makefile 入門

システム運用係　　田川　善教

複数ファイルに分かれているプログラムをコンパイルするとき、更新したファイルのみをコンパイルすると時間の節約となり、効率よく開発を行うことができます。このような機能を提供してくれるツールが make です。make は設定ファイル（makefile）を作成することで、関係するファイルの更新日時を比較して、必要なものだけを実行します。例えば、あるオブジェクトに対してソースプログラムの更新日時が新しい場合にはそのプログラムのコンパイルを行い、更新日時が古い場合にはコンパイルを省略します。

また、makefile を用いることで、コンパイルの度に長いコマンドを毎回入力する必要がなくなり、入力ミスを防ぐことができます。

makefile を作成する

ここでは以下のファイルを仮定します。（各ファイルのサンプルは本稿の後に記載してあります。また、本稿の内容については SR8000/MPP にて動作を確認済です。）

main.c	（メインプログラム）
header.h	（main.c に include されるヘッダーファイル）
sub.c	（メインから呼ばれるサブルーチンプログラム）

makefile を作成します。上記のプログラムと同じディレクトリーに Makefile（または makefile）という名前でファイルを作成し、以下のように記述します。

Makefile 記述例１（# 以降はコメント）

```
a.out : main.o sub.o                    # ターゲット a.out と依存するファイル
<--TAB-->cc –o a.out main.o sub.o –lm      # a.out を作成するコマンド

main.o : main.c header.h                # ターゲット main.o と依存するファイル
<--TAB-->cc –c main.c                   # main.o を作成するコマンド

sub.o : sub.c                           # ターゲット sub.o と依存するファイル
<--TAB-->cc –c sub.c                    # sub.o を作成するコマンド
```

※<--TAB-->は TAB キーを１回入力する。

設定ファイルに従い、プログラムのコンパイルを行います。次のコマンドを入力します。

```
% make
```

これで a.out が作成されます。このとき main.o や sub.o がない場合はそれぞれコンパイルを行います。また、ターゲットに対して依存ファイルの更新日時が新しい場合もそのターゲットを作成します。例えば前回の make の実行から header.h のみを更新した状態で make を実行すると、main.o と a.out のみ作成を行います。（sub.o は既に作成済みのため省略される。）

※上記のような makefile の動作確認を行うためには、依存ファイルの更新が必要となりますが、ファイルの日時のみの更新であれば touch コマンドを使用できます。

例：
```
% touch sub.c header.h
% make
```

makefile の書式

記述例１の 1,2 行目を例にとって説明します。

```
a.out : main.o sub.o
<--TAB-->cc –o a.out main.o sub.o
```

１行目のコロンの左側をターゲットと呼び、make で作成するもの（a.out）を指定します。１行目のコロンの右側にはターゲットが依存するファイル（main.o sub.o）を指定します。2 行目は a.out を作成するためのコマンドで、TAB キーを１度入力してから記述します。

なお、記述例１にあるとおり a.out の作成が目的の場合は、その記述を他のターゲットより先に makefile に記述する必要があります。

マクロ機能

makefile のマクロ機能を使用すると、記述例１は次のように書き換えられます。

Makefile 記述例２

```
CC = cc                                 #
LIB = -lm                               # マクロの定義部
OBJECTS = main.o sub.o                  #

a.out : ${OBJECTS}
<--TAB-->${CC} –o $@ ${OBJECTS} ${LIB}   # $@はターゲット(a.out)を表す

main.o : main.c header.h
<--TAB-->${CC} –c main.c

sub.o : sub.c
<--TAB-->${CC} –c sub.c
```

マクロは makefile の最初に定義し、引用する際は${}または$()で囲みます。"OBJECTS"のように対象を複数指定するときは空白で区切ります。このようにマクロを使用することで、プログラムやオプションの追加など、設定を変更して実行する場合に便利です。

サフィックスルール

サフィックスルールを使用すると、同じ拡張子のものをまとめて記述することができます。対象となるファイルが多数あるときなどに有効です。記述例２を書き換えたものを示します。

Makefile 記述例３

```
CC = cc
LIB = -lm
OBJECTS = main.o sub.o

a.out : ${OBJECTS}
<--TAB-->${CC} –o $@ ${OBJECTS} ${LIB}

.c.o :                          # .c から .o を作成するルール
<--TAB-->${CC} –c $<            # .c から .o を作成するコマンド
                                # ($< は予約されたマクロで依存するファイルを表す)
main.o : header.h               # ルール以外の依存関係を記述
```

記述例３では.c ファイルから.o ファイルを作成するルールを記述しています。この場合、main.o，sub.o の依存ファイルはそれぞれ main.c，sub,c とみなします。main.o と header.h の関係は別途記述します。

その他

任意のファイル名を makefile として使用する場合は -f オプションで指定します。
例）make –f *filename*

サンプルプログラム

・header.h

```
#define  NUM  10
```

・main.c

```
#include "header.h"
main()
{
     int a;
     a = calc(NUM);
     printf("%d¥n",a);
}
```

・sub.c

```
#include <math.h>
int calc(x)
int x;
{
     return pow(2,x);
}
```

※本プログラムをコンパイルして a.out を実行すると、"1024"が出力されます。