことになり，アップデートした意味がない．これら再起動が必要なパッケージ群については，`+INSTALL` にデーモンを再起動させる処理を加えるなどメタデータの修正・拡張が必要である．

7.2.2 ベースパッケージの命名規則

現状，`basepkg` のパッケージ名は `syspkg` のものを引き継いでおり，ある一定の命名規則に従ってパッケージの名前が命名されている．たとえば，ディレクトリ階層をまとめたパッケージであれば “root” プレフィックスがつき，バイナリをまとめたパッケージであれば “bin” などとなる．

[*14] 現在 NetBSD 7.1.1 が最新リリースで，最新の安定版は CVS 上の netbsd-7 ブランチでメインテナンスされている．このブランチについては毎日ビルドされている．一つ前の netbsd-6 などになると数日おきである．netbsd-7 は毎日ビルドされているため，セキュリティパッチが反映されたバイナリがせいぜい一日遅れで入手可能である．最近の NetBSD Security Advisory では，“各ユーザが daily build から日付をみて該当するアーカイブ（例:base.tgz）をダウンロードし，上書きする” 手順が書かれている．

しかしながら，一般ユーザにとって分かりにくいものも多く改善が必要である．たとえば，セキュアなリモートログインに用いられる OpenSSH は，OpenSSH プロジェクトの配布物も一般的に知られている名称も OpenSSH ないしは ssh であるにもかかわらず，syspkg では base-secsh-bin というパッケージ名になっている．この “secsh” という名前から OpenSSH を連想することは困難である．一方，pkgsrc (7) は一般的に通じる security/openssh を採用している．一般ユーザに分かりやすいよう，パッケージ名は base-secsh-bin ではなく base-openssh-bin などと変更するべきである．

命名規則だけでなく粒度にも修正が必要な例もある．たとえば，暗号ツールキット OpenSSL の共有ライブラリ libssl.so は，base-crypto-shlib[*15] というパッケージに登録されているが，一般ユーザにとってわかりやすい命名規則は base-crypto-shlib ではなく base-openssl-shlib であろう．一方，コンパイル時に用いる OpenSSL のライブラリ libssl.a は comp-c-lib パッケージの内容物であるが，comp-c-lib は非常に多くの種類のライブラリを含んでおり粒度が大きい．つまり base-openssl-shlib の粒度と揃っていない．こういった粒度の統一性も課題である．

7.2.3 システムアップデートの自動化

7.1.1 節で述べたように，多くの Linux ディストリビューションは，システムの自動アップデートを可能にしているが，NetBSD ではサポートされていない．これを実装するためには，basepkg とベースパッケージ配布サーバ，ベースパッケージの脆弱性情報リストの維持管理が必要である．

pkgsrc (7) は脆弱性を自動チェックする仕組みをもっている．具体的には，pkg_admin (1) を用いて，NetBSD Project 公式の FTP サーバから脆弱性情報の一覧（pkg-vulnerabilities）をダウンロードし，インストールされているパッケージとの比較を行う仕組みである．basepkg は pkgsrc (7) を前提にしているため，ベースパッケージに対して同様の仕組みを用意すれば，同じような自動チェックが行えるようになる．用意するべきデータベースは pkg-vulnerabilities と同じフォーマットでよい．

また，NetBSD のベースシステムではなく pkgsrc に登録されているだけだが，Linux ディストリビューションの apt や yum のような自動パッケージアップデート機能の提供を目的としたソフトウエア pkgsrc/pkgtools/pkgin も開発されている．

これらの要素を組み合わせることで，システムの自動アップデートが可能になるだろう．/var/db/pkg と /var/db/basepkg の区分の問題や，etc カテゴリパッケージのインストールの問題など課題は残るものの，basepkg によるパッケージ化とパッケージ配布サーバ，pkgin との連携がおこなえるようになれば pkgin ひとつでシステムすべてのパッケージを管理できるようになると期待できる．

[*15] ちなみに libcrypt.a が crypt (3) など昔からある暗号関数群のライブラリ，libcrypto.a(o がついている crypto) が openssl の暗号関数群ライブラリである．

8. まとめ

我々は NetBSD ベースシステムをパッケージ化するシステム basepkg を開発した．basepkg は最大限 pkg_*を使う「単一のシェルスクリプト + （メタ）データベース」という構成になっている．basepkg は syspkg にくらべ小さくて見通しがよいプログラムであり保守していきやすいと期待される．それに加え，インストール制御用スクリプトの実装やデータベースの修正など，syspkg より細かな粒度での効率的なベースシステム管理機能を実装した．しかしながら，現実的な運用体制には至っておらず，運用していく上で不足している機能の実装や修正すべき不具合などを解決していく必要がある．

参考文献

[1] DiBona, C., Ockman, S. and Stone, M.(eds.): *Open Sources: Voices from the Open Source Revolution*, Oreilly & Associates Inc (1999). （倉骨彰訳：オープンソース・ソフトウェア　彼らはいかにしてビジネススタンダードになったのか，オライリー・ジャパン（1999）．

[2] Mewburn, L. and Green, M.: build.sh: Cross-building NetBSD, *Proceeding of BSDCon '03*, (online), available from ⟨https://www.usenix.org/legacy/event/bsdcon03/tech/full_papers/mewburn/mewburn.pdf⟩ (2003).

[3] Project, T. N.: syspkgs, The NetBSD Project (online), available from ⟨wiki.netbsd.org/projects/project/syspkgs⟩ (accessed 2017-12-31).

[4] Parkes, L.: Lots of broken syspkgs, Must Have Coffee (online), available from ⟨gnats.netbsd.org/cgi-bin/query-pr-single.pl?number=46937⟩ (accessed 2017-12-31).

[5] 松浦智之，大野浩之，當仲寛哲：ソフトウェアの高い互換性と長い持続性を目指す POSIX 中心主義プログラミング，デジタルプラクティス，Vol. 8, No. 4, pp. 352–360（オンライン），入手先 ⟨https://ci.nii.ac.jp/naid/170000148949/⟩ (2017).

[6] Gancarz, M.: *The UNIX Philosophy*, Digital Press (1994). （芳尾桂 訳：UNIX という考え方，オーム社開発局（2001））．

[7] Kernighan, B. W. and Plauger, P. J.: *Software Tools*, Addison-Wesley (1976). （木村泉訳：ソフトウェア作法，共立出版（1981）．

[8] Kernighan, B. W. and Plauger, P. J.: *The elements of programming style*, McGraw-Hill (1978). （木村泉訳：プログラム書法第 2 版，共立出版（1982））．

[9] Boswell, D. and Foucher, T.: *The Art of Readable Code*, Oreilly & Associates Inc (2011). （角征典訳：リーダブルコード　より良いコードを書くためのシンプルで実践的なテクニック，オライリー・ジャパン（2012））．