2024 年ソフトウェア演習2B

第３ 回課題

B243392 ALCANDER IMAWAN 2024 年 6 月 25 日

見やすくするために、Report2 のプログラムをいかに添付した。

プログラム

Message.h

```
1: #include <string>
2: #include <vector>
3: #include <iostream>
4:
5: class Message {
6:
7: private:
8:         std::vector<std::string> message;
9:
10: public:
11:         Message(); // Constructor
12:         Message(const std::string& message_string);
13:         Message(const std::vector<std::string>& message_vector);
14:
15:         ~Message(); //Destructor
16:
17:         void addMessage(const std::string& message_string);
18:         std::string getMessage(int message_id);
19:         void showAllMessages(void);
20:         int getNMessages(void);
21: };
```

Message.cpp

```
1: #include "Message.h"
2: #include <string.h>
3: #include <string>
4: #include <stdlib.h>
5:
6:
7: // Constructer initializing
8: Message::Message(){}
9:
10: //single message constructor
11: Message::Message(const std::string& message_string){
12:         message.push_back(message_string);
13: }
14:
15: //multiple vector messages constructor
16: Message::Message(const std::vector<std::string>& message_vector){
17:         message = message_vector;
18: }
19:
20: Message::~Message(){}//Destructor
21:
22: //add ma message to the list
23: void Message::addMessage(const std::string& message_string){
24:         message.push_back(message_string);
25: }
26:
27: //get a message by message_id
28: std::string Message::getMessage(int message_id){
29:         if(message_id >= 0 && message_id < message.size()){
30:                 return message[message_id];
31:         }
```

```
32:         else {
33:                 return "Message ID not found";
34:         }
35: }
36:
37: //showing all messages
38: void Message::showAllMessages(void){
39:         for(const auto& msg : message){
40:                 std::cout << msg << std::endl;
41:         }
42: }
43:
44: //get the num of messages
45: int Message::getNMessages(void){
46:         return message.size();
47: }
```

main.cpp

```
1: #include "Message.h"
2:
3: int main (int argc, char *argv[]){
4:         //testing default constructor
5:         Message obj1;
6:         obj1.addMessage("Hello World.");
7:         obj1.addMessage("Hello 2nd one¥n");
8:         std::cout << "Number of messages: " << obj1.getNMessages() << std::endl;
9:         obj1.showAllMessages();
10:
11:          //testing single message constructor
12:          Message obj2("This is single message constructor¥n");
13:          obj2.showAllMessages();
14:
15:          //testing vector of messages constructor
```

```
16:         std::vector<std::string> vec = {"1st message", "2nd message", "3rd message"};
17:         Message obj3(vec);
18:         std::cout << "Number of vector messages: " << obj3.getNMessages() << std::endl;
19:         obj3.showAllMessages();
20:
21:         //testing getMessage
22:         std::cout << "testing getMessage for index 1 vector : " << obj3.getMessage(1) <<
std::endl;
23:
24:         return 0;
25: }
```

Q1

静的リンクライブラリの作成方法は以下の手順で示す。

1. ソースプログラムをオブジェクトファイルにコンパイルする。’.cpp’ファイルを’.o’ファイルにコンパイルする。
2. 静的リンクライブラリを作成する。
3. 静的リンクライブラリを’main.cpp’にリンクする。

コマンドラインは以下の通りになる。

1. g++ -std=c++11 -c message.cpp -o message.o　//message.o を作成する。
2. g++ -std=c++11 -c main.cpp -o main.o //main.o を作成する。
3. ar rcs libmessage.a message.o //libmessage.a 静的ライブラリを message.o を用いて作る。
4. g++ -std=c++11 main.o -L -lmessage -o main //main.o をコンパイルするとき、libmessage をリンクしてコンパイルする。
5. ./main //プログラムを実行する。

解説

学校の linux 環境は-std=c++11 を設定必要がある。なぜかというと、学校の環境上ではデフォルトの C++コンパイラがバージョン 11 ではないからだ。

まず、すべてのソースファイルをコンパイルする。コンパイルしたソースファイルを用いて、静的ライブラリ libmessage.a を message.o に基づいて作る。Ar は archiver の省略である。次に、静的ライブラリを main プログラムコンパイルするときにリンクする。-lmessage は libmessage.a を指す。-lmessage を使わない時、’static_library_name.a’に変えても構わない。最後に、main アプリを実行する。

動作確認

```
[a2433920@xdev05 q1]$ g++ -std=c++11 -c Message.cpp -o message.o
[a2433920@xdev05 q1]$ g++ -std=c++11 -c main.cpp -o main.o
[a2433920@xdev05 q1]$ ar rcs libmessage.a message.o
[a2433920@xdev05 q1]$ g++ main.o -L. -lmessage -o main
[a2433920@xdev05 q1]$ ./main
Number of messages: 2
Hello World.
Hello 2nd one

This is single message constructor

Number of vector messages: 3
1st message
2nd message
3rd message
testing getMessage for index 1 vector : 2nd message
```

Q2

動的リンクライブラリの作成方法は以下の手順で示す。

1. ソースプログラムをオブジェクトファイルにコンパイルする。'.cpp'ファイルを'.o'ファイルにコンパイルする。でも、位置に依存しない形にしないといけない。そのため、'-fPIC'を使わないといけない。
2. 動的リンクライブラリ(シェアライブラリ)を作成する。
3. 動的リンクライブラリを'main.cpp'にリンクする。

コマンドラインは以下の通りになる。

1. g++ -std=c++11 -fPIC -c message.cpp -o message.o　//message.o を作成する。
2. g++ -std=c++11 -shared -o libmessage.so message.o //libmessage.so 動的ライブラリを message.o を用いて作る。
3. g++ -std=c++11 main.cpp -L libmessage.so -o main //main.o をコンパイルするとき、libmessage.so をリンクしてコンパイルする。
4. export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH //動的リンカーが動的ライブラリを探すため、'LD_LIBRARY_PATH'を環境関数に設定しないといけない。
5. ./main //プログラムを実行する。

解説

まず、message.o のオブジェクトを作る。-fPIC(Position-Independent Code)はファイルの場所を独立するファイルのフラグである。次に、動的ライブラリを作る。-shared フラグは動的ライブラリを作るコマンド。main アプリをコンパイルするとき、動的ライブラリをリンクする。動的リンカーが動的ライブラリを探すため、'LD_LIBRARY_PATH'を環境関数に設定しないといけない。最後に、main アプリを実行する。

動作確認

```
[a243392@xdev08 q2]$ g++ -std=c++11 -fPIC -c Message.cpp -o message.o
[a243392@xdev08 q2]$ g++ -std=c++11 -shared -o libmessage.so message.o
[a243392@xdev08 q2]$ g++ -std=c++11 main.cpp -L. libmessage.so -o main
[a243392@xdev08 q2]$ export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
[a243392@xdev08 q2]$ ./main
Number of messages: 2
Hello World.
Hello 2nd one

This is single message constructor

Number of vector messages: 3
1st message
2nd message
3rd message
testing getMessage for index 1 vector : 2nd message
[a243392@xdev08 q2]$ S
```

Q3

静的ライブラリと動的ライブラリの削除時の動作の違いは、これらのライブラリがどのようにリンクされ、実行ファイルによって使用されるかに起因する。

静的ライブラリの場合、静的ライブラリとリンクすると、リンク時に静的ライブラリの内容が実行ファイルにコピーされる。つまり、実行可能ファイルが作成されると、もはや静的ライブラリファイルには依存しなくなる。ライブラリのコードは実行ファイルに直接埋め込まれる。したがって、実行ファイルを作成した後、静的ライブラリを削除しても、実行ファイルは実行される。

一方、動的（共有）ライブラリとリンクする場合、実行ファイルにはライブラリのコードは含まれない。その代わりに、実行時に解決されなければならない共有ライブラリへの参照が含まれる。動的ライブラリは、プログラムの実行時に利用可能である必要がある。動的ライブラリーが削除されると、動的リンカーは必要なライブラリーを見つけロードすることができなくなり、プログラムの実行に失敗する。

動作確認

Q1 のライブラリを消す結果

```
[a2433392@xdev08 q1]$ ./main
Number of messages: 2
Hello World.
Hello 2nd one

This is single message constructor

Number of vector messages: 3
1st message
2nd message
3rd message
testing getMessage for index 1 vector : 2nd message
[a2433392@xdev08 q1]$
```

Q2 のライブラリを消す結果結果

```
[a2433392@xdev08 q2]$ ./main
./main: error while loading shared libraries: libmessage.so: cannot open shared
object file: No such file or directory
[a2433392@xdev08 q2]$
```

自己チェック項目

4 静的リンクライブラリを作成できる.

4 静的リンクライブラリのリンク方法を理解した.

4 動的リンクライブラリを作成できる.

3 動的リンクライブラリのリンク方法を理解した.

4 それぞれのライブラリをリンクしたプログラムの実行時の振る舞いを理解した.