「マルチメディア，分散，協調とモバイル
(DICOMO2016)シンポジウム」 平成28年7月

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

松浦 智之[1]　大野 浩之[2]　當仲 寛哲[1]

概要：ソフトウェアは，より高度な要求，あるいは時代と共に変化する要求に応えるべく，絶え間なくバージョンアップを繰り返している．しかし多くのソフトウェアは，今自分の置かれた環境において求められる性能や機能を満たすことばかり偏重し，他の環境や将来の環境における互換性をあまり考慮していない．そこで著者らは，UNIX 系 OS が最低限満たすべきとした仕様をまとめた国際規格である POSIX に着目した．POSIX は現状で多くの UNIX 系 OS が準拠してるうえに，1990 年の初出以来，その仕様はほとんど維持されている．このような性質をもつ規格に極力準拠しながらプログラミングすることで，ソフトウェアは高い互換性と長い持続性を得られる可能性がある．そして著者らは，このようにして POSIX の仕様に極力準拠しながらプログラミングをする指針を具体的にまとめ，POSIX 中心主義と名付けた．本稿では，POSIX 中心主義としてまとめたプログラミング指針を提案するとともに，現在行っている互換性と長期持続性の検証について報告する．

POSIX Centric Programming to Make Software More Compatible and Sustainable

TOMOYUKI MATSUURA[1]　HIROYUKI OHNO[2]　NOBUAKI TONAKA[1]

1. はじめに

ソフトウェアに対する要求は日々高度複雑化している．ゆえに現在の多くのソフトウェアは，他者が提供する言語やライブラリ，モジュール，フレームワーク，ミドルウェア等のソフトウェア（以降，これらを依存ソフトウェアと呼ぶ）の利用無しには作成することはもはや困難である．

そして依存ソフトウェアもまた，日々高性能・高機能化している．基本的に互換性を保ちながらバージョンアップされることが多いが，互換性の失われたバージョンアップが行われたり，あるいはまったく別の競合ソフトウェアがリリースされたり，依存ソフトウェアがさらに依存しているソフトウェア（OS 等）の必要バージョンが上がったりすることもある．

そのためソフトウェア開発者は，自身のソフトウェアに何の変更を加える必要がなくても作り直しを迫られることがあり，一方ソフトウェア利用者は，再インストールや別の新たなソフトウェアを見つけることに苦労を強いられる．

ソフトウェアは，開発者の想定以上に長期間，また多くの環境で利用される場合が多いにも関わらず，互換性や長期持続性への配慮はあまりなされず，今求められる機能や性能を満たすことを偏重する傾向が強い．そして実際に，多く人々がソフトウェアの互換性や継続利用で不便な思いをしている．また，開発者が互換性の高いソフトウェアや，10 年，20 年の長きに渡って利用できるソフトウェアを作成しようと考えても，現在の依存ソフトウェアのもとでは実現が難しい．

本稿では，このような現状を打開するため後述する POSIX 中心主義を提唱し，ソフトウェアの高い互換性と長い持続性を可能とするプログラミング方式について述べる．

2. 背景

本研究は，ソフトウェアの互換性や長期持続性が低い現状に悩んでいた著者らがこの現状を打破するためにはどうすればよいかという，解決方法を模索することから始

[1] （有）ユニバーサル・シェル・プログラミング研究所
[2] 金沢大学 総合メディア基盤センター

まった．

ソフトウェアの互換性や長期持続性を高める方法を提案している既存の研究成果はないか探したものの，その目的に完全に合致したものは見つけられなかった．しかし，調査の過程で，ソフトウェアの互換性を高めるために過去さまざまな規格が提案されているという記事を発見することはできた [1][2][3]．

その中で示されていた規格の一つが POSIX であった．POSIX は，用語としての言及数が，CiNii 検索における論文件数や，主要な Web 検索サービスにおける該当ページ数において，他規格と比較すると 2016 年現在に至るまで圧倒的に多く，実際にソフトウェア開発現場でも今なお活発に耳にする用語である．したがってこの POSIX という規格こそが，ソフトウェアの互換性のみならず，持続性をも高める鍵になるという認識に至った．

3. POSIX 中心主義の概要

POSIX 中心主義とは，ソフトウェアの互換性や長期持続性を高めるために著者らが提唱するソフトウェアのプログラミング指針である．本節ではその概要を述べる．

3.1 基本指針と利用する言語

POSIX 中心主義におけるプログラミングの基本指針は，その名のとおり POSIX.1(IEEE Std 1003.1) 文書に記されている仕様を中心にプログラミングをすることである．「中心」とは，ここでは「極力準拠」という意味である．

POSIX の仕様に準拠したプログラムを作成することにすると，開発言語はシェルスクリプトまたは C 言語（C99）を利用することになる．その理由は単純で，POSIX で用意されているプログラム言語としてのコマンド（以下 POSIX コマンドと記す）に Perl や Ruby，Java といった現在よく利用される高級言語は存在せず，存在するものは Bourne シェル（sh コマンド）と C コンパイラ（c99 コマンド）だけだからである．

どちらを選択しても POSIX を中心としたプログラミングができることにはなるが，基本的にはシェルスクリプトを利用する．C 言語は低水準言語であり，バイトオーダ等のハードウェア構造を意識しなければならない一方，シェルスクリプトであれば，そのようなハードウェア依存は POSIX コマンドが吸収しているため，意識せずにプログラミングできることが理由である．

したがって POSIX 中心主義プログラミングとは，POSIX の仕様に準拠したシェルスクリプトを中心としたプログラミングということになるが，それにより以下に述べるいくつかの利点がある．

3.1.1 開発効率と処理効率の両立

シェルスクリプトは C 言語と比べて処理の遅さを指摘されるが，それは必ずしも正しい認識ではない．

確かにシェルスクリプトはインタープリタ型言語であるため，ステップ数が多いほど処理効率は悪化する．また各ステップに外部コマンドを起動する記述があればそれも大きな処理効率悪化につながる．しかし手続き型の書き方からストリーミング型の書き方に改めるように工夫すれば，ステップ数の増加を抑えられ，処理効率は大きく改善する．

例えば “file1.txt” から “file10000.txt” までのファイルのうち 3 の倍数の番号のものだけ消すという処理を行うコードの書き方を考えた場合，while 文と test コマンド（[）でループするのではなく，事前に AWK や sed で処理対象のファイル名を生成し，最後に xargs と rm コマンドで一括削除すればよい（図 1）．

■手続き型コーディング（ステップ数が多く処理効率が低い）

```
i=3
while [ $i -le 10000 ]; do
  file="file${i}.txt"
  rm -f $file
  i=$((i+3))
done
```

■ストリーミング型コーディング

```
awk 'BEGIN{for(i=3;i<=10000;i+=3){print i;}}'|
sed 's/.*/file&.txt/'                        |
xargs rm -f
```

図 1 シェルスクリプトの書き方の違い

データ処理においても同様にして分岐やループは最小限に抑え，POSIX コマンドをパイプで繋ぎながら積極的にデータ処理を任せる方針をとる（図 4 参照）．

このようにすればステップ数が抑えられるため，C 言語並の処理速度が引き出せる．なぜなら，外部コマンド起動によるコストの大半は最初に 1 回発生するだけになるうえ，POSIX コマンド自体のほとんどは C 言語で書かれており，それらのコマンドは起動後，C 言語の速度でデータ処理を行うからである．

そしてシェルスクリプトはコンパイルを要しない言語であり，業務プログラムで多用されるテキスト処理も記述が比較的容易である．したがって前述のように書き方に気をつければ，シェルスクリプトは開発効率と処理効率を両立できる．

3.1.2 互換性の増加

シェルスクリプトは OS 依存が激しいものと思い込まれることが多いが，それも正しい認識ではない．

OS 依存が激しいのは，依存性のあるシェル文法やコマンド，オプション等を知らずに使ってしまうからである．POSIX で規定されている範囲の仕様はほとんどの UNIX 系 OS が満たしており，その範囲で書かれたシェルスクリプトであれば，それらすべての UNIX 系 OS で動くため，

むしろ高い互換性を有している．

3.1.3 インストール作業コストの抑制

POSIX で規定されている範囲の仕様はほとんどの UNIX 系 OS が満たしている．そのようにして POSIX 準拠を謳っている OS であれば，どんなに最小構成のインストールをしてもその直後から POSIX の範囲で作成されたプログラムが動作する．

これは，他言語で書かれたプログラムのように予め依存ソフトウェアをインストールする必要もなく，ただプログラム一式をコピーすれば直ちに動作するということを意味している．C 言語ではなくシェルスクリプトで書くことで自身のプログラムをコンパイルする作業すら不要になる．よってソフトウェアのインストールで悩まされるという苦労を大幅に低減できる

3.1.4 メンテナンスコストの抑制

他のソフトウェアに依存するプログラムは，その依存ソフトウェアがバージョンアップすると正常に動作しなくなったり，起動しなくなったりすることがあり，継続利用したければその都度修正を迫られる．依存ソフトウェアが増えるほどそのリスクは高まる．

一方，POSIX の範囲の仕様のみに依存して作成されたプログラムにはこのリスクがない．OS 開発団体は POSIX 準拠を謳う限り，どんなにバージョンアップをしても POSIX の仕様は必ず満たすからである．よって，OS に何らかの脆弱性が発覚したり，現バージョンの OS のサポートが切れるなどによりバージョンアップを迫られる場合でも何も心配せずバージョンアップができる．

もし現在利用中の OS のサポートそのものが打ち切られることになったとしても，POSIX 準拠を謳う他の OS が現状では潤沢に存在するので，そちらにプログラム一式をコピーすればよい．

3.2 3 つの指針

POSIX 中心主義というプログラミング指針はさらに，図 2 に記した 3 つの小指針から構成される．これらは作成するアプリケーションに必要とされる機能に応じて使い分け，あるいは組み合わせる．

3.2.1 POSIX 準拠

「POSIX 準拠」は POSIX 中心主義における原則的な指針であり，すべてのプログラムは問題がなければこの指針を守りながらプログラミングされなければならない．基本指針は，名称が示すとおり POSIX で規定されている仕様に準拠することである．具体的には，単独の UNIX ホスト上で動かすプログラムや，クライアントサーバ構成をとるアプリケーションにおけるサーバ側プログラムに適用する．

3.2.2 交換可能性担保

「交換可能性担保」は，POSIX の範囲では実装が難しい機能を実現するため，POSIX 準拠指針に対して例外的に追加する指針である．具体的には，外部の Web API を利用する必要があったり，メールを送受信する必要のあるプログラムなどに適用する．具体的な指針は，ネットワーク操作のためのコマンドなどが十分に用意されていない POSIX コマンド群のために，互換性や長期持続性を損なわないようにするための条件付きでそれらを実現するコマンドの仕様を認める．

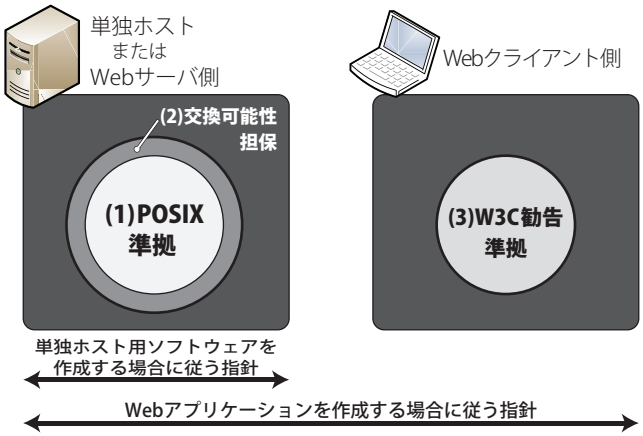

図 2　POSIX 中心主義を構成する 3 つの小指針と適用範囲

3.2.3 W3C 勧告準拠

「W3C 勧告準拠」は，クライアントサーバ構成をとるアプリケーションにおけるクライアント側プログラムに適用する．2016 年現在，クライアント端末としてのプラットフォームの主流は実質的に Web（HTML/CSS/JavaScript 等の規格の集合）であるが，Web プラットフォームには POSIX 準拠指針が適用できない．しかし，Web にもやはりブラウザ間の互換性確保を目的として下位互換を重視しつつ多くのブラウザ開発団体が準拠している W3C 勧告があるため，これに準拠することでブラウザ上のプログラムにおいても POSIX 準拠と同じ目的の達成に努める．

4. POSIX 中心主義のの詳細

本節では POSIX 中心主義の実践方法の詳細を述べる．

4.1 POSIX 準拠指針の詳細

POSIX 準拠におけるプログラミング上の基本的な指針は，POSIX.1(IEEE Std 1003.1) 文書に記されている内容に従うことである．特に問題が無い限りはこの文書に記されている文法やコマンド，オプション，正規表現，期待される出力結果のみに依存したソースコードを書かなければならないものとする．この文書は Web ページとして公開されており [11]，“opengroup posix”などのキーワードで検索可能である．従ってプログラマは常にその内容を確認しながらプログラミングができる．

4.1.1 POSIX の一部にある非互換・曖昧な仕様への対応

POSIX という規格は，UNIX 系と称される OS が 1980 年代までに各々独自に機能拡張をして互換性が低下してい

るなか，OS 間の最低限の互換性を確保するため 1990 年に初めて発表されたものである．基本的には OS 間で共通する仕様を抜き出した内容になってはいるが，実装間で互いに両立できない仕様に関してそのうちの 1 つを選択せざるを得ず，結果的に互換性が確保できないものもある．tr コマンドがそのような例の一つである．例えば“0”から“9”の文字範囲を指定したい場合，System V 系の実装では“[0-9]”と記述しなければならないが，BSD 系の実装では“0-9”と記述しなければならない．そして POSIX では後者の書式が採用されているが，本指針における POSIX 準拠では POSIX 文書に書かれているからといって後者の記述を推奨はしない．System V 系，BSD 系どちらでも通用するように“0123456789”のように範囲指定記号を使わない記述や，そういう問題の生じない sed コマンドで代用することを推奨する．

また，POSIX で明文化されきれていない細部の仕様もあり，OS によって動作が異なることもある．そのようなものに関しては，OS に依存しないように POSIX 文書を見ながら気を付けなければならない．

このような細部の非互換・曖昧な仕様に関し，著者らは，情報として整理しながら公開しており [4][6][7]，新たな知見が得られる都度更新している．

4.1.2 POSIX コマンド群の使いづらさへの対応

POSIX コマンドは UNIX 系 OS 間の互換性確保のため，ホスト操作に用いる最低限のものしか用意されていない．ゆえに，アプリケーション開発で多用される処理を行うコマンドが十分に揃っておらず，実際の開発には不便である．

そこで著者らは，開発によく用いられる処理を POSIX コマンドを組み合わせたシェルスクリプトで独自に作り置き，普段はこれを組み合わせてプログラミングすることで開発効率を向上させるようにしている．作るべきコマンドとして参考にしているものはシェルスクリプトによるシステム開発手法である「ユニケージ」[5] を提案している USP 研究所がリリースしている usp Tukubai コマンド [8] である．

POSIX には join という表を内部結合・外部結合するコマンドがもともと存在するうえに，usp Tukubai コマンドにはリレーショナルデータベース（RDB）処理に便利なコマンドが用意されているため，usp Tukubai を POSIX 準拠で移植することで RDB を要するアプリケーションをも POSIX 準拠しながら開発できる．

図 3 は，全会員の名前と ID が格納されているテーブルからブラックリストに登録されている会員 ID を除外して表示する処理を SQL 文で記述した例であるが，図 4 のように同等の処理を POSIX コマンドで記述できる．

これに加え，現代の Web アプリケーションで多用される処理（JSON や XML 解釈，URL エンコーディング，Cookie リード・ライト他）を Tukubai コマンドのインターフェースに倣って独自に作成もしている．

```
SELECT
  MEM."会員 ID",
  MEM."会員名"
FROM
  blacklist AS MEM
    RIGHT OUTER JOIN
  members   AS BL
    ON BL."会員名" = MEM."会員名"
WHERE
  BL."会員名" IS NOT NULL
ORDER BY
  MEM."会員 ID" ASC;
```

図 3 RDB 的操作の例（SQL で記述）

```
cat blacklist.txt |
# 第 1 列:BL 会員 ID#
sort -k 1,1       | ←会員 ID で並替え
uniq              >sorted_bl.txt

cat members.txt                        |
# 第 1 列:会員 ID 第 2 列:名前          #
sort -k 1,1                            | ←会員 ID で並替え
join -1 1 -2 2 -v 2 sorted_bl.txt -    ← BL の会員 ID で
                                         join できない
                                         のみを抽出
```

図 4 RDB 的操作の例（POSIX の範囲内で記述）

これらの結果，著者らは POSIX に準拠しながら，開発に有用な多くのコマンドを用意できた（表 1）．

表 1 POSIX 準拠しながらも実現させた主なコマンド

名称	機能
calclock	日常時間と UNIX 時間を変換する
zen/han	全角文字・半角文字を変換する
sm2	表の値合計を求める（sum-up）
mktemp	一時ファイルを作る（同名コマンドを POSIX 準拠で実装）
parsrj.sh	JSON テキストをパースする
parsrx.sh	XML テキストをパースする
cgi-name	Web ブラウザから送られた CGI 変数を受け取る（Cookie 変数受け取りにも流用可）
pexlock	排他ロックをかける
pshlock	共有ロックをかける
base64	Base64 エンコーダ・デーコーダ（GNU Coreutils からの移植）
uelencode	URL エンコード（RFC3986）する
mkcookie	Cookie 文字列を作る
mime-make	MIME マルチパートデータを作成する（ファイルアップロードや添付ファイル付メール向け）
sessionf	セッション管理をする（Web アプリケーション向け）

これらを含め，独自作成したコマンドは，パブリックド

メイン（CC0）にて Web 上で公開しており [9][10]，かつ追加コマンドや修正があれば随時更新している．

4.2 交換可能性担保指針の詳細

POSIX には sed や AWK コマンドなどのチューリング完全なコマンドが含まれるため，いかなる計算をも記述できる．しかし，不得意な処理もある．

1 つはバイナリ処理である．POSIX コマンド群は行指向・列指向のテキスト処理には向いているがバイナリデータやビット演算を効率的に行えるものがほとんどない．テキストデータに変換することで実現することもできるが，実用的な処理速度が出ない．もう 1 つはネットワーク処理である．POSIX コマンドでネットワーク（INET ドメイン）に通じているものは唯一 mailx というメール送受信コマンドであり，現在主流の Web（HTTP）を扱うことはまったくできない．

そこで，一定条件のもと POSIX にないコマンドの利用を認める．その条件が交換可能性担保である．

4.2.1 交換可能性

交換可能性は次のように定義する．

> 今利用している依存ソフトウェア（A）と同等機能を有する別の実装（B）が存在し，何らかの事情により A が使えなくなった時でも，B に交換することで A を利用していたソフトウェアを継続して使える性質

POSIX 準拠という指針を定めた理由も，交換可能性を担保するためである．なぜなら，POSIX 規格に準拠した OS は多くの開発団体が各々実装しており，どれか 1 つがサポート打ち切りやセキュリティその他の問題で使えなくなったとしても他実装へ容易に移行可能であることが担保されているからである．

そこで POSIX 準拠指針では実現が難しい処理については，交換可能性を担保しながら POSIX 外のコマンドの利用を認める．

4.2.2 交換可能性担保の具体例

4.2.2.1 同名の互換コマンドが存在する場合

例えばメール送信やデータの暗号化に関してはそれぞれ sendmail コマンド，openssl コマンドの利用をそのまま認める．これらのコマンドは同名で互換性を持たせた別実装が存在するからである．sendmail コマンドは，メール転送エージェント（MTA）としての sendmail の他，Postfix や qmail，exim など多くの MTA が用意している．一方 openssl コマンドは，2014 年に発覚した Heartbleed 脆弱性の教訓として LibreSSL という別実装が登場したことにより交換可能性が担保できるようになった．

ただし，sendmail コマンドも openssl コマンドもオリジナル版が持つすべてのオプションが移植されているわけではないため，交換可能性を確保するためには使ってよいオプションに注意が必要である．

4.2.2.2 別名の互換コマンドが存在する場合

例えば Web（HTTP）アクセスを行いたい場合は，Web アクセスに対応する 2 つのコマンド wget，curl に両対応するようにコードを記述することでそれらの利用を認める．

別名のコマンドは基本的に書式が異なるため，コマンドの存在確認を行ったうえで if 文等を使って分岐させ，同じ処理を複数の記述する．

同名コマンドの時と同様に使ってよいオプションに注意が必要である．例えば curl コマンドには Web フォーム送信やファイルアップロード用の“`-F`”オプションがあるがこれは使ってはならない．wget コマンドには相当する機能が無いからである．これらの処理を行いたい場合は，表 1 にも記した urlencode や mime-make コマンドを使って生成する．こうすれば wget，curl コマンドそれぞれ“`--post-file`”，“`--data-binary`”オプションにより送信できる．

4.3 W3C 勧告準拠指針の詳細

2016 年現在，一般ユーザに対する操作画面はクライアント端末上の Web ページとして作成するケースが主流になっており，アプリケーション開発において Web（HTML/CSS/JavaScript）は必要不可欠な存在になっている．Web の規格は POSIX とはまったく異なる発展を遂げてきたが，POSIX に似た事情を抱えている．

UNIX 系 OS は各実装が独自に機能を拡張し，互換性が低くなってしまった状況を改善すべく POSIX 規格が策定されたが，Web では各 Web ブラウザが独自に機能を拡張し，互換性が低くなってしまった状況を改善すべく W3C という組織が設立され，勧告（W3C 勧告）がアナウンスされるようになった．

そこで，クライアント端末向けに Web プログラムを作成するにあたっては W3C 勧告に準拠するという指針を設ける．W3C 勧告もまた Web 上で閲覧でき [12]，プログラマは常にその内容を確認しながらプログラミングできる．

4.3.1 各種 JavaScript ライブラリ使用の禁止

W3C 勧告準拠の指針の下では jQuery 等に代表される各種 JavaScript ライブラリを使わない．W3C 勧告の範囲の API をのみを使い，一からプログラムを書く．後述する XMLHttpRequest に関しても汎用的な処理は 40 行程度で記述でき [18]，一度書けばあとは他のプログラムにも流用可能であるため，以後はコピーして使えば開発効率をさほど低下させることはない．

ライブラリを使わない理由は，そのライブラリが各 Web ブラウザの独自機能を利用していないという確証がないからである．プログラムでそのようなライブラリを利用すると，将来の Web ブラウザでは正常に動作しなくなる可能性が高まる．

4.3.2　ドラフト仕様の利用

W3C 勧告の掲載されているページには勧告になる前のドラフト仕様も存在する．ドラフト仕様は勧告になる前に変更されたり廃止されたりする可能性があるので極力利用すべきではない．しかし，既に主流となっている仕様の 1 つである XMLHttpRequest は 2016 年 5 月現在もドラフトである．

このような仕様の利用は，普及度を鑑みながら最低限の利用に留めるべきである．

5.　POSIX 中心主義の効果検証

POSIX 中心主義は，もともと著者らが自身の開発するソフトウェアの互換性や持続性を高めたくてまとめたプログラミング指針であり，業務アプリケーションの作成等で既に実践しているが，どれくらい効果的であるかを検証するためのソフトウェアを作成している．

結論（特に長期持続性の検証するための）を得るには長年に渡る検証が必要ではあるが，本指針を提案しはじめてから約 2 年が経過した現時点での状況を報告する．

5.1　互換性の検証

POSIX 中心主義の指針に従えば，多くの OS 上でそのまま動作するソフトウェアを開発できることを実証する目的で作成したものの 1 つに「恐怖！小鳥男」（以下小鳥男と記す）[13] がある．これは POSIX 準拠，および交換可能性担保の指針に基づいて作成されたシェルスクリプト製 Twitter クライアントである（図 5）．

図 5　シェルスクリプト製 Twitter クライアント「恐怖！小鳥男」

5.1.1　小鳥男の概要

小鳥男は CUI 型の Twitter クライアントプログラムである．ツイートの投稿や検索，フォロー等，日常 Twitter 上で行う操作が一通り行えるが，内部的にはどの操作も，1)OAuth 認証文字列を生成し，2)Twitter Web API にリクエストを出し，3)JSON 形式で返されるレスポンスデータを解釈するという手順からなる処理である．

手順 1) では暗号化のために openssl コマンド，手順 2) では Web アクセスのため curl または wget を使用するが，手順 3) の JSON データ解釈を含めてその他の処理は POSIX コマンドの範囲で実装できる．したがって，小鳥男は OpelSSL（または LibreSSL）と cURL（または Wget）という 2 つの POSIX にはないソフトウェアを要するものの，これらは多くの UNIX 系 OS でプリインストールされていたり，あるいは容易にインストールできるため，それらさえインストールされていればあとは小鳥男の tar アーカイブファイルを展開するだけで利用可能になる．

したがって小鳥男は，既存の主要な CUI 型 Twitter クライアント（T[14] や termtter[15] 等）と比べ，インストールが極めて簡単である．

5.1.2　小鳥男の動作実績

POSIX 準拠および交換可能性担保という指針に基づいて作成された小鳥男は，これまでに表 2 に記した環境で動作確認，あるいはその報告がなされている．

表 2　恐怖！小鳥男の動作確認がとれた環境

名称	補足
CentOS	Linux 系，5.3，5.9，6.5
Raspbian	Linux 系，Debian Jessie ベース，2015-09-28 版
FreeBSD	9.1-RELEASE，10.2-RELEASE
AIX	7.1.0.0
Mac OS X	Mavericks (10.9)
Cygwin	64bit（2016-01-24）版 *1
gnupack	Cygwin 系，devel（2015.11.08）版 *1
Ubuntu on Windows	Linux 系，Windows 10 Insider Preview（ビルド 14316）版

*1 Cygwin 向けの専用コードを追加した．

これまでのところ，Cygwin 系以外の環境には，特にそれらの環境を意識したコードを追加することなく，作ったプログラムがそのまま動作している．プログラミング時に注意したことは POSIX 中心主義の指針を守ることのみである．

ただし Cygwin とその派生環境である gnupack では，一箇所だけ専用のコードを追加しなければならなかった．これは Cygwin 系環境が Windows 上に構築されているという理由により，ps コマンドの仕様が POSIX とは異なるという事情による．

この動作実績は，POSIX 中心主義プログラミングによって作成されたソフトウェアの互換性の高さを示している．

5.2　長期持続性の検証

POSIX 中心主義に従って作成されたソフトウェアが長期持続性を有することを実証するため，東京地下鉄株式会社（東京メトロ）のソフトウェアコンテストに，POSIX 中心主義に基づいて作成したソフトウェアを応募した．

5.2.1　東京メトロ主催「オープンデータ活用コンテスト」

2014 年，東京メトロは設立 10 周年を記念していくつかの記念行事を開催した [16]．そのうちの一つが，自社の鉄道に関するデータを一般の人々に公開し，そのデータを活用するアプリケーションの素晴らしさを競う「オープンデータ活用コンテスト」の開催であった．

このコンテストは，2012 年のロンドンオリンピックに向けてロンドン地下鉄が自社のデータをオープンデータ化し，観光客に向けて優れたアプリケーションを集めることに成功した事例を参考に，2020 年の東京オリンピック・パラリンピックを見据えて企画された [17]．

筆者らはその意図を汲み取り，コンテスト開催から 6 年後の東京オリンピック本番時でも動作し続けることを目標にしたソフトウェアを作成した．

5.2.2　コンテスト応募作品「メトロパイパー」

応募した作品は「メトロパイパー」である（図 6）．これは駅施設にある発車標（列車の行先や発車時刻，接近情報などを表示する電光掲示板）をイメージしたものであり，駅と方面を指定すれば，どの行先の列車が何分後に到着するのかを，東京メトロがオープンデータとしてリアルタイムに公開している列車在線位置情報を取得して Web ブラウザ上に表示する．

W3C 勧告の指針に従い，その範囲におけるレスポンシブルデザインが施されており，スマートフォンのような横幅の狭い画面であることを感知すればそれに適したレイアウトに自動で切り替わる設計になっている．すなわち，POSIX 中心主義プログラミングでありながらモバイルフレンドリなソフトウェアになっている．

本作品はソースコードと共に今も公開している [18]．

図 6　東京五輪までの動作を目指したソフト「メトロパイパー」

5.2.3　コンテストの結果とその後の経緯

コンテストの受賞作品は 2015 年 2 月に発表された [19]．ユーザインターフェースの優れた作品が受賞作として選ばれる傾向が強く，iPhone や Android 等のスマートフォンアプリケーションが上位を占める結果となった．

しかし，コンテストの応募締め切りから約 1 年半経過した現在，応募作品の動作状況を調査してみると，図 7 のとおりの結果になった．

作品の応募総数は 281 であったが，コンテストの公式ページから既に作品が抹消されていたり，作品を起動するためのページがデッドリンクになっているものが 150 あり，したがって既に半分以上がアクセスすらできなくなっていた．さらに，アクセスはできても正常に動作していないと思われる作品が 11 あり，これらを合わせると既に正しく利用できなくなっている作品数は 161 にも上る．この中には一部のコンテスト受賞作も含まれていた．

もちろんコンテストが終了したことで，維持管理の手間などから意図的に公開が終了された作品も多いだろうが，少なくとも 11 の作品は依存ソフトウェアの変化により動

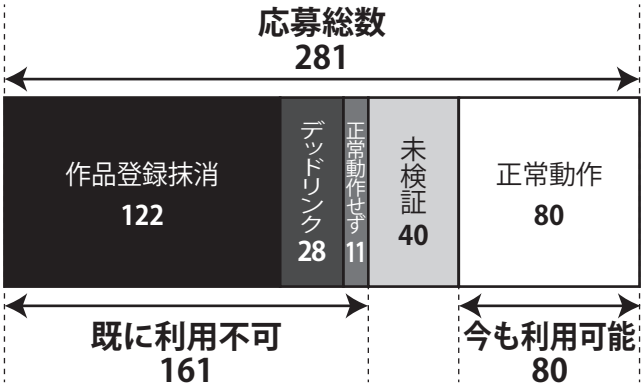

図 7　コンテスト締め切りから 1 年半後における作品の動作状況（2016/05/08 現在）

かなくなったと思われる．正常に動作していると思われる作品の数は，未検証のもの（著者らが所有していない端末向けの作品）がすべて正常動作すると仮定しても 120 で，わずか 1 年半で 4 割程にまで減っており，2020 年の東京オリンピックには僅かな数しか残らないものと予想される．

これは，ソフトウェアにとってはたった数年でも維持管

理がいかに手間であるかを示す結果である．これに対し，メトロパイパーは特にプログラムに一切修正を加えずに2016年5月現在も正常動作を続けており，今後も持続性の検証を続けていく予定である．

5.3 Windows 10 に搭載される POSIX 環境

2016年3月，Microsoftは同社OSのWindows 10にUbuntu on Windows（UOW）と呼ばれるUbuntu Linux互換システムを搭載すると発表した[20]．これは仮想OSではなく，Ubuntu実行ファイルから呼び出されるシステムコールをWindowsカーネルがリアルタイムに解釈して実行するという本格的なものである．WindowsにはSubsystem for Unix Applicationと呼ばれるPOSIX環境があったものの，上位エディションでしか提供されず，また追加ソフトウェアのインストールも難しく，実用性は高くなかった．

2016年5月現在，UOWのPreview版が利用可能で，表2のとおり小鳥男の動作も確認され，POSIX中心主義のための環境としても実用的なものと予想される．UOWが正式公開されれば，POSIX中心主義に適した端末が一気に数億台増えることになり，その意義は非常に大きい．

6. まとめ

本稿は，ソフトウェアの互換性と持続性を高めるためにPOSIX.1(IEEE Std 1003.1)文書に記されている範囲の仕様だけに極力依存してプログラミングをする，POSIX中心主義とよばれるプログラミング指針を提案した．POSIX規格に準拠をすると，開発言語としては実質的にシェルスクリプトを使うことになるが，処理速度や開発効率の面で決して非実用的ではないのみならず，互換性や保守性の面で現在主要な他言語と比較しても高い理由を述べた．そしてPOSIX中心主義というプログラミング指針はさらに，(1)POSIX準拠，(2)交換可能性担保，(3)W3C勧告準拠という3つの小指針から構成され，目的とするアプリケーションに応じて使い分けたり組み合わせるものであることを示し，それぞれの具体的内容について述べた．最後に，POSIX中心主義の効果について，互換性，長期持続性の両面について検証活動と，現時点では期待した結果が得られている旨を報告した．今後も検証を続けるとともに，本指針の一層の具体化を進める予定である．

謝辞

POSIX中心主義の研究は現在，USP研究所と金沢大学の共同研究の一環として進められています．本指針発案のきっかけはユニケージ開発手法であり，それを推進するUSP研究所の皆様に感謝すると共に，本指針を支持し，御指導いただいている金沢大学の共同研究者の皆様，特に本稿をご査読くださった北口善明助教，森祥寛助教に感謝いたします．

参考文献

[1] 越田一郎，OSインターフェース標準化の動向, 情報処理学会研究報告システムソフトウェアとオペレーティング・システム（OS） 1987(37(1987-OS-035)), 1-6, 1987-06-12

[2] 斎藤信男，UNIXの標準化とPOSIX，コンピュータソフトウェア 6(1), 84-92, 1989-01-13

[3] システムインタフェース検証研究グループ，システムインタフェース検証・認証の現状，情報処理 34(3), 324-335, 1993-03-15

[4] 松浦智之，すべてのUNIXで20年動くプログラムはどう書くべきか，C&R研究所，2015-07-25

[5] 中村和敬, 當仲寛哲，Unixシェルスクリプトによる企業システム構築，情報処理学会第77回全国大会，2A-01，(2015)

[6] richmikan@github，どの環境でも使えるシェルスクリプトを書くためのメモ (オンライン)，入手先 ⟨http://qiita.com/richmikan@github/items/bd4b21cf1fe503ab2e5c⟩（参照 2016-05-08）.

[7] richmikan@github，どのUNIXコマンドでも使える正規表現 (オンライン)，入手先 ⟨http://qiita.com/richmikan@github/items/b6fb641e5b2b9af3522e⟩（参照 2016-05-08）.

[8] USP engineers' community，usp Tukubaiについて (オンライン)，入手先 ⟨https://uec.usp-lab.com/TUKUBAI/CGI/TUKUBAI.CGI?POMPA=ABOUT⟩（参照 2016-05-08）.

[9] 秘密結社シェルショッカー日本支部，Tukubaiコマンドクローン (オンライン)，入手先 ⟨https://github.com/ShellShoccar-jpn/Open-usp-Tukubai/tree/master/COMMANDS.SH⟩（参照 2016-05-08）.

[10] 秘密結社シェルショッカー日本支部，misc-tools (オンライン)，入手先 ⟨https://github.com/ShellShoccar-jpn/misc-tools⟩（参照 2016-05-08）.

[11] What is POSIX?, The Open Group (オンライン)，入手先 ⟨https://collaboration.opengroup.org/external/pasc.org/plato/⟩（参照 2016-05-08）.

[12] World Wide Web Consortium，All Standards and Drafts (オンライン)，入手先 ⟨https://www.w3.org/TR/⟩（参照 2016-05-08）.

[13] 秘密結社シェルショッカー日本支部，恐怖！小鳥男 (オンライン)，入手先 ⟨https://github.com/ShellShoccar-jpn/kotoriotoko⟩（参照 2016-05-08）.

[14] sferik, T – A command-line power tool for Twitter. (オンライン)，入手先 ⟨http://sferik.github.io/t/⟩（参照 2016-05-08）.

[15] The Termtter Team，Termtter – Termtter is a terminal-based Twitter client. (オンライン)，入手先 ⟨http://termtter.github.io/⟩（参照 2016-05-08）.

[16] 東京地下鉄, ニュースリリース, 2014年3月27日第4報 (オンライン)，入手先 ⟨http://www.tokyometro.jp/news/2014/pdf/metroNews20140327_10nen.pdf⟩（参照 2016-05-08）.

[17] 東京地下鉄, 東京メトロの「オープンデータ活用」の取組み (オンライン)，入手先 ⟨http://www.soumu.go.jp/main_content/000372119.pdf⟩（参照 2016-05-08）.

[18] 松浦智之, 西原翼，メトロパイパー (オンライン)，入手先 ⟨http://metropiper.com/⟩（参照 2016-05-08）.

[19] 東京地下鉄, オープンデータ活用コンテスト結果発表 (オンライン)，入手先 ⟨http://awards.tokyometroapp.jp/⟩（参照 2016-05-08）.

[20] Microsoft Developer Network，Running Bash on Ubuntu on Windows! (オンライン)，入手先 ⟨https://channel9.msdn.com/Events/Build/2016/P488⟩（参照 2016-05-08）.