推薦
投稿論文

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]

[†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

ソフトウェアは，より高度な要求，あるいは時代ととも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，ほか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．そこで筆者らは，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（Portable Operating System Interface）に着目した．POSIX は現状で多くの UNIX 系 OS が準拠している上に，1988 年の初出以来，その仕様はほとんど維持されている．このような性質を持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，筆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．

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．故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）の利用なしに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）のバージョン要件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で改修の必要がなくても作り直しを迫られることがある．一方，ソフトウェア利用者も，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．

近年のソフトウェアは，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，実際多くの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の低さで悩んでいた筆者らが，解決方法を模索することから始まった．

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，さまざまな規格が提案されているという文献は発見できた[1][2][3].

それらの文献で示されていた1990年前後の各種の規格のうちの1つがPOSIXであった．この規格は，2010年代においても現存するUNIX系OSの大多数が規範としており，それらの環境におけるプログラミングを説明・学習するための基本になっている[4]．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考えた．

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筆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．本章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（IEEE Std 1003）規格で定められている内容を中心にプログラミングすることである．ここでの「中心」とは，規格に極力準拠するという意味である．

POSIX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）に限定される．その理由は，POSIXで用意されているプログラミング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に準拠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中心主義では，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をする．シェルスクリプトを選択することには，以下に述べる3つの利点がある．

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．

シェルスクリプトはインタ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．

たとえば，“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る．この場合，

```
■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

while文とtestコマンド（[）でループさせるのではなく，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させる（図1）．

この両者のシェルスクリプトをUNIX機［CentOS 7.2, Intel Xeon W5580（4コア3.2GHz），メインメモリ48GB, HDD 1.6TB］で実行したところ，前者の所要時間は2.67秒だった一方，後者は0.05秒であった（両者とも5回平均値）．

4.1.2項で述べるように，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と，やはり処理速度が改善する．

このことを実証する例として，郵便番号を与えると該当する住所を検索するシェルスクリプトによるプログラム[5]を公開している．日本国内の現在の郵便番号約14万件に対し，先程のファイル削除の検証におけるサーバ側の検索時間は0.05秒（5回平均値）であった．同種のプログラムはWeb通販サイトなどでさまざまな言語により実装されているが，本プログラムをCGIスクリプトとして動作させて比較しても遜色はなく，実用上も問題ない早さで応答することが確認できる．

そしてシェルスクリプトは元々，業務プログラムでも多用されるテキスト処理の記述が容易な言語の一つである．したがって，前述のように書き方に気を付ければ，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・メンテナンスコストの抑制

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，依存ソフトウェアをあらかじめ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時の作業やトラブルが抑えられる．

依存ソフトウェアのインストールが不要ということは，後にそれらがバージョンアップされ，その影響で正常に動作しなくなるというリスクもない．OSには依存しているものの，OSのバージョンアップの影響は受けない．そのOSがPOSIX準拠を謳う限り，POSIXの仕様だけは維持されるからである．もし利用中のOSのサポートそのものが終了するとしても，POSIX準拠を謳うOS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．このようにして，運用開始後のメンテナンスにおける作業やトラブルも抑えられる．

3.2　3つの小指針

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る．これは名称が示すとおり，POSIXで規定されている仕様にプログラムを準拠させるということである．プログラムは可能な限りこの指針を守ることとする．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が対象である．

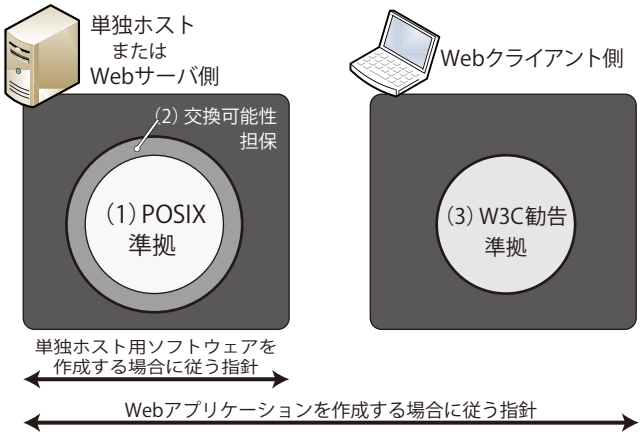

図2　POSIX中心主義を構成する3つの小指針と適用範囲

3.2.2　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処理を実現するための指針である．後述する「交換可能性」という性質を担保することを条件に，POSIXの範囲外のコマンドの使用を認める．具体的には，外部のWeb APIの操作やメールの送受信，各種バイナリデータ処理を要する場合などである．

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クライアント側アプリケーション開発のための指針である．2016年現在，この構成においてクライアント側に用いられる主な技術はWeb（HTML/CSS/JavaScript等の規格の集合）であり，POSIX準拠の指針が適用できない．しかし，Webにも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勧告があり，発生の経緯がPOSIXと類似している．そこで，クライアント側のプログラムはW3C勧告への準拠に努める．

3.3　POSIX中心主義の適する分野

POSIX中心主義は以上の特徴を持つことから，活用に適する分野と適さない分野がはっきりと分かれる．

説明してきた3つの小指針すべての根底にある考え方は互換性である．したがって，1つの開発団体が提唱し，まだ同等機能を有する別実装が存在しないソフトウェアを利用することはPOSIX中心主義に反する．このようなソフトウェアは，性能・機能の優位性を競って発表され，発表から日の浅いオープンソースソフトウェアや，営利企業が発表するプロプライエタリなソフトウェアに多い．

たとえば，NVIDIA社が提供するCUDA（Compute Unified Device Architecture）という開発環境がある．これを使えばGPU資源を利用して高速な計算が行える．しかし，CUDA上のC言語（CUDA C）で拡張された機能を利用したプログラムを実行させられる他実装は，今のところ存在しない．この例のように，時代の最先端の性能や機能が求められる分野に，POSIX中心主義プログラミングは適しない．

一方で，最先端の性能を求めず，むしろ互換性や持続性がより求められる分野も多数存在する．たとえば会計

処理等，主に古くから紙の書類で行われていた事務作業である．これらの分野はコンピュータと比べ，歴史も十分に長く，要求の変化も緩やかである．そしてC言語やシェルスクリプト等，多数の開発団体による互換実装が存在するソフトウェアで開発できる．POSIX中心主義は，このような特徴を持つ分野での開発に比較的適している．

4. POSIX 中心主義の詳細

本章では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（IEEE Std 1003）規格に極力準拠することである．特に問題がない限りはこの文書で記されている文法やコマンド，オプション，正規表現，保証される出力結果のみに依存したプログラムを書くものとする．この文書はWebページとして公開されており[6]，“opengroup posix”などのキーワードで検索可能である．したが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た中，OS間の最低限の互換性を確保するため1988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．

たとえばtrコマンドで文字範囲を指定する際，System V系の実装では“[0-9]”，BSD系の実装では“0-9”などと書かねばならず互換性がない．POSIXは後者を選択したものの，提唱するPOSIX中心主義ではその書き方を推奨しない．“0123456789”のように範囲指定の記号を使わず，いずれの環境でも正しく動く記法を推奨する．

また，POSIXで明文化し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，筆者らは，情報として整理しながら公開しており，新たな知見が得られる都度更新している[7]．

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，主にファイルの操作や内容の加工，およびバッチ処理などの役に立つ原始的なものしか用意されておらず，アプリケーション開発向けのコマンドは十分に揃っているとはいえない．これらPOSIXコマンドの呼び出し元となるシェルスクリプトに関しては，3.1.1項で示したように，他言語と比較して効率面で著しく不利な要因はないものの，多くのアプリケーションで多用される処理まで毎回それらの原始的なコマンドから作っていては効率が悪い．

そこで筆者らは，汎用性の高い処理を，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」[8]を提案している（有）ユニバーサル・シェル・プログラミング研究所（以下，USP研究所と記述する）がリリースしているusp Tukubaiというコマンド群[9]である．

たとえば，現代のアプリケーションでは，データ管理のためにデータベースを利用するものが多く，中でもリレーショナルデータベース（RDB）は多用されるものの一つである．しかしながら，各種のRDB実装は，外部とのインタフェースの差異が大きいことや，データの内部構造がバージョンごとに異なってデータ移行作業が必要になるなど，互換性や持続性を重視する現場においては問題も多い．データベース処理もPOSIXの範囲で実装できればこの問題を改善できるが，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上，usp Tukubaiには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の例である．前半のSQL文による記述例は，後半に示したように同等の処理をPOSIXコマンドでも記述できる．

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・ライト等）をTukubaiコマンドのインタフェースに倣って独自に作成もしている．

これらの結果，筆者らはPOSIXに準拠しながら，開発

```
■ SQL 文で記述
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;

■ POSIX コマンドで記述
cat blacklist.txt |
# 第 1 列 :BL 会員 ID #
sort -k 1,1       | ←会員 ID で並べ替え（5 行下も同様）
uniq              > sorted_bl.txt

cat members.txt             |
# 第 1 列 : 会員 ID 第 2 列 : 名前 #
sort -k 1,1                 |
join -1 1 -2 2 -v 2 sorted_bl.txt - ←
```

ブラックリストの会員IDで外部結合できない行のみを抽出

図 3　SQL と POSIX による RDB 的操作の例

表 1　POSIX 準拠しつつ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）
parsrx.sh	XML テキストをパースする（CSV 版，JSON 版もあり）
makrj.sh	JSON テキスト生成をする（CSV 版もあり）
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（共有ロック版もあり）
base64	Base64 エンコーダ・デコーダ（GNU Coreutils からの移植）
urlencode	URL エンコードする（デコーダもあり）
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	HTTP のセッション管理をする

に有用な多くのコマンドを用意できた（表1）．

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 [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れば実現できるが，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 が使えなくなったときでも，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 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等の問題で使えなくなったとしても他実装へ容易に移行できるからである．

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

たとえば，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 のほか，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 脆弱性の教訓から LibreSSL という別実装が登場した．

ただし，両コマンドと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，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

たとえば 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プログラミング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上で if 文等を使って分岐させ，各々のコマンド書式に従って同等の処理を記述する．

同名コマンドのときと同様に，使ってよいオプションに注意が必要である．たとえば，curl コマンドには Web フォームやファイル送信用の“-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ないからである．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．

4.2.2.3　HTTP サーバに対する考え方

逆に，Webアクセスを受け付けたい場合（サーバ側プログラム）には，やはりPOSIX規格にはない何らかのHTTPサーバソフトウェアを用意する必要がある．HTTPサーバにはApacheやnginx，lighttpd等の実装が存在するが，作成するアプリケーションはそれらの中から最低でも2つ以上の実装で動作するよう，1つの実装にしかない機能を利用せずにプログラムを書く．こうすることで，依存が必要なHTTPサーバソフトウェアについても交換可能性が担保できる．

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業界では各OSが機能を独自拡張し，互換性が低下した状況を改善すべくPOSIX規格が策定された．一方，Web業界では各Webブラウザが機能を独自拡張し，互換性が低下し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[11]，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

この指針の下ではjQuery等の各種JavaScriptライブラリを原則使わない．W3C勧告の範囲の仕様にのみ基づき，プログラムを書く．後述するXMLHttpRequestに関しても基本的な部分は40行程度で記述できる[12]．一度書けばほか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はさほど低下しない．

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．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は最小限に留める

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や廃止される可能性があるので極力利用すべきではない．

先程述べたXMLHttpRequestもその一つである．これは，Webクライアントが，ページ全体の再読み込みなしにWebサーバとのデータ送受信を実現する機能であり，Googleマップ等を支える要素技術として知られるAjaxも，この機能を活用したものである．しかしこのXMLHttpRequestは，2016年9月現在，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とどめるべきである．

4.3.2.1　XMLHttpRequest に関する事例

2016年10月6日，ワーキングドラフトの段階にあったXMLHttpRequestに関する文書は，勧告化への作業が中止され，元々提唱していた別団体（WHATWG）に戻されて議論されることになった[13][14]．WHATWG上では2017年5月現在も議論が続いており[14]，今後仕様が変更される可能性もある．

この事例からも，ドラフト段階にある仕様の利用は最小限にとどめるべきであることが分かる．

5.　POSIX 中心主義の効果検証

POSIX中心主義は，ソフトウェアの互換性や持続性を高めたいと思った筆者らが，2014年頃から徐々に実践し，理論化していったプログラミング指針である．実際にソフトウェアの互換性や持続性向上にどの程度の効果があるか結論（特に長期持続性）を得るには長年に渡る検証が必要ではあるが，実践から約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プログラムが作れることを実証するために「恐怖！小鳥男」（以下小鳥男と称す）[15]というプログラムを作成した（図4）．

5.1.1　小鳥男の概要

小鳥男はCUI（キャラクタユーザインタフェース）型のシェルスクリプト製Twitterクライアントプログラムである．POSIX準拠および交換可能性担保の指針に基づき作成した．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．

内部的にはどの操作も，

1）OAuth認証文字列の生成

図 4　シェルスクリプト製 Twitter クライアント「恐怖！小鳥男」

2）Twitter Web API にアクセス
3）レスポンスデータ（JSON）の解釈
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 コマンドだけで実装できる．したがって，小鳥男は 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，あとは小鳥男のファイル一式をコピーすれば利用できる．このため，既存の主要な CUI 型 Twitter クライアント（T[16] や termtter[17] 等）と比べ，インストールがきわめて簡単である．

5.1.2　小鳥男の動作実績

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．

これまでのところ，Cygwin 系以外の環境には，特定の環境専用の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．一方，Cygwin では 1 カ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 が Windows 上に構築されていて，ps コマンドの仕様が POSIX とは異なっていたという事情による．

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．

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
Ubuntu on Windows	Linux 系，Windows 10 Insider Preview（ビルド 14936）版

*1　Cygwin 向けの専用コードを追加した．

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）（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[18]．そのうちの 1 つが，自社の鉄道に関するデータを一般に公開し，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」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公開し，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[19]．

筆者らはその意図を汲みとり，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」

応募したソフトウェアは「メトロパイパー」である（図 5）．これは駅施設にある発車標（列車の行先や発車時刻などを表示する電光掲示板）をイメージしたものである．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 勧告準拠の指針に従いつつ，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．すなわち，POSIX 中心主義プログラミングでもモバイルフレンドリなソフトウェアが作れる．本作品はソースコードと共に今も公開している[12]．

5.2.3　コンテストの結果とその後の経緯

入賞作品は 2015 年 2 月に発表された[20]．ユーザインタフェースの優れた作品が多く選ばれ，iPhone や Android 等の携帯端末向けアプリケーションが上位を占

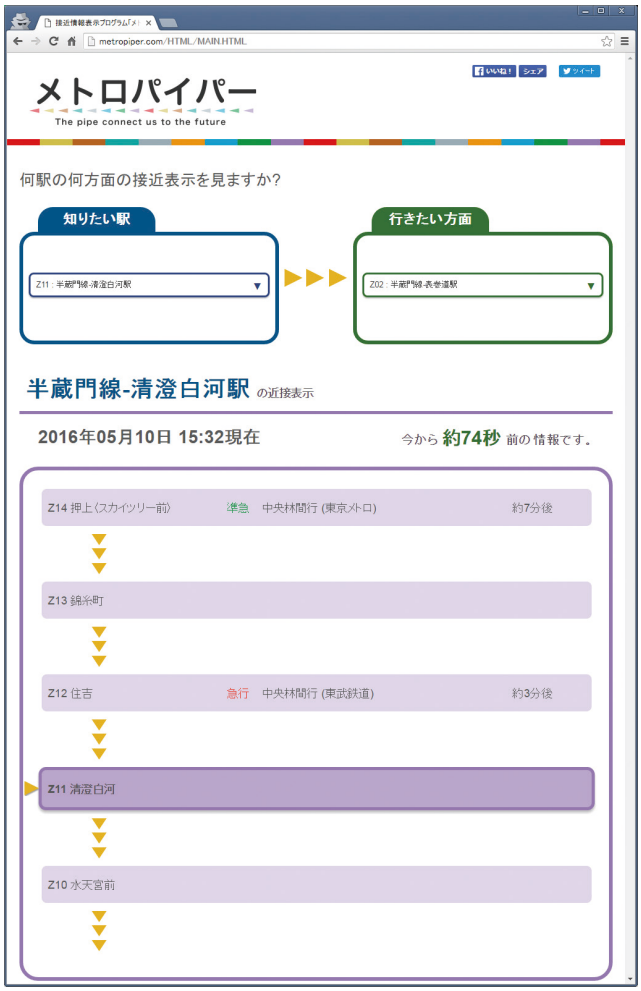

図 5　東京五輪までの動作を目指したソフト「メトロパイパー」

めた.

しかし，コンテストの応募締切から約2年が経過した現在，応募作品の動作状況を調査してみると，図6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すで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は163で，すで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，応募作品の動作画面（スクリーンショット）や説明文に記されているとおりの動作・使い方ができず，正常に動作していないと思われる作品が11あった（ただし明らかな説明文の間違いにより動作が異なるようなものは持続性のなさに起因するものではないため，ここでは正常と数えた）. よって，すでに正しく利用できなくなっている作品数は174である. この中には一部のコンテスト入賞作も含まれていた.

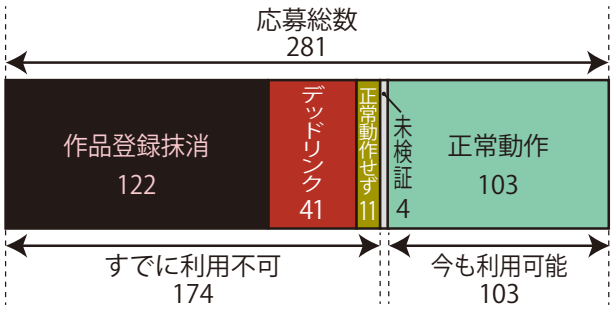

図 6　コンテストから約 2 年後（2016-09-30）の応募作品動作状況

このように，利用できる作品は2年で4割ほどにまで減った．2020年の東京オリンピック開催の時点ではさらに減少するものと予想される．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．一方，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9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.

なお，作品登録やデッドリンクになった作品は，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あると考えられるが，ここでは集計に加えた．理由は，ソフトウェアの持続性は依存ソフトウェアの持続性以外の要因も受けるからである. たとえば，全応募作品中33作品はiOS専用であり，このうち13作品はデッドリンクになっていた. iOSアプリケーションは，公開（App Storeに登録）するために年間99ドルのアカウント登録料が必要である．一方，無料公開でなければならないというコンテスト応募条件があり，維持費が負担できず利用できなくなった可能性も考えられる.

依存ソフトウェアの持続性が十分でも，維持費（ほかにもサーバの維持費，ドメイン管理費などさまざまなものがある）が高額で公開ができなくなってしまっては結果として持続性は低く，持続性の高さに関してはさまざまな要因を考えなければならない．POSIX中心主義は，交換可能性の担保，すなわち選択肢に多様性を持たせることを重要としているため，持続性を阻害するさまざまな要因に対する耐性が高い.

5.3　Windows10 に搭載された POSIX 環境

2016年8月，Microsoft は同社OSにWindows Subsystem for Linux（WSL）と呼ばれるUbuntu Linux互換システムを搭載した[21]．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9月現在，ベータ版ではあるが，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すでに実用的が高い．デスクトップPCのOSシェアはWindowsが約9割を占めている[22]．この9割の部分が徐々にWSLに対応したWindows10以降に置き換わっていくと予想されるため，POSIX中心主義は今後大多数のデスクトップPCで通用する指針になると思われる．

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（IEEE Std 1003）規格に極力準拠してプログラミングをする，POSIX中心主義と称すプログラミング指針を提案した．POSIX規格に準拠をすると，開発言語は実質的にシェルスクリプトになるが，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中心主義の効果の検証活動を報告した．互換性，長期持続性の両面については，現時点で期待した結果が得られており，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．本指針発案のヒント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ともに，本指針を支持し，御指導くださる金沢大学の共同研究者の皆様，特に本稿をご推敲くださった北口善明助教（現東京工業大学准教授），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），(37 (1987-OS-035))，pp.1-6 (1987).
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), pp.84-92, (1989).
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)，pp.324-335 (1993).
4）Stevens, W. R., 他：詳解UNIXプログラミング第3版，翔泳社 (2014).
5）秘密結社シェルショッカー日本支部，「郵便番号→住所」補完をshでやるデモ，https://github.com/ShellShoccar-jpn/zip2addr（2016年9月30日現在）
6）What is POSIX?, The Open Group, https://collaboration.opengroup.org/external/pasc.org/plato/（2016年9月30日現在）
7）松浦智之：Windows/Mac/UNIXすべてで20年動くプログラムはどう書くべきか，C&R研究所 (2016).
8）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 (2015).
9）USP engineers' community, usp Tukubaiについて，https://uec.usp-lab.com/TUKUBAI/CGI/TUKUBAI.CGI?POMPA=ABOUT (2016年9月30日現在)
10）秘密結社シェルショッカー日本支部，installer, https://github.com/ShellShoccar-jpn/installer（2016年9月30日現在）
11）World Wide Web Consortium, All Standards and Drafts, https://www.w3.org/TR/（2016年9月30日現在）
12）松浦智之，西原　翼，メトロパイパー，http://metropiper.com/（2016年9月30日現在）
13）XMLHttpRequest Level 1 W3C Working Group Note06 October 2016, https://www.w3.org/TR/2016/NOTE-XMLHttpRequest-20161006/（2017年5月22日現在）
14）XMLHttpRequest Living Standard, https://xhr.spec.whatwg.org/（2017年5月22日現在）
15）秘密結社シェルショッカー日本支部，恐怖！小鳥男，https://github.com/ShellShoccar-jpn/kotoriotoko（2016年9月30日現在）
16）sferik, T-A command-line power tool for Twitter., http://sferik.github.io/t/（2016年9月30日現在）
17）The Termtter Team, Termtter-Termtter is a terminal-based Twitter client., http://termtter.github.io/（2016年9月30日現在）
18）東京地下鉄，ニュースリリース，2014年3月27日第4報，http://www.tokyometro.jp/news/2014/pdf/metroNews20140327_10nen.pdf（2016年9月30日現在）
19）東京地下鉄，東京メトロの「オープンデータ活用」の取組み，http://www.soumu.go.jp/main_content/000372119.pdf（2016年9月30日現在）
20）東京地下鉄，オープンデータ活用コンテスト結果発表，http://awards.tokyometroapp.jp/（2016年9月30日現在）
21）Microsoft, Windows Subsystem for Linux, https://blogs.msdn.microsoft.com/wsl/（2016年9月30日現在）
22）マイナビ，8月OSシェア, http://news.mynavi.jp/news/2016/09/02/092/（2016年9月30日現在）

松浦 智之（正会員）richmikan@richlab.org
プログラマ，テクニカルエンジニア（ネットワーク）等の本業の傍ら，主にその分野の同人誌を作り，コミックマーケット等に出展中．2016年より大学コンソーシアム石川にて「シェルスクリプト言語論」を開講し，講師も担当．

大野 浩之（正会員）hohno@ohnolab.org
金沢大学総合メディア基盤センター教授．博士（理学）．東京工業大学大学院講師，郵政省通信総合研究所（現NICT），内閣官房併任を経て2006年より現職．非常時情報通信，情報セキュリティ，ものづくり支援等の研究に従事．

當仲 寛哲（正会員）koho@usp-lab.com
UNIX哲学に基づくグルーテクノロジー「ユニケージ開発手法」考案者．東京大学卒業後，（株）ダイエーで基幹システムを刷新し業務改革．2005年ユニケージ開発手法の研究・教育・普及を行うUSP研究所を設立．代表取締役所長．

投稿受付：2017年1月23日
採録決定：2017年5月8日
編集担当：坂下　秀（(株）アクタスソフトウェア）