共有ライブラリの作成方法と注意点

東京大学情報基盤センター
黒田 久泰

静的ライブラリでは、実行プログラムにライブラリを含めた形でリンクされます。共有ライブラリでは、実行プログラムにライブラリは含まれておらず実行するときにライブラリがロードされます。そのため共有ライブラリをリンクして作成したプログラムの方がプログラムサイズは小さくなります。また、複数のプログラムが共通のライブラリを利用する場合、共有ライブラリを利用するとディスクスペースの節約になります。

ここでは説明をわかりやすくするため最適化オプションは省略していますが、実際のコンパイルでは最適化オプションも忘れずに指定してください。

1. 共有ライブラリの作成方法

ここでは例として、ライブラリのソースファイル名を sub.c、共有ライブラリ名を libsub.so、メインプログラムのソースファイル名を main.c、実行可能ファイルを main として説明します。

・日立最適化 C コンパイラの場合

1. 共有ライブラリのオブジェクトファイルの作成
   ```
   % cc -64 -c sub.c
   ```
2. 共有ライブラリの作成
   ```
   % ld -b64 -G -bnoentry -bexpall -o libsub.so sub.o
   ```
3. 実行可能ファイルの作成
   ```
   % cc -64 -o main main.c libsub.so
   ```

・GNU コンパイラの場合

1. 共有ライブラリのオブジェクトファイルの作成
   ```
   % gcc -maix64 -fpic -c sub.c
   ```
2. 共有ライブラリの作成
   ```
   % gcc -maix64 -shared -o libsub.so sub.o
   ```
3. 実行可能ファイルの作成
   ```
   % gcc -maix64 -o main main.c libsub.so
   ```

日立最適化 FORTRAN90 の場合は日立最適化 C コンパイラの場合と同様です。ライブラリのソースファイル名を sub.f、メインプログラムのソースファイル名を main.f とした場合、下記のようにします。

```
% f90 -64 -c sub.f
% ld -b64 -G -bnoentry -bexpall -o libsub.so sub.o
% f90 -64 -o main main.f libsub.so
```

それぞれのオプションの詳細については、man コマンドあるいはマニュアル等をご覧ください。

2. プログラム実行時の注意点

共有ライブラリをリンクして作成したプログラムを実行するときには注意しておくことがあります。例えば、次のようなエラーが出力された場合には、環境変数の設定が正しく行われていないことが原因です。

```
Could not load program ./main:
Dependent module libsub.so could not be loaded.
Could not load module libsub.so.
System error: No such file or directory
```

次のようなエラーが出力された場合は、共有ライブラリのファイルの属性が正しく設定されていないことが原因です。

```
Could not load program ./main:
Dependent module ./libsub.so could not be loaded.
System error: Exec format error
```

2.1 環境変数の設定

環境変数 LIBPATH または LD_LIBRARY_PATH に共有ライブラリを探すディレクトリを指定します。カレントディレクトリとディレクトリ~/MyLib と~/MyLib2 に共有ライブラリが置かれている場合には、

```
% setenv LIBPATH .:$HOME/MyLib:$HOME/MyLib2
```

あるいは、

```
% setenv LD_LIBRARY_PATH .:$HOME/MyLib:$HOME/MyLib2
```

のように指定します。複数のディレクトリを指定する場合、その間にコロン(:)を記述します。また、csh 用設定ファイル ~/.cshrc に登録しておくと、ログイン時に自動的に反映されます。

2.2 共有ライブラリのファイルの属性設定

本センターのSR11000 上で共有ライブラリを利用する場合、共有ライブラリのファイル属性のotherにread権限が必要です。具体的には、chmodコマンドを用いて、

```
% chmod o+r 共有ライブラリのファイル名
```

のように設定する必要があります。これは、本センターの SR11000 が性能を重視したファイルシステムを採用しているために生じる制約となっています。フリーソフトなどをインストールした場合には共有ライブラリが使われていることがあります。こういう場合には、共有ライブラリのファイルの属性設定を自ら行う必要があるため特に注意が必要です。

なお、共有ライブラリが置かれているディレクトリに対しては、other に read 権限及び execute 権限のどちらも許可する必要はありません。また、32 ビートモードで作成した共有ライブラリの場合にもファイル属性の other に read 権限の許可を与える必要はありません。

3. 共有ライブラリをプログラムから動的に呼び出す方法

共有ライブラリはプログラム中から動的に呼び出して利用することもできます。この場合、リンク時に共有ライブラリ名を指定する必要はありません。ただし、この場合でも、環境変数の設定及び共有ライブラリのファイル属性の other に read 権限の許可を与えておく必要があります。

ここではプログラムから共有ライブラリを動的に呼び出すサンプルプログラムを紹介します。sub.c は共有ライブラリのプログラムで、フィボナッチ数列の値を返す関数 fib()が含まれています。メインプログラムである main.c は 10 番目のフィボナッチ数列の値を求めるために共有ライブラリ内にある関数 fib()を呼び出してその結果を出力します。

sub.c（共有ライブラリのプログラム）

```
int fib(int i)
{
   if(i<0) return -1;
   if(i==0) return 0;
   if(i==1) return 1;
   return fib(i-2)+fib(i-1);
}
```

・共有ライブラリのプログラムの説明

fib(i) は i 番目のフィボナッチ数列の値を返す関数です。

・メインプログラムの説明

dlopen()は共有ライブラリをオープンします。

dlerror()は動的リンク処理中に発生した最後のエラーを記述する文字列が入ります。

dlsym()は共有ライブラリ内のシンボルのアドレスを取得します。

dlclose()は共有ライブラリをクローズします。

main.c（メインプログラム）

```
#include <stdio.h>
#include <dlfcn.h>
int main(void)
{
   int i;
   void *handle;
   char *error;
   int (*fib)(int);
   handle=dlopen("libsub.so",RTLD_LAZY);
   if(!handle){
      fprintf(stderr,"%s¥n",dlerror());
      return 1;
   }
   fib=dlsym(handle,"fib");
   if((error=dlerror())!=NULL){
      fprintf(stderr,"%s¥n",error);
      return 1;
   }
   i=(*fib)(10);
   printf("fib(10)=%d¥n",i);
   dlclose(handle);
   return 0;
}
```

コンパイルと実行の方法は下記の通りです。

```
% cc -64 -c sub.c
% ld -b64 -G -bnoentry -bexpall -o libsub.so sub.o     ←共有ライブラリを作成
% cc -64 -o main main.c          ←リンク時に共有ライブラリを指定する必要はない
% chmod o+r libsub.so            ←共有ライブラリのファイル属性の変更は必要
% setenv LIBPATH .:$LIBPATH     ←カレントディレクトリ「.」がすでに登録されている場合は不要
% ./main
fib(10)=55
```

正しく 10 番目のフィボナッチ数列の値である 55 が表示されていることがわかります。