OLIVE 香川大学学術情報リポジトリ

香川大学経済論叢
第67巻 第3・4号 1995年2月 183-215

システム管理から見たプログラムとスクリプトとの機能比較

今井慈郎*
三谷宗子
本田道夫

1. はじめに

UNIXのシステム管理ではユーザからのニーズ（ユーザサービス）や外的な要請（加盟組織からの各種連絡・指示）に素早く対応する必要がある。要求される処理は多くの場合機械的であるが，コマンドレベルの操作（手作業）か，C言語などのプログラミング言語を用いたプログラムによる実行かのどちらを用いて実現するかという選択を常に迫られることになる。単発的な要求であれば手作業の方が効率の点でも有効な場合があるが，繰り返し操作を必要とする場合には処理速度や信頼性の点でプログラム，あるいはそれと同等なスクリプトによる実行が有効になる。この見極め，すなわち時間コストと処理の信頼性のトレードオフ，には試行錯誤が生じる可能性がある。どちらを採用するかは，システム管理のみならず計算機のオペレーション全体に関連した重要な課題の1つと言える。

実際のシステム管理では，要求される仕事を単純なオペレーションのみで処理できるようなコマンドが存在するかどうか，まず最初にマニュアルなどを検索することから始まる。もし最適なコマンドが存在すればそれだけ短期間で処理は終了する。しかし，一般的にはいくつかのオペレーションを組合せて必要

＊E-mail : {imai, mitani, honda}@ec.kagawa-u.ac.jp

OLIVE 香川大学学術情報リポジトリ

な作業を行うことになる。通常，C言語などでプログラムを書き，システム管理に必要な処理を実行させるという選択肢を採用することはあまり多くない。その理由はいくつか考えられるが，まず第一に，対応速度が常に要求される点があげられる。仕事を仕上げるのに要する時間は短いほどよく，数週間を要してもかまわないなどというケースは皆無と言ってもよい。ゆっくりとプログラムなど考えていては要求される応答スピードに答えられない。第二に，信頼性の問題が指摘される。システム管理を行う場合，特権モードを持つスーパーユーザ(UNIX では“root”と呼ぶ)になって処理を行うことが頻繁に生じる。もちろん，最初から root でログインする場合もあるが，通常はキーになる処理を行う直前で root になる(UNIX ではコマンド“/sbin/su”を利用)のが一般的であろう。そして，その場合は処理の実行効率よりも，高い信頼性が重要視される。もちろん，信頼性だけを向上させるためにわざわざ試行錯誤を何回か行い，その結果に応じてデバッグを精密に繰り返すなどという時間的余裕など，システム管理の局面では極めて希である。

プログラム作成ではコンパイル，そして多くの場合，頻繁なデバッグが不可欠となる。そこでこれに代わって大きな役割を果たすのがスクリプトの利活用である。スクリプトにはいくつかの種類があるが，シェルスクリプト(C-shell script)や grep/sed/awk/perl などのフィルタ系のスクリプトの使用頻度が高く処理効率も良いと思われている。UNIX 文化の常識として，『シェルスクリプトは Bourne-shell 用のスクリプトが広い利用範囲を有する』という意見をよく耳にする。しかし，アメリカ合衆国はもとより日本においても大学関係者や研究者の間ではむしろ csh(あるいはその拡張版としての tcsh)の利用が一般的であり，香川大学の UNIX 環境においても同様であると言える。従って，ここでも csh 用シェルスクリプトによる例示の方が一般的であるとの認識に立っている。

本稿では，主にシステム管理の観点に立脚して具体的な UNIX マシンのシステムオペレーションを取り上げ，そのオペレーションをコマンドレベルで実現するか，それともプログラム記述あるいはスクリプト作成によって実現するか，

OLIVE 香川大学学術情報リポジトリ

といった方法論を議論し，これらを決定する際の指針を明確にするよう試みている。そして，方法論を具体的に議論する課程において，システム管理という局面では，コンパイラ言語であるC言語プログラムと，多くのインタープリタ型フィルタ系のスクリプトとのどちらが適しているかを対比させながら，それらの特徴や機能を定量的に比較検討している。また，実用上生じる具体的な問題点を指摘し，各々の方法論のもつ利害得失などについても言及している。

2. UNIX のシステム管理と操作

SVR4 系 UNIX では，システム管理を行う上で頻繁に要求される機能を統合し，会話的に処理できるコマンドがある。ここではシステム管理オペレーションをコマンドレベルで実現する代表例として sysadm を取り上げ，そのコマンドの特徴や問題点を示す。ついで，sysadm 利用の改善策としてのプログラムおよびスクリプトを用いたオペレーションの実行例を示す。

2-1　システム管理のための会話型多機能コマンド sysadm

ユーザ登録は UNIX のシステム管理の中でもニーズの高い処理の１つである。どのようなユーザ名で登録するか，環境設定の規定値を決めるホームディレクトリの割り当て，あるいはドットファイルの内容をどのように決めるか，など多くの選択肢がある。しかし，ユーザ登録の操作自体はかなり機械的であり，最も基本的なシステム管理操作となっている。香川大学情報処理センターに導入された UNIX マシンの OS は SVR4 系(正確には，SVR4.2)であり，ユーザ登録を含めたシステム管理を総合的に支援するソフトウェアとしてコマンド "/usr/sbin/sysadm" が提供されている。図 2-1 では sysadm を起動した後，ユーザの追加登録を行うために登録の追加を指定した画面を示している。このコマンドは，

1)　システム管理を会話的に実行可能
2)　UNIX システム管理の初心者でも安心して操作可能

という特徴を有している。

OLIVE 香川大学学術情報リポジトリ

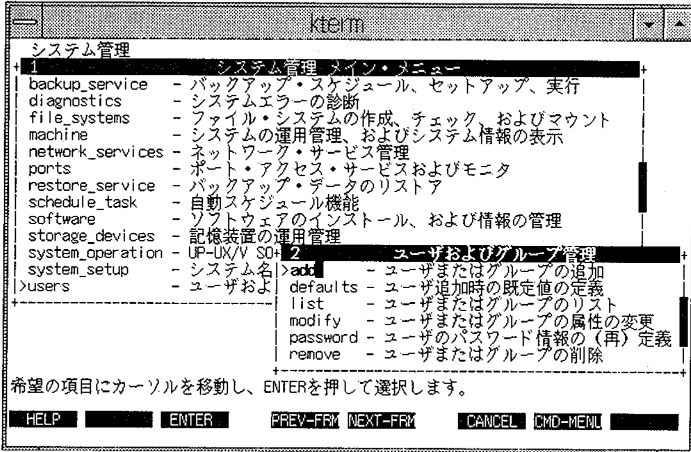

図 2-1　sysadm の起動後にユーザ追加登録を選択した画面

従来の BSD 系 UNIX では sysadm も存在せずユーザ登録などは煩雑な手順を必要とした。すなわち，

1）　コマンド “vipw” でファイル “/etc/passwd” を編集

2）　新規登録ユーザのための各種情報を追加

3）　ユーザ用のディレクトリを作成

4）　オーナーやグループを変更

5）　システムを再起動

といった一連の作業が要求される。一方，SVR4 系 UNIX では sysadm を利用することにより，メニュー画面上の空欄を埋める作業で誰にでもユーザ登録が可能である。UNIX 特有のファイル/ディレクトリ間の詳細な関係などを理解していなくても，メニューに従って操作を行うだけでシステム管理に必要な作業が簡単に実行できる。このようにメニューに従って，キーオペレーションを行うだけで，ユーザの追加登録などはシステム管理が仮に不案内なユーザであっても容易にできることになる。追加登録を選択すると個々のユーザ毎に情報をタイプ入力するようメニュー画面が表示される。

OLIVE 香川大学学術情報リポジトリ

図2-2のようなメニュー画面が表示されれば，コメント（GCOSフィールド），ログイン名などの順番で必要事項をユーザ1人1人を登録する毎にタイプ入力すればよいことになる。当然ながらミスタイプを発見すればカーソルの移動で訂正でき，確認作業に要する時間さえ厭わなければ初歩的タイプミスのほとんどが発見できるはずである。sysadmを利用すれば，システム管理の素人がオペレーションすることも可能であると言える由縁である。

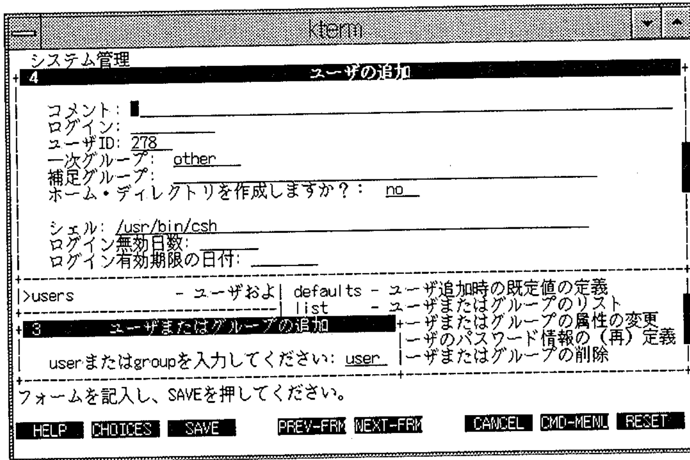

図 2-2　sysadmのユーザ登録メニュー画面

このように利点の多いコマンドsysadmであるが，大きな問題点も存在する。例えば，数人のユーザ登録ならば会話型処理でもあまり苦痛を感じないが，ユーザ登録すべき人数が数十人となると会話型処理が単に苦痛であるばかりでなく，別の問題点が顕著に現れてくる。それを整理すると，

(1)　コマンド実行中にメニュー画面の記載を手作業で行うための処理効率の低下

(2)　人手による作業のため，ヒューマンエラー混入の危険性増大

などが考えられる。(1)は緊急度が大きければ大きいほど重大な問題となる。システム管理では緊急度が低い要求は基本的に少ないため処理効率の問題は避け

OLIVE 香川大学学術情報リポジトリ

ることはできない。特に，学部教官の同時ユーザ登録とか学生のユーザ登録などをsysadmで行う場合，処理効率の低下は深刻な問題となる。一方，(2)は登録すべきユーザ名や各種情報をファイルなどを機械可読形式（テキストファイルやワープロファイル）で受理しても，一度プリントアウトして印刷結果を見ながら再度キータイプする必要が生じる。これでは読み間違いと共にキータイプ間違いも生じ易く信頼性に大きな問題が発生する。もし，信頼性を向上させようと思えば作業終了後の再点検など無駄なオーバーヘッドを要することにもなる。これ以外にも問題がある。例えば，sysadmでは画面制御のESCシーケンスが複雑に混入するため，コマンド“/usr/bin/script”などを用いた作業手順のドキュメント化にも適していない。

そこで，このような問題点を回避するため，我々は敢えてsysadmを使用しないで，次のような手順でユーザ登録を行っている。すなわち，プログラムあるいは各種フィルタ(これにはシェルスクリプトも含まれる)を可能の限り用いることにより，

(1) 新規登録ユーザ名などを含む情報ファイルの機械的解析，

(2) ユーザ登録を行うコマンド系列の自動生成

(3) 生成されたコマンド系列をシェルスクリプトとして登録処理のバッチ的実行

といった流れでシステム管理の1つであるユーザ登録操作を実現している。この方法論を採用することにより，処理効率の点においても信頼性の点においても格段の改善が図られている。ではプログラム記述による実現とスクリプト作成による実現とどちらがより適していると言えるだろうか。

2-2 プログラムおよびスクリプトによるコマンド系列の生成

まず，プログラムを利用したコマンド系列の生成例を示し，続いてフィルタ系のスクリプトによるコマンド系列の生成例を示す。そして両者を具体的に比較する。同一情報ファイルを用いて，プログラムあるいはスクリプトを利用してユーザ登録のためのコマンド系列を自動生成する利点はかなり多い。例えば，

OLIVE 香川大学学術情報リポジトリ

繰り返し作業やヒューマンエラーなどにより効率低下や危険性の問題を回避できるだけでなく，ファイル形式でユーザ登録に必要となる情報を用意しておくと，ユーザ登録のみならず，個々のユーザのパスワードロック解除や，ユーザ情報の修正やユーザ登録削除などにもそのまま利用でき，システム管理への用途が大きく広がるという利点が生じる。次に示すプログラムおよびスクリプトは基本的に同じ作業を機械的に実行させる目的で紹介している。従って，目的を特化してプログラムあるいはスクリプトを作成することにすれば，機能的にはどちらかが他方に対して優れているとは一概に判断しにくい場合も多い。以下では，C言語プログラムそしてawkスクリプトを例示し，具体例を用いた概観上の比較を行う。

次に示す図2-3のプログラムおよび図2-4のスクリプトは，どちらも登録すべき複数のユーザに関するデータファイル(例えば“name.tbl”)を読み込んで，ログイン名，グループ名，GCOSフィールド，ホームディレクトリ，およびログインシェルなどの各項目を選択的に読み取り，個々のユーザ毎にユーザ登録のための詳細情報を必要なフラグに設定したコマンド“/usr/sbin/useradd”の系列を自動生成するための処理を実現している。このコマンド系列はcsh用シェルスクリプトとなっており，出力結果を格納したファイルに実行可能属性を与えて実行させることにより一括して登録作業を行うことができる。このため，ヒューマンエラーが混入する危険性は大幅に回避できることになる。プログラムあるいはスクリプトと共に，自動生成させたコマンド系列をファイルに格納しておけばユーザ登録をどのように行ったかが容易にドキュメント化できる。このような利用方法も，システム管理を行う上でプログラムやスクリプトを利用する大きな利点と言える。比較のため，C言語のプログラムは標準入力および入力ファイル指定のどちらでも可能となっている。

図2-3に示すC言語プログラムはプログラムによってユーザ登録の機械的処理を実現する実例として掲示する。

OLIVE 香川大学学術情報リポジトリ


```
#include <stdio.h>
/*
 第 1 項目(argTab[0]) ログイン名
 第 2 項目(argTab[1]) 学科名(既定値: 経済学部)
 第 3 項目(argTab[2]) コメント:氏名(GCOS フィールド)
 第 4 項目(argTab[3]) コメント:所属(GCOS フィールド)
 第 5 項目(argTab[4]) コメント:大学名(GCOS フィールド)
 第 6 項目(argTab[5]) グループ(teacher | student)
 第 7 項目(argTab[6]) スケルトンディレクトリの指定(省略時: /etc/skel)
 第 8 項目(argTab[7]) ログインシェルの指定(省略時: /usr/bin/csh)
*/
#define MAXBUFSIZE 128
#define MAXTABSIZE 16

char buf[MAXBUFSIZE], *argTab[MAXTABSIZE];

main(int argc, char *argv[])
{
   char linebuf[MAXBUFSIZE], *fgets();
   int   lineNo, argNo, analyzeArg();
   FILE *fp;
   void delWhite(), generateComSeq();

   if ( argc >1 )
      fp = fopen(*++argv, "r");
   else
      fp = stdin;

   printf("#!/usr/bin/csh¥n");

   lineNo = 0;
   while(fgets(linebuf, MAXBUFSIZE, fp) != NULL){
      delWhite(linebuf);
      argNo = analyzeArg();
      generateComSeq(argNo);
      ++lineNo;

   }

   fclose(fp);
   printf("echo User Registrations: %d¥n", lineNo);
}


void delWhite(char linebuf[])
{
   int cp, i;
```

OLIVE 香川大学学術情報リポジトリ


```
    cp = 0;
    i = 0;
    while( linebuf[cp] != '¥n' ){
      if ( linebuf[cp] == ' ' || linebuf[cp] == '¥t' ){
        buf[i++] = ' ';
        while( linebuf[cp] == ' ' || linebuf[cp] == '¥t' )
        ++cp;
      }
      else
        buf[i++] = linebuf[cp++];
    }
    buf[i] == '¥0';
  }


  int analyzeArg()
  {

   int cp, num;

   cp = 0;
   num = 0;
   argTab[num] = &(buf[cp]);

   while(buf[cp]){
     if ( buf[cp] != ' ' )
       ++cp;
     else{
       buf[cp] ='¥0';
       argTab[++num] = &(buf[++cp]);
     }
   }
   return ++num;
}


void generateComSeq(int argNo)
{
if (argNo == 8){
     printf("/usr/sbin/useradd -m -d /home/PRO/%s ", argTab[0]);
     printf("-c ¥"%s %s %s¥" -e ¥"¥" ", argTab[2], argTab[3], argTab[4]);
     printf("-g %s -k %s ", argTab[5], argTab[6]);
     printf(" -s %s %s¥n", argTab[7], argTab[0]);
  }
  if (argNo == 6){
     if ( !strcmp(argTab[5], "teacher") ){
```

OLIVE 香川大学学術情報リポジトリ


```
        printf("/usr/sbin/useradd -m -d /home/PRO/%s ", argTab[0]);
        printf("-c ¥"%s %s %s¥" -g %s", argTab[2], argTab[3], argTab[4], argTab[5]);
      }else{ /* student */
        printf("/usr/sbin/useradd -m -d /home/STUD/%s ", argTab[0]);
        printf("-c ¥"%s %s %s¥" -g %s", argTab[2], argTab[3], argTab[4], argTab[5]);
      }
      printf(" -e ¥"¥" -k /etc/skel -s /usr/bin/csh %s¥n", argTab[0]);
    }
  }
```

図 2-3 ユーザ登録用コマンド系列の自動生成を行うC言語プログラム

一方，図2-4に示すawkスクリプトはスクリプトによるユーザ登録の機械的処理を実現する実例として掲示する。

プログラムあるいはスクリプトを利用したコマンド系列の自動生成という方法論を採用することで，

- コマンドレベルの処理で問題となっていた，手作業による処理効率の劣化やヒューマンエラー混入の危険性などをかなりの確率で回避可能となる
- 副産物として，システム管理のドキュメント化（これ自身もドキュメントとして利用できるが，コマンド“/usr/bin/script”で処理全体をロギングできる）にも大きく寄与する

といった効果が期待できる。しかし，図2-3および2-4で示されるプログラムとスクリプトとの特徴の違いは大きく，アルゴリズム自体を意図的に似せているものの，その記述スタイルやサイズなど，第一印象からのみの判断であればユーザにとって全く別次元の存在といった印象を与える。

次節では，システム管理という観点に主眼を置きつつ，上記のようにかなり異なった印象を与えるプログラムおよびスクリプトの特徴を可能な限り定量的に比較対比して，どちらを利用するのがシステム管理としてより適切か，またシステム管理という大きな範疇のなかで用途によってどのように使い分けるべきか，その場合の両者の利害得失はどのようになっているかなどについて整理する。

OLIVE 香川大学学術情報リポジトリ


```
#!/usr/bin/awk -f
# 第1項目($1) ログイン名
# 第2項目($2) 学科名(既定値: 経済学部)
# 第3項目($3) コメント:氏名(GCOSフィールド)
# 第4項目($4) コメント:所属(GCOSフィールド)
# 第5項目($5) コメント:大学名(GCOSフィールド)
# 第6項目($6) グループ(teacher, student)
# 第7項目($7) スケルトンディレクトリの指定(省略時: /etc/skel)
# 第8項目($8) ログインシェルの指定(省略時: /usr/bin/csh)
BEGIN {
 print "#!/usr/bin/csh";
 print "";
}

NF == 8 {
    printf("/usr/sbin/useradd -m -d /home/PRO/%s ", $1);
    printf("-c ¥"%s %s %s¥" -e ¥"¥" -g %s -k %s ", $3, $4, $5, $6, $7);
    printf(" -s %s %s¥n", $8, $1);
  }
NF == 6 {
    if ($6 ~ /teacher/){
        printf("/usr/sbin/useradd -m -d /home/PRO/%s ", $1);
        printf("-c ¥"%s %s %s¥" -e ¥"¥" -g %s -k /etc/skel ", $3, $4, $5, $6);
    }else{ #Student
        printf("/usr/sbin/useradd -m -d /home/STUD/%s ", $1);
        printf("-c ¥"%s %s %s¥"   -g %s -k /etc/skel ", $3, $4, $5, $6);
    }
    printf(" -s /usr/bin/csh %s¥n", $1);
}

END {
    print "echo User Registrations: " NR;
}
```

図 2-4　ユーザ登録用コマンド系列の自動生成を行う awk スクリプト

OLIVE 香川大学学術情報リポジトリ

3. プログラムとスクリプトの機能比較

同一機能を実現するプログラムとスクリプトとを例示し，両者の静的な比較および実行速度の比較を行う。システム管理という側面のみならず，ファイルのアクセスは頻繁に要求される処理である。プログラムとしてはUNIXでは既定値とも言えるC言語を想定しているが，スクリプトとしてはシェルスクリプトやフィルタ系のgrep/sed/awk/perlなどのスクリプトまで様々な対象が考えられる。ここでは，フィルタ系のawkとシェルスクリプトとしてcshとの組み合わせを考え，C言語プログラムとawk/cshスクリプトとの定量比較を試みる。ここで，何故cshおよびawkをスクリプトの代表としたかについては，筆者達自身の使用経験とそれから判断される有用性に基づいて決定しているが，一般的にもcshおよびawkの使用頻度はかなり高いと言える。

3-1 同一機能を実現するプログラムとスクリプトの例示

プログラムとスクリプトとを比較対照する上で基準とする項目はある同一機能を実現する上での，表現能力および処理能力と定める。表現能力とはある基準となる機能をどのように表現するかを示し，定量的評価するにはサイズと作成所要時間の計測が中心である。しかし，作成所要時間は個人差を考慮すると一意には決定しにくい。サイズは客観的であり，当然ながら作成所要時間とも深い関係にあると言える。そこで，表現能力として本稿ではサイズに焦点をあてる。一方，同一基準での処理能力比較は実行速度の比較と置き換えることが可能であり，実行速度の計測結果で処理能力を判断することとする。

同一基準となる機能として，システム管理上頻繁に要求されるディスクの使用状況の確認作業を行う機能を想定する。すなわち，現在どの程度ディスクが利用されているか，特別に大きなサイズのファイルは存在しないか，一定間隔でのチェックによりディスク使用率がどのように推移しているか，などを調べるため指定されたディレクトリ配下のファイルを検索してそのファイル名とサイズを報告し，合計のファイルサイズも表示する機能をプログラムあるいはス

OLIVE 香川大学学術情報リポジトリ


```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>

#define DIRMASK 0x41c0
int total;

main( argc, argv )
int argc;
char *argv[];
{
DIR *fp;
struct dirent *p;
struct stat sbuf;
char fileName[128];

    if ( argc == 1 ){
        fprintf(stderr, "Usage: %s dirName¥n", argv[0]);
        exit(0);
    }
    total = 0;
    fp = opendir( argv[1] );
    while ( (p=readdir(fp)) != NULL ){
        if ( !strcmp(p->d_name, ".") || !strcmp(p->d_name, "..")  )
            continue;
        sprintf(fileName, "%s/%s", argv[1], p->d_name);
        if ( !stat(fileName, &sbuf) ){
         if ( (sbuf.st_mode & DIRMASK) == DIRMASK ){
                subdir(fileName);
            }else {
                printf("%15d %s¥n", sbuf.st_size, p->d_name);
              total += sbuf.st_size;
            }
        }
     }
     closedir(fp);
     printf("Total: %8d Bytes¥n", total);
 }

 subdir(dirName)
 char *dirName;
 {
 DIR *fp;
 struct dirent *p;
 struct stat sbuf;
 char fileName[128];
```

OLIVE 香川大学学術情報リポジトリ


```
        fp = opendir( dirName );
        while ( (p=readdir(fp)) != NULL ){
            if ( !strcmp(p->d_name, ".") || !strcmp(p->d_name, "..")   )
                continue;
            sprintf(fileName, "%s/%s", dirName, p->d_name);
            if ( !stat(fileName, &sbuf) ){
                if ( (sbuf.st_mode & DIRMASK) == DIRMASK ){
                    subdir(fileName);
                }else {
                    printf("%15d %s¥n", sbuf.st_size, p->d_name);
                  total += sbuf.st_size;
                }
            }
        }
        closedir(fp);
}
```

図 3-1 C言語プログラム（ファイル名は“filesize.c”）での実現例

クリプトを用いて実現し，両者の表現能力および処理能力を比較対照することにする。

C言語プログラムにおいて，ファイルアクセスを効率よく実現するにはUNIX が提供するシステムコールを活用するのが便利である。[(1),(2),(3)] 従って，適切なインクルードファイルを読み込む必要が生じる。処理の都合上，再帰的な記述を行うが，C言語プログラムとしては恐らく典型的な処理記述の1つであり，プログラムを作成する上で特段の技法などを要求されるものではない。参考文献などが多く出版されており，最も一般的に利用されているプログラミング言語であるので個人差もあるが，記述の所要時間はプログラムサイズに比例する程度と言える。図 3-1 にC言語プログラムを示す。

スクリプトの場合，C言語プログラムのシステムコールに相当するのが既存のコマンドである。コマンドの実行結果を上手に利用することで，スクリプト全体のサイズを低減し，作成の所要時間など開発に要するコストを低く押さえることができる。図 3-2 (a)(b)にそれぞれ awk および csh のスクリプトを示す。csh や awk も参考文献が多く用意されており，[(4),(5),(8)] スクリプトを作成する上で有効

OLIVE 香川大学学術情報リポジトリ

である。図 3-2 (b)のシェルスクリプトを実行すると既存のコマンド“ls”をロングフォーマット表示のフラグつきで再帰的に実行させ，表示結果の文字列から図 3-2 (a)で示す awk スクリプトを用いて必要な情報を選択的に抽出する。

```
#!/usr/bin/awk -f
BEGIN { total = 0; }
$1 ~ /^-/ { printf("%15d %s¥n", $5, $9); total += $5; }
END     { printf("Total: %8d Bytes¥n", total); }
```

図 3-2 (a)　awk スクリプト（ファイル名は“filesize.awk”）での実現例—その 1 —

```
#!/usr/bin/csh

if ($#argv == 0) then
    echo "Usage: " $0 "dirNmae"
    exit 1
endif

  ls -alFR $1 | filesize.awk
```

図 3-2 (b)　csh 用シェルスクリプト（ファイル名は“fils.csh”）での実現例—その 2 —

図 3-1 と図 3-2 (a)および 3-2 (b)の組合せとを比較すると，共に実行結果の表示方法，使用方法 (usage) の表示，UNIX が提供する既存の機能（システムコールおよびコマンド）の利用，処理の再帰的な実行など多くの共通項があるものの，特に両者のサイズが際立って異なる点が印象的である。そこで，両者の違いを可能な限り定量的に比較し，第一印象のみに左右されないプログラム記述およびスクリプト作成の利害得失を論じたい。

3-2　実行所要時間に基づく実際の比較

定量比較を行う上でまず比較対象となるのは，そのプログラムサイズおよびスクリプトサイズ (csh と awk の合計) である。そこで，UNIX のコマンド“/usr/bin/wc”を用いて，プログラムとスクリプトの行数，ワード (単語) 数および文字数（これにはリターンコードなども含まれる）の計数を行う。これによって具体的なサイズ比較が容易になる。次に，プログラムの起動する場合に不可

OLIVE 香川大学学術情報リポジトリ

欠なコンパイルを行う。サイズのような静的データではないので，コンパイル処理時間は実行するマシンの仕様に大きく依存する。そこで，いくつかの利用可能なマシン上でコンパイル時間を測定する。一方，スクリプトではコンパイルを必要としないので，処理を実行する前に余分な時間を要しない（実際は，スクリプトファイルに実行属性を付与しなければならないが，その所要時間は無視できる)。むしろ，問題となるのはスクリプトを実行させるソフトウェアが移植されているかどうかであるが，今回対象としている csh および awk は通常，UNIX に付属しているソフトウェアであり，使用上の問題は生じない。後述する perl や awk の GNU 版である gawk などはマシンによれば移植されていない可能性もあるが，著名なソフトウェアであり，ほとんど問題なく移植される傾向にあると言える。

実行速度を定量的に計測するため，本稿では UNIX のコマンド “time” を利用している。通常 time は csh の組込みコマンドで提供されるが，それ以外にも “usr/bin/time” (SUN ワークステーションでは “/usr/5 bin/time”) などの同種のコマンドが利用できる。10 回程度繰り返して平均をとるなどの操作をしなければ，あまり高精度の測定は期待できない。しかし，ユーザが体感できる速度差のレベルは一般的には 10 分の 1 秒程度までであろうと思われる。しかもファイル検索に要する時間は対象となるディレクトリ配下のファイル数によって決まり，ある程度のディスク容量を対象とすることになるので必然的に所要時間は秒の単位以上となる。従って，本稿で論じる時間精度自身が高々10 分の 1 秒程度の精度での比較となっている。

表 3-1 にプログラムとスクリプトとの比較表を示す。スクリプトは csh シェルスクリプトと awk スクリプトとの組合せになっているが，2 つを合計しても C 言語プログラムの 20%程度のサイズである。プログラムの大きな特徴はコンパイル時間が必要となる点である。スクリプトにはこの項目が存在しない。実際の機能比較では合計 4MB 程度のファイル群からなるディレクトリの検索に要する時間を測定している（後述する例ではもっと容量の大きなディレクトリの検索を行っている)。測定値は処理が実行される 2 つのモード，ユーザモード

OLIVE 香川大学学術情報リポジトリ

とシステムモードの所要時間の和として表現される。さすがに，実行速度はプログラム（機械語コード）の方がスクリプトに比べて4倍程度高速である。しかし，プログラムおよびスクリプトの開発に要するコストまで考慮すると，実行速度の差をどこまで有意の差と見るべきかという疑問も生じる。表現能力と処理能力とを総合的に判断するとスクリプトに分があると言うべきかもしれない。

表 3-1　UNIX サーバ gauss 上でのプログラムとスクリプトとの定量比較

	プログラム	スクリプト	
	C 言語プログラムファイル (filesize.c)	csh スクリプトファイル (fils.csh)	awk スクリプトファイル (filesize.awk)
サイズ (行数/ワード数/バイト数)	65/170/1435	9/19/114	4/27/142
コンパイル所要時間　(秒)	0.2+0.5	存在せず	
4,334,340 バイト程度のホームディレクトリのファイル検索所要時間(秒)	0.0+0.1	0.1+0.3	

次に，主として実行速度に焦点をあてた両者の比較を複数の UNIX マシン上で行うことにする。できるだけ検索すべきディレクトリの容量を同一とするため，図 3-1 のようなネットワーク環境において実行速度を測定することにした。

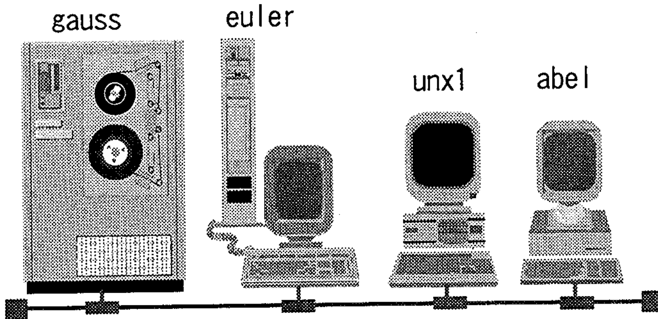

図 3-1　定量比較を行うための UNIX サーバ環境

OLIVE 香川大学学術情報リポジトリ

ここに，gauss および euler はファイルサーバであり，gauss のディスクアレイで実現された大容量ファイルを NFS (Network File System) で euler および unx 1 にリモートマウントしている。従って，同一ディレクトリを euler および unx 1 でも検索できることになっている。一方，abel は gauss/euler/unx 1 などとは異なる CPU を有した UNIX マシンであり，世界的規模で利用されている著名な計算機の 1 つである。比較対照の参考値の 1 つを導出するのに利用しているが，gauss のディスクアレイを NFS マウントしていないので，検索すべきディレクトリおよびその容量は他のマシンでの測定値と異なる。しかし，図 3-1 で示された環境は UNIX サーバを利用する場合の典型的な環境の 1 つを現しており，評価する環境としてはむしろ一般的であると考えられる。

表 3-2 複数の UNIX マシン上でのプログラムとスクリプトとの速度比較

	gauss	euler	unxl	abel
マシン仕様	CPU：R4400MC×2 298MIPS/256MB SVR4.2MP	CPU：R4400SC 149MIPS/128MB SVR4.2	CPU：R3000 33MIPS/128MB SVR4	CPU：microSPARC 75MIPS/32MB 4.3BSD
コンパイル時間	(0.2+0.5)sec	(0.3+0.1)sec	(0.6+0.7)sec	(0.5+0.5)sec
プログラム検索時間(その1)[注1]	(1.1+35.4)sec	(1.0+26.7)sec	(2.1+38.5)sec	—
スクリプト検索時間(その1)[注1]	(6.1+42.3)sec	(9.6+36.3)sec	(27.0+1：08.9)sec	—
プログラム検索時間(その2)[注2]	—	—	—	(0.5+4.0)sec
スクリプト検索時間(その2)[注2]	—	—	—	(5.7+4.5)sec

[注 1] gauss/euler/unxl では合計 506,629,559 バイトのファイル群からなるディレクトリの検索時間

[注 2] abel では合計 188,875,455 バイトのファイル群からなるディレクトリの検索時間

予め予想されていた点であるが，unx 1 は gauss および euler に比べてプログラムおよびスクリプトの実行速度が劣っている。その理由として，unx 1 のマシン性能，特に CPU の違いによる影響が大きいと思われる。実際の測定結果を見るまでの予想と大きく反している結果に，euler 上での実行結果の良さがある。euler の直接管理下のディスクではなく，ネットワークで NFS マウントさ

OLIVE 香川大学学術情報リポジトリ

れたgaussのディスクをアクセスしているにもかかわらず，gaussよりもむしろパフォーマンスが優れているとさえ言える。これは，いくつかの理由が考えられるが，ディスクアクセスの度にネットワークを経由したシステムコールの発生に起因している点を考慮すると，gaussのマルチCPUシステムのチューニングがあまり巧くいっていないか，あるいはgaussのファイルサーバとしての機能の完成度が高く，NFSを他のマシンにほとんど意識させない程度になっているのか，などの要因が考えられる。

表3-2から得られる結論として，

- 実行速度だけから見ると，予想通りプログラムの方がスクリプトよりも高速である
- ファイルを頻繁にアクセスするなど，入出力に絡んだ処理はプログラムとスクリプトとの違いをあまり感じさせない
- スクリプトの実行はプログラムの実行と比較して，ユーザモードでの実行時間が長い
- NFSを利用したファイルアクセスはパフォーマンスの大きな劣化にはつながらない

などが得られることになる。この結論からだけで判断すれば，『ファイルアクセスなど頻繁に入出力が生じる操作ではプログラム記述の所要時間が長いなど開発コストが高くつく方法論を敢えて採用する必要性は少ない』，ということになる。システム管理もこの範疇に含まれるため，短い応答時間を要求される操作を主にスクリプトで実現するという方法論は，開発コストの低減と妥当な実行速度の達成を同時に満足する点で評価できると言える。この結論は一般的にも予想できる答えであり，一部の問題点を除いてシステム管理全般に該当するものと言える。

しかし，予め想定していなかったプログラムでの実行とスクリプトでの実行との違いも認められた。これは具体的に比較検討を行うため，プログラムおよびスクリプトを実行させて，その表示結果までもロギングするという方法で今回の測定を行ったため確認できたと言える。また，プログラムとスクリプトと

OLIVE 香川大学学術情報リポジトリ

の違いに加えて，実行するマシンが異なる場合，同じプログラムやスクリプトであるにもかかわらず表示を含む実行結果が微妙に異なる状況が発生した点にも言及したい。

3-3　比較によって生じた問題点とその考察

当初の予測では，プログラムが最終的に個々のマシンに大きく依存した機械語コードにコンパイルされて実行されるのに対して，スクリプトではマシン独立性が強く移植性などの点でも優位を示すと思われていた。しかし，SVR 系の gauss/euler/unx 1 ではそのまま実行できたスクリプトが，BSD 系の abel 上では実行する度に一度画面をクリアするという現象を生じさせた。結果が異なるといった致命的な問題ではないものの，画面クリアは思考の連続性を妨げる場合もあり，スクリプトを移植する上で何等かの改善策を必要とする。多くの場合，UNIX 個人環境の設定変更か最悪でもシステム設定の変更で解決すると思われるが，スクリプトの利用という観点から見れば移植性の大きな障害と言える。C 言語プログラムを実行する場合には，このような問題は生じていない。一般的にも C 言語の場合，グラフィックスを利用するなどといった特殊な用途の記述がなければ，パソコンからスーパーコンピュータにいたるまで，ほとんど変更なしにプログラムが移植できる点が大きな特徴となっている。

次に，スクリプトの実行において，エラーまたは警告が発せられる点があげられる。UNIX の場合，ファイルにはそれぞれ所有者が存在し，他のユーザからは許可されているないモードではアクセスができない。従って，プログラムであれ，スクリプトであれ，アクセス許可モードの範囲内で実行させることが必要となる。ところが，今回の比較では gauss/euler/unx 1 の総てにおいて，プログラムでは検索できたファイルに対して，スクリプトではアクセスできない事態が発生した。より正確には，スクリプト内でコマンド “/usr/bin/ls” を使用しているが，そのコマンドが特定のファイルに対してアクセス不可能であるというメッセージを表示した点を指摘しなければならない。通常，同コマンドは他のコマンドと比べても使用頻度が高く，多様なオプションが用意されている

OLIVE 香川大学学術情報リポジトリ

ため，利便性の高いコマンドである。しかも，アクセス許可モードに対しても比較的ロバストであり，読出し許可モードにないファイルであってもその存在を確認することが可能である場合も多い。今回の csh シェルスクリプトにおいても記述を簡潔にするため重要な役割を演じさせている。UNIX が用意しているコマンドのなかで，スクリプトに利用される可能性が上位に位置付けられると推移定される。従って，スクリプトをプログラムと比較する上で ls の使用が問題視される点は残念である。もちろん，プログラムであれ，スクリプトであれ，root となって実行すれば許可モードの問題はほとんどクリアされる。しかし，root 権限での実行はかなり慎重に決断すべきであり，筆者の中には苦い経験を持つ人間もいるなど，最大限の注意が必要であろう。結果として，プログラムの優位性を指摘する観点から言えば，既存のコマンドを組み合わせて構成するスクリプトでは使用するコマンドの特性によっては，思わぬ形でスクリプトの限界が生じる例証となる点を指摘しておく必要がある。

C 言語プログラムの優位性を指摘する観点からは，さらにもう1点言及しなければならない。例えば, printf() と fprintf() などの出力関数などの使い分けができる点があげられる。実際，ディレクトリを検索してその配下のファイル名をサイズと共に表示する場合，比較的少量のファイル数であれば画面表示もそれ程苦痛ではない。しかし，指定したディレクトリ配下の個々のファイル名やサイズより合計サイズがいくらかを確認したいだけという場合もありうる。その場合, 個々のファイル名やサイズが大量に表示されるのは無意味であるため，コミ捨てディレクトリといわれる “/dev/null” にリダイレクションして，画面を不要なファイル名やそのファイルサイズで満たすことを避けたいと考える。しかし, 単に標準出力をそのまま /dev/null にリダイレクションすると, 調査対象であるファイルサイズの合計も /dev/null に出力，すなわち捨てられてしまうことになり，不都合が生じる。スクリプトの場合，特に awk スクリプトなどでは，フィルタ系ソフトウェアツールの設計思想から，入力および出力をそれぞれ標準入力および標準出力として処理するよう設計されており，リダイレクションは可能でも標準エラー出力を上手に利用して, 合計ファイルサイズのみ,

OLIVE 香川大学学術情報リポジトリ

標準エラー出力に表示してリダイレクションの影響を免れるという手法を活用できない。

一方，C言語プログラムであれば，ソースコードの中から出力を行う該当箇所を例えばprintf(…)からfprintf(stderr, …)へとわずかに変更するだけ容易に目的を達成できる。当然ながら，変更前の実行速度と変更後のそれとの有意な差は全く認められない。問題への適応性や移植性を向上させるため，状況に応じてソースコードを一部変更できるという条件下でプログラムとスクリプトとを比較すると，どちらもソースコードを変更すること自体は容易であっても，自由度の大きなC言語プログラムに対して，既存のコマンドを利用しているスクリプトではコマンドを利用する分ソースコード自体は短くなるものの，自由度が制限されるという典型的なトレードオフの問題を内在させている。もちろん，後述するように，複数のファイルに対して同時にオープンして読み書きできるスクリプト系の言語であるperlなどを利用すればこの問題も回避できる。perlの持つ利点・欠点を論じることは次節に譲るとして，プログラムに対してスクリプトの利用が優位であると手放しで結論付けできないのも事実である。

4. スクリプトの多様性と適応分野の例示

ここでは筆者たちが日常システム管理に利用しているシェルスクリプト，awkおよびperlのスクリプトを紹介し，スクリプトの特徴および機能について具体的に例示する。また，それぞれのスクリプトが持つ得意分野をカテゴリー分けし，プログラムおよびスクリプトが得意とする適応範囲を具体的に示す。

4-1 スクリプトの具体的使用事例の紹介

基本的にはスクリプトは短いほど好ましいと考えられているので掲載する例は比較的短いものばかりである。しかし，perlのスクリプトは一様に長くなる傾向があり，特徴であると共にperlを使用する上での注意事項となっている。まず最初は，csh用シェルスクリプト（図4-1を参照)を紹介する。これはワークステーションのフロッピーディスクドライブを用いて，MS-DOSでフォー

OLIVE 香川大学学術情報リポジトリ

マットされたフロッピーディスクからテキストファイルを読み書きさせるスクリプトである。このシェルスクリプトの特徴は同じスクリプトテキストを“readfd”および“writefd”という2つのエントリにハードリンクすることにより，スクリプトを起動する際にどちらのコマンド名で呼び出されたかを認識してフロッピーディスクの読み出しか書き込みかを自動識別する点にある。同一テキストコードで2つの異なる機能を実現している。もちろん機能を別個に実現することは容易であるが，処理に必要となるコマンドの利用方法をシェルスクリプトとして1つにパッケージ化することで，利用方法自体をドキュメント化する上で有効となる一例として紹介している。

```
#!/usr/bin/csh
set comid = $0
if ($#argv == 0) then
    if ("$comid" == "readfd") then
        echo "readfd can read SJIS-coded textfiles of MS-DOS from FD(2HD)"
        echo " and convert them into the same named EUC-coded textfiles."
    else
        echo "writefd can convert EUC-coded textfiles into SJIS-coded ones"
        echo " write them as the same named MS-DOS files onto FD(2HD)."
    endif
    echo "Usage:"
    echo $comid " file"
    echo $comid " file1 file2 ..."
else
    foreach file ($argv[*])
        if ("$comid" == "readfd") then
            msread /dev/rif/04 $file | stou -r > $file
        else if ("$comid" == "writefd") then
            utos -rw $file | mswrite /dev/rif/04 $file
        endif
    end
endif
```

図 4-1　csh 用スクリプトの一例

OLIVE 香川大学学術情報リポジトリ


```
(UNIXmachine)staff[1] cat chkPW1.awk
#!/usr/bin/awk -f
BEGIN {
    FS =":";
    while (getline <"/etc/passwd" > 0)
        system("passwd -s " $1);
}
(UNIXmachine)staff[2] cat chkPW2.awk
#!/usr/bin/awk -f
BEGIN { ps = 0; np = 0; lk = 0; }
$2 ~ /PS/ { ps += 1; }
$2 ~ /NP/ { np += 1; print "NoPasswordUser " $1; }
$2 ~ /LK/ { lk += 1; print $0 ; }
END {
    print "Total Users: "NR;
    print "Passwd-setUsers: " ps " NoPasswdUsers: " np " Passwd-lockedUsers: " lk ;
}
(UNIXmachine)staff[3] chmod +x chkPW?.awk
(UNIXmachine)staff[4] ls -F !$
ls -F chkPW?.awk
chkPW1.awk*   chkPW2.awk*
(UNIXmachine)staff[5] su
パスワードを入力してください:
UNIXmachine# chkPW1.awk | chkPW2.awk
daemon   LK
bin   LK
=== 中略 ===
NoPasswordUser chikuse
NoPasswordUser yokoyama
NoPasswordUser s90e734
NoPasswordUser s91e708
NoPasswordUser s94g327
guest   LK
Total Users: 100
Passwd-setUsers: 58 NoPasswdUsers: 26 Passwd-lockedUsers: 16
UNIXmachine# ^D
(UNIXmachine)staff[6] ^D
```

図 4-2　awk スクリプトおよびその実行結果の一例

次は，awk スクリプトおよびその実行結果（図 4-2 を参照）を紹介する。システム管理ではユーザに関する情報を簡単に調査したい局面が頻繁に生じる。

OLIVE 香川大学学術情報リポジトリ

ここでは，あるマシン上に登録されているユーザのパスワードがどのような設定状況になっているかをチェックし結果を表示するスクリプトの具体例を示す。2つの部分から構成されており，スクリプト“chkPW1.awk”というパイプの前段部分では，“/etc/passwd”を読出し，個々のユーザ毎にパスワードの設定状況を調べる。パイプ後段部分のスクリプト“chkPW2.awk”では，各ユーザのパスワードが3つの状態，すなわち「パスワード設定済み」，「ノーパスワード状態」，または「パスワードロック状態」のどの状態であるかを計数して，それぞれのユーザ数やユーザ名を表示している。

最後の例として，perlスクリプト“chkUser.pl”およびその実行結果(図4-3を参照)を紹介する。指定されたUNIXマシンに誰がログインしているかを調べ，そのユーザ名とユーザ数とをマシン毎に調査したいという要求は，システムあるいはネットワークをシャトダウンしたり，警告を発したりする場合など頻繁に生じる。また，複数のマシンの利用状況などの統計処理を行う場合，このような機能を利用できればシステム管理としては有効である。そこで，スクリプト“chkUser.pl”を適当な時間間隔で実行させて統計情報をまとめれば，情報処理センターの各UNIXマシンがどのように利用されているかの状況を的確に把握することが可能になる。このスクリプトを実行することで，ログインしている各ユーザがどのような処理を行っているかもロギング可能である。基本的には，リモートシェルを実行させるコマンド“rsh hostname -l username w”を，指定されたマシン(hostname)毎に実行させた後，その実行結果をセーブしたファイルから必要な情報をフィルタリングして簡単な統計処理を行っている。この場合，問題となるのはスクリプト中で複数のファイルを読み出しあるいは書き込み可能状態にしてオープンする機能をどのように実現するかであるが，このようなファイル入出力機能の充実はperlの大きな特徴となっている[(6)]。もちろん，cshとgrep/sed/awkなどを組合せても同等の機能を実現することは可能であるが，処理を比較的簡潔に表現できる点がperlの評価すべき利点である。

フィルタ系のスクリプト言語と言えば，awkと並んでperlを列挙するのが

OLIVE 香川大学学術情報リポジトリ


```
Script started on Mon Oct  3 20:28:18 1994
1 gauss imai> cat chkUser.pl
#!/usr/local/bin/perl
# Usage
if($#ARGV < 0){
    print "Usage: ", $0, " host[ host ...]¥n";
    exit;
}
# Command Initialization
$com = "rsh fourier -l imai w"; # これはリモートシェルの使用例でもある
@comlist = split(/ /, $com);

# Getting our hostname
system("/usr/ucb/hostname >Hostname");
open(fh, "Hostname");
$hostname = <fh>;
close(fh);
chop($hostname);
# Getting username
system("/usr/ucb/whoami >Username");
open(fh, "Username");
$username = <fh>;
close(fh);
chop($username);
# Setting date & time
system("date '+DATE: %m/%d/%Y TIME: %H:%M:%S'>Result");
# Execution "rsh 'specified machine' -l Username w OR w "
# fetchiing machine name given at ARGV-list
# storing the result of executions onto "Result"
for ($i = 0; $i <= $#ARGV; ++$i){
    $comline = "(echo -n HostName ";
    $comline .= $ARGV[$i];
    $comline .= " ; ";
    if ($hostname eq $ARGV[$i]){
        $comline .= "w";
    }else {
        $comlist[1] = $ARGV[$i];
        $comlist[3] = $username;
        $combuf = join(' ', @comlist);
        $comline .= $combuf;
    }
    $comline .= ")>> Result";
    system($comline);
}
close(fh);
```

OLIVE 香川大学学術情報リポジトリ


```
# Analysis of "Result"
# reporting login-user's names of specified machines
open(fh, "Result");
while ($_ = <fh>){
    if (/^DATE:/){
        $loop = 0;
        print $_;
        next;
    }
    s/   / /g;
    @buf = split;
    if ($buf[0] eq "HostName"){
        if ($loop == 1){
            print " ···· ", $count;
            if ($count <= 1){ print " User¥n"; }
            else { print " Users¥n"; }
            $count = 0;
        }
            print " > ", $buf[1], " : ";
     }
     elsif ($buf[0] eq "User"){ $loop = 1;}
     else{
         ++$count;
         print "¥n              " if $count == 11;
         print "[ ", $buf[0], " ]";
     }
 }
 close(fh);
 print " ···· ", $count;
 if ($count <= 1){ print " User¥n"; }
 else { print " Users¥n"; }
 system("rm Hostname Username");
```

```
2 gauss imai> chmod +x !$
chmod +x chkUser.pl
3 gauss imai> !$
chkUser.pl
Usage: chkUser.pl host[ host ....]
4 gauss imai> !$ fourier gauss euler unix1
chkUser.pl fourier gauss euler unix1
DATE: 10/03/1994 TIME: 20:29:38
 > fourier : [ tominaga ] ···· 1 User
 > gauss : [ imai ] ···· 1 User
 > euler : [ reis0100 ] ···· 1 User
 > unix1 : ···· 0 User
```

OLIVE 香川大学学術情報リポジトリ


```
5 gauss imai> cat Result
DATE: 10/03/1994 TIME: 20:29:38
HostName fourier   11:30am   up 25 days, 22:54,   1 user,   load average: 0.16, 0.04, 0.00
User        tty              login@    idle     JCPU     PCPU    what
tominaga pts/6             10:49am       42          2          1    -tcsh
HostName gauss    8:29pm   up 6 days, 11:22,   1 user,   load average: 1.28, 1.05, 1.01
User        tty              login@    idle     JCPU     PCPU    what
imai        pts/0            8:23pm                  4          1   script perl.log1002
HostName euler   11:33am   up 6 days, 11:23,   1 user,   load average: 1.46, 1.39, 1.03
User        tty              login@    idle     JCPU     PCPU    what
reis0100 pts/3            11:33am                  1               -tcsh
HostName unix1   8:32pm   up 11:30,   0 user,   load average: 0.00, 0.00, 0.00
User        tty              login@   idle    JCPU    PCPU   what
6 gauss imai> ^D
script done on Mon Oct   3 20:31:58 1994
```

図 4-3 perl スクリプトおよび UNIX マシン利用状況の簡易統計処理の一例

一般的傾向と言える。しかし，perl はその記述能力の高さからか，どうしてもスクリプトサイズが大きくなってしまう傾向がある。常識的な観点から言っても，既にプログラムと呼ぶべきサイズである。また，C 言語プログラムと比較しても遜色のない作成所要時間を覚悟する必要がある。機能実現するための開発コストがC言語に比肩しうる点に十分注意を払う必要があり，他のスクリプト言語と一括して議論できない要素を有していることを指摘したい。このようにプログラム対スクリプトの構図のみならず，各種スクリプト言語間においても得意の分野が何であるかを十分に把握して使用することが必要である。これはひとりシステム管理の問題のみならず，便利なツールをいかに有効かつ効果的に利活用するかという不可欠の知識であろう。

4-2 プログラムおよび各種スクリプトの適応範囲のカテゴリーについて

C 言語プログラム，csh や sed/awk あるいは perl などのスクリプトの特徴，機能そして利用対象などを定性的に比較対照すると次のようにまとめられる。システム管理者のみならず，一般のユーザにとっても，問題に応じてプログラム記述を選ぶか，スクリプト作成を選択するか，を判断する指針は有効である。

OLIVE 香川大学学術情報リポジトリ

また，スクリプト系と一括される傾向にある各種スクリプト言語が，どのような問題に利用した場合に適応性を発揮するかを具体的に言及したい。

1)　実行効率が重要視される処理記述に適したC言語プログラム

バイナリファイルが，ロード時においてもコード実行時においても，高速であるのは常識である。しかも，C言語プログラムとなれば最適化コンパイラが用意されており，複数のコンパイルフェーズで最適化され，様々なレベルでのコードチューニングも可能である。実行効率が最も重要視されるコマンド記述ではほとんど総てのスクリプト言語よりも有効である。仮にプログラム開発の所要時間がハイコストであるとしても，頻繁に利用されるコマンドの実現には不可避の選択肢である。従って，C言語などのプログラムとして記述し，最適化コンパイラでバイナリコードを生成するのが実行速度を重要視する場合の一般的選択である。システムコールなどを有効に利用することも優位であり，UNIX を利用する上では『最後の切り札的存在』である。プログラムのバージョンアップも必要となるが，UNIX 環境であれば make や sccs/rcs などの優秀なプログラム開発ツールが用意されており，一度開発すればメンテナンスのコストなどの問題は比較的少ないと言える。

2)　並行処理記述などシステム制御に適したシェルスクリプト

C言語では関数 fork()や exec()を用いて簡単に並行処理を記述できる。このような処理が UNIX 環境では大きな処理効果を生む場合も少なくない。C言語でのプログラミングがシステム記述に適している点は衆目の一致するところである。個々の操作をタスク（プロセス）レベルで並行処理させ，きめの細かいタスク制御を実現しようとすれば，やはりC言語でのプログラミングということになる。しかし，システム管理上，粒度の細かいタスクレベルの平行処理を要求される場合はきわめて希である。どちらかと言えば，要求される操作の単位がコマンドレベル（ジョブレベル）である状況が一般的である。とすれば，C言語プログラム内で関数 system()などを多用し，パラメータの引き渡しに工夫しながら各々のコマンドを呼び出すより，シェルスクリプトにおいて簡潔にコマンドレベルの平行処理を記述するという選択肢の方が，目的の処理を実

OLIVE 香川大学学術情報リポジトリ

現する所要時間は確実に短くなる。シェルスクリプトであれば，強力な処理記述方法であるパイプラインやリダイレクションといった表現を容易に採用できる点も大きな特徴と言える。

3) 文字列処理に適したawkスクリプト

テキスト処理指向の強いUNIXでは文字列をどのように扱うかでプログラム作成および実行の効率が大きく左右される。C言語に文字列処理関数が多く用意されているのも例証と言える。awkスクリプトはC言語の制御構造に似た記述が可能であり，C言語でのプログラミング経験があればawkのスクリプト記述も短期間で習得できる。フィルタ系のコマンド“/usr/bin/sed”や“/usr/bin/grep”などと組合せると，C言語でのプログラムよりも目的の処理を簡潔に実現できる。特に，文字列処理を行うフィルタ関連の処理には使用効果が高い。

4) C言語プログラム以上に強力な記述能力を持つperlスクリプト

perlはオールマイティな処理系であり，状況によってはC言語以上の記述能力を有している。DOS環境でも適切に動作するawkに対して，perlもDOS環境に移植され一部の機能は利用できるものの，やはりUNIX環境で初めて効果を発揮するスケールを有する。perlスクリプトの場合，数行で望みの処理を実現することもできるが，その記述能力の高さからほとんど汎用のプログラミング言語と同じように使用される傾向にある。フィルタの作成からシステム制御コマンドの記述まで，幅広い適応範囲がperlの特徴と言える。

$$perl \geq \sum \{csh + grep/sed/awk\}$$

という図式が成り立ち，コンパイラ指向ならC言語，インタプリタ指向ならperlと言っても過言でない広い守備範囲が大きな特徴である。perlをスクリプト系だからという理由だけでcshやawkと同列に扱うべきではないという使用上の認識も必要である。

OLIVE 香川大学学術情報リポジトリ

5. おわりに

csh/awk 等のスクリプトが持つ最大の特徴は，デバッグが簡単でスクリプト作成に要する時間が短くて済む点である。原理的にはコンパイル時間も存在せず，実行効率（実行速度）があまり重要視されない状況ではスクリプトの特徴である記述から実行へのスムーズな流れが大きな利点であり，システム管理に適した使用形態を提供してくれる。ソースファイルとバイナリファイル，時にはオブジェクトやライブラリが必要になるC言語プログラムとは異なった次元に存在する。もちろん csh/awk/perl などは汎用性の高い処理系であり，当然ながら時間と工夫でいくらでも高機能で用途の広いスクリプトを作成できる。しかし，結果としてサイズも長大となり，実行速度のみならず短い作成所要時間という最大の利点を享受できないことになる。従って，そのようなサイズのスクリプトは既にプログラミング言語の扱うべき範疇を侵していると言うべきであろう。特に awk などでは1行スクリプト (one liner) の効用が強調されるほどである[5]。長大なスクリプト記述は，本来の趣旨から言っても，あまり望ましいものではない。

csh や grep/sed/awk などと，（機能的にはperlも含まれる）プログラミング言語とを，目的と要求される状況(所要時間や実行効率など)に応じて適切に選択し，上手に使い分けることが UNIX 環境では重要である。システム管理のみに限らず，プログラムとスクリプトをいかに活用するかが，要求される仕事の処理効率を大きく左右すると言える。この点では，C言語と同様に grep/sed/awk/perl など UNIX 上の多くのユーティリティが移植された DOS 環境などでも同様と言える。

今回の論説をまとめる上でいくつかの課題が生じたのでアットランダムに列挙する。例えば，

- システム管理という観点からファイル入出力に対象を限定したが，一般的にはプログラムの実行とスクリプトの実行とではどちらが優位になるか

OLIVE 香川大学学術情報リポジトリ

- C言語のプログラムとperlのスクリプトとの定量的比較はどのような結果となるか
- awkやperlの場合，スクリプトをC言語のプログラムへコード変換するツールが存在するがその効率はどうか
- プログラムの記述とスクリプトの作成との所要時間の統計的比較は可能か

などでである。確かに，これらの問題はシステム管理[(7)]をスクリプトベースで行うか，プログラムベースで実現するかを比較検討する上で重要な判断材料になりうる問題と言える。ファイル入出力はどんなに高速なCPUを使用してプログラムを実行しても，プロセスにかなりの待ち状態を発生させてしまう。従って，実行効率を低下させる要因となり，スクリプトに対するプログラムの優位を印象づけない結果となった。本稿の結論の１つである『システム管理には多くの場合，開発コストおよび実行速度を総合して判断すると，スクリプト作成の方がプログラム記述よりも優位である』，という結論を導出させる遠因となった。C言語プログラムとperlスクリプトとはどちらも大きなサイズのテキストとなる傾向にある。しかし，同程度の処理能力を表現するテキストサイズはどちらが短いか，あるいは同程度のテキストサイズでの処理能力やその実行時間の定量的比較はどうかなどの課題も発生した。機会があれば授業やゼミなどの実習を利用してプログラム記述とスクリプト作成との所要時間についても可能な限り統計的に比較してみたいと思っている。これらについては今後の課題とし，いくつかの結論が導出された時点でまとめて報告したい。

謝　辞

㈱技術評論社SoftwareDesign編集部の谷戸伸好氏には本稿を執筆する上で示唆に富んだアドバイスをいただきました。㈱関西日本電気ソフトウェアの武藤直美氏および㈱四国日本電気ソフトウェアの中沢美弥子氏にはUNIXのシステム管理を行う上でいつも重要なアドバイスをいただいています。情報処理センターの瀬野芳孝氏，曽根計俊氏，および丸山久美子氏は日頃からセンターマシンの管理を一緒に行っている言わば『仲間』であり，３氏の協力なくしては

OLIVE 香川大学学術情報リポジトリ

システム管理などできない状態です。情報管理学科教授宍戸栄徳先生には様々な局面で有用な助言をいただいています。同学科助教授富永浩之先生にはawk[8]をはじめ，興味深い話題をいつも提供していただいています。ここに記して感謝の意を表します。

参考文献

(1) Kernighan, B. and Richie, D. (1988) "The C Programming Language (2nd Ed.)," Prentice-Hall (石田晴久訳 (1989)『プログラミング言語C (第2版)』，共立出版)

(2) Kernighan, B. and Pike, R (1984) "The UNIX Programming Environment," Prentice-Hall (石田晴久監訳 (1985)『UNIXプログラミング環境』，アスキー出版局)

(3) Rochkind, M. (1985) "Advanced UNIX Programming," Prentice-Hall (福崎俊博訳 (1987)『UNIXシステムコール・プログラミング』，アスキー出版局)

(4) Anderson, G. and Anderson, P. (1986) "The UNIX C Shell Field Guide," Prentice-Hall (落水浩一郎・大木敦雄訳 (1987)『UNIX C SHELLフィールドガイド』，パーソナルメディア)

(5) Aho, A., Kernighan, B. and Weinberger, P. (1988) "The AWK Programming Language," Addison-Wesley (足立高徳訳『プログラミング言語AWK』，アジソンウェスレイトッパン)

(6) Wall, L. and Schwaltz, R. (1990) "Programming Perl," O'Reilly and Associates (近藤嘉雪訳 (1993)『Perlプログラミング』，ソフトバンク)

(7) Fiedler, D. and Hunter, B. (1991) "UNIX System V Release 4 Administration (2nd Ed.)," Sams Corp (中原紀訳 (1993)『UNIX SVR4システム管理』，HBJ出版局)

(8) 植村富士夫 and 富永浩之 (1993)『awkでプログラミング』，オーム社