4L-7 OS/omicronにおけるプログラム生成手順とその仕様記述ファイル

並木美太郎，高橋延匡
(東京農工大学・工学部・数理情報工学科)

1．はじめに

当研究室では、日本語を用いたプログラミング環境とそのためのOSであるOS/omicron[1,2]を開発している。大規模プログラムの開発には、ソースコード管理が必要不可欠な要素である。しかし、ソースコード管理システムでなくとも、コンパイルすべきファイルのコンパイル・リンク手順の明確化・文書化は必要な機能である。例えば、変更のあったソースファイルのみをコンパイルし、無関係なリロケータブルオブジェクトモジュールはそのままリンクする機能があると便利である。さらに、それがソフトウェア文書の一部として使用できれば、すこぶる使いやすい。

既存のシステムでは、例えば、UNIXではプログラムの生成手順を記述し、その手順にそってプログラムをコンパイル・リンクするプログラムmakeを備えている。しかし、このmakeは記述が煩雑な上、理解しにくい。仕様書としての可読性は低い。本報告では、

(1)プログラムの生成手順を日本語を用い、仕様書のごとく記述し、
(2)さらにその手順にそってコンパイル・リンクするためのファイルの仕様とそのためのプログラム

について述べる。

2．設計方針

通常、ソースプログラムからロードモジュールを生成する場合、次の点に着目するであろう。

あるプログラムは，
ソースプログラム1，…ソースプログラムnからなり，
手順1，…手順nによって作られる．

このようなプログラムの生成手順が、

(1)日本語で記述できる
(2)記述から、ソースプログラム、リロケータブルオブジェクト、ロードモジュールのそれぞれを生成する
(3)さらに、その作用が変更のあったファイルにのみなされる

と都合がよい。本システムでは、上記3点を設計方針とする。

3．生成手順ファイルの構文

3．1 基本形

生成手順ファイルの基本形は次の構文で与えられる。

[構文1]
〈対象名〉 は
〈構成要素のならび〉 からなり
〈手段〉 によって
作成される ．

ゴシック体は予約語(生成手順ファイルにはゴシック体として記述されない)、各語は分かち書きとする。ピリオド“.”は、文の区切りを表わす。この構文で定義された〈対象名〉と〈構成要素〉から、あるプログラムのソースプログラムの構成要素を構築する。例を次に示す。なお、ＯＳ／ｏはフル２バイトコードを内部コードとしているので、ファイル名は日本語(JIS X0208)になっている。また、{}でくくられた行は、コマンドインタープリタのコマンド行、「」でくくられたものは、コメントを意味している。

[例1]
「次の内容は例1であり、この文はコメントである。」
CAT は トークン.o コードジェネレータ.o からなり
{
link パーザ.o コードジェネレータ.o -exec CAT
} によって 作成される．
トークン.o は ヘッダ.h トークン.c からなり
{
cat トークン.c
} によって 作成される．
コードジェネレータ.o は
ヘッダ.h コードジェネレータ.c からなり
{
cat コードジェネレータ.c
} によって 作成される．

この例は、CATというプログラムは、トークン.oとコードジェネレータ.oをオブジェクトをリンクして作られ、トークン.oとコードジェネレータ.oはcatによってコンパイルされることを意味している。

上記を仕様記述ファイルとして手順実行プログラムの入力ファイルにする。手順実行プログラムでは、仕様記述ファイルの内容にそって解釈・実行を行なう。

3．2 構成要素と手段の分離、正規表現

例1では、“トークン.o”と“コードジェネレータ.o”の手順(cat)は同じであったにもかかわらず、個別に書いた。しかし、同じ手段を用いるものに対しては、その記述は一個所で行いたい。そこで、〈手段〉の定義を分離し、〈対象名〉に正規表現を許す。“*”は任意の文字列と、“?”は任意の1文字とマッチする。〈構成要素〉と〈手段〉の中では、正規表現は許されないが、〈対象名〉でマッチした文字を$0から$9により参照することが可能である。$0はマッチした部分全体を、$1から$9はN番目の正規文字を返す。

先の例1の“トークン.o”と“コードジェネレータ.o”は次のように記述できる。

[例2]
トークン.o は ヘッダ.h トークン.c からなる ．
コードジェネレータ.o は
ヘッダ.h コードジェネレータ.c からなる ．
*.o は
{ cat $1.c } によって 作成される ．

〈対象名〉に対する構文は、次の通りである。

[構文2]
〈正規表現を含んだ対象名〉 は
[〈構成要素のならび〉 からなり]
[〈手段〉 によって]
作成される ．
|
〈正規表現を含んだ対象名〉 は
[〈構成要素のならび〉] からなる ．

3．3 実行名

例1、例2では、〈手段〉は{}でくくってコマンドを指定していた。しかし、種々の環境で様々なコンパイラを使っている時は、名前でコマンドを定義したい。そこで、次の構文を用意した。

Config - A Program For Configrating Programs From Its Documentation in Japanese on OS/omicron
Mitarou NAMIKI, Nobumasa TAKAHASHI
Tokyo Univ. of Agriculture and Technology, Dept. of Information Science

```
「このファイルは日本語コンフィグレータの生成手順である。 1988年5月1日 並木美太郎　改版:1988年07月04日 早川栄一 」
config は
 main.o regmatch.o errmess.o token.o reserve.o define.o deftable.o regexp.o istrsave.o newtime.o execobj.o
 change.o getch.o regcexp.o strsave.o nsystem.o loadgo2.o　からなり　{
     /command/linker -list object.p -list /library/osolib.p  -exec config
     /command/patchload config 50 16 12
} によって 作成される .
「構成とその作成方法の記述」
「エントリポイント。 引数の解析など。」
main.o は main.c type.h error.h からなる .
「生成すべきファイル名から構成要素を得、コマンドを実行するためのトップレベルの関数を含んだモジュール」
execobj.o は execobj.c type.h struct.h attr.h token.h error.h からなる .
                        <中略>
*.o は CAT によって 作成される .
「コンパイラはcatである。」
CAT は { /command/ccat $1 } を 実行する .
```

図1　プログラム生成手順ファイルの例

```
[構文3]
  〈実行名〉 は 〈コマンド〉 を 実行する .
      |
  〈実行名〉 は 〈実行名〉 を 実行する .
```

なお、〈コマンド〉は { }でくくられたものである。例2は次の形に記述できる。

```
[例3]
  トークン.o は ヘッダ.h トークン.c からなる .
  コードジェネレータ.o は
  ヘッダ.h コードジェネレータ.c からなる .
  *.o は Cコンパイラ によって 作成される .
  Cコンパイラ は { cat $1.c } を 実行する .
```

3．4 マクロ

デバッグや最適化のフラグなどがコンパイルオプションとして指定されることがある。このようなオプションは変更などの操作を考慮すると、〈手段〉の中に埋め込むのは、好ましいことではない。そこで、マクロ置換を用意した。

```
[構文4]
  〈マクロ名〉 は
  〈語"である"までの語の並び〉 である .
```

で与えられ、置換を行いたい個所で "[マクロ名]"とする。なお、この置換は、〈対象名〉以外の任意の場所で使用することができる。例を次に示す。

```
[例4]
  コンパイルフラグ は -j である .
  ヘッダファイル は ヘッダ.h である .
  トークン.o は
  [ヘッダファイル] トークン.c からなる .
  コードジェネレータ.o は
  [ヘッダファイル] コードジェネレータ.c からなる .
  「これはコメントである。上記の[ヘッダファイル]が
   ヘッダ.hにおきかわる」
  *.o は Cコンパイラ によって 作成される .
  Cコンパイラ は
  { cat [コンパイルフラグ] $1.c } を 実行する .
  「コンパイルフラグが-jにおきかわる」
```

3．5 目的名

一つの生成ファイル中に、行うべき作業を複数個記述したい場合がある。例えば、1)ディレクトリ"美太郎"からディレクトリ"並木"の下にファイル"config.c"をコピーし、2)"config.c"をコンパイル・リンクし、3)"config.c"を消去したいとする。無論、各ステップをまとめて記述すればよいが、可読性が悪い。処理内容毎に名前を付けたい。そこで、次の構文を用意した。

```
[構文5]
  〈目的名〉 は
  〈目的のならび〉 を 目的とする .
  〈目的のならび〉 ::=
    [〈対象名〉 | 〈実行名〉 | 〈 目的名〉 ]...
```

この構文を用いると、先の例は次のように記述できる。

```
[例5]
  このファイル は
  ファイル複写 config ファイル消去
  を 目的とする .
  ファイル複写 は
  { copy 美太郎/config.c 並木} を 実行する .
  config は
    並木/config.c からなり
  {
    cat config.c
    link config.o } によって 作成される .
  ファイル消去 は
  { era 並木/config.c } を 実行する .
```

4．実現に関して

まず、手順実行プログラムは、生成手順の仕様ファイルを読む。それから、仕様ファイル中に記述されている〈目的〉と〈構成要素〉から、ある目的を達成するために必要な〈構成要素〉のファイルの木を作る。この木に表われるファイルの作成された時間関係を調べてみると、上位のものは下位のものから作成されたわけだから、かならず、上位の時間の方が遅い。従って、もし、下位のものが変更されるとその時間関係は逆転することになる。これにより、変更のあったファイルを見つけることができる。その後は、変更のあった〈構成要素〉から作成されるものを〈手順〉によって作成していけばよい。

この手順実行プログラムの生成仕様ファイル(一部)を図1に示す。

5．おわりに

日本語による生成手順ファイルについて述べた。図1に示したように、生成手順が日本語で記述できる効果は大きい。我々の最終目標は、日本語を用いたプログラム環境である。本報告のプログラムの他、フル2バイトコードを標準とした言語Cコンパイラ CAT[3]がＯＳ／ｏ上で稼働しており、日本語ワードプロセッサをprogrammer's work benchとして利用できる環境[4]となっている。

参考文献

[1]鈴木他,"OS/omicronにおける日本語プログラミング環境",情処学コンピュータ・システム・シンポジウム, 1987.11.

[2]高橋延匡,"研究プロジェクト総説:OS/omicronの開発", 情処学OS研資39-5,1988.6.

[3]並木他,"OS/omicron用システム記述言語C処理系catのソフトウェア工学的見地からの方式設計",信学論Vol J71-D,NO.4,pp652-660,1988.4.

[4]並木他,"ＯＳ／ｏにおける日本語プログラミング環境と日本語ワードプロセッサのPWB化",信学論Vol J71-D,NO.6,pp994-1003,1988.6.