Wydział Fizyki i Informatyki Stosowanej

Programowanie Niskopoziomowe

Konspekt

Biblioteki statyczne i pluginy

Aleksandra Poręba
nr. indeksu 290514

Data seminarium: 8 kwietnia 2019

Data laboratorium: 27 maja 2019

Spis treści

1 Abstrakt

Celem projektu jest zaznajomienie się z koncepcjami bibliotek statycznych oraz wtyczek, oraz praktyczne wykorzystanie uzyskanej wiedzy w zadaniach. Pierwsza część pokrywa wiedzę dotyczącą tworzenia własnych bibliotek statycznych oraz ich analizy. Następne zadania dotyczą tematu pluginów - projektowania i implementacji prostych systemów wtyczek oraz pisania własnych pluginów, do gotowych już programów. Całość jest realizowana w języku C.

2 Tworzenie bibliotek statycznych

2.1 Wymagania

2.1.1 Program ar

Do tworzenia bibliotek statycznych może zostać użyty program `archiver`. Aby stworzyć bibliotekę należy użyć opcji `r` oraz `c`, służących odpowiednio do dodania plików do archiwum oraz stworzenia archiwum.

```
ar rc [nazwa biblioteki] [pliki obiektowe]
```

Nazwa powinna zaczynać się od `lib`, a kończyć rozszerzeniem `.a`. Dodatkową, pomocną komendą może być:

```
nm [nazwa_biblioteki.a]
```

wypisująca zawartość biblioteki wraz z eksportowanymi przez pliki symbolami.

2.1.2 Kompilacja z biblioteką statyczną

Aby skompilować plik za pomocą `gcc` z załączonymi statycznie bibliotekami należy dodać następujące flagi:

```
-L[ścieżka do bibliotek]
-l[nazwa biblioteki]
```

2.1.3 Dodatkowe

Podczas wykonywania zadania należy pamiętać o kolejoności załączania bibliotek. Problem został przedstawiomy na seminarium.

2.2 Treść zadania 1

Zadanie polega na dopisaniu brakującego pliku biblioteki, a potem stworzenie jej, by można było skompilować program za pomocą komendy

```
gcc main.c -L./lib -lfun2 -lfun1
```

bez żadnych errorów/warningów. Plików `main.c` oraz `fun1.c` nie można zmieniać!

3 Tworzenie systemów wtyczek

3.1 Wymagania

3.1.1 Koncepcja systemów wtyczek

Wtyczki są tworzone jak biblioteki dynamiczne. Program powinien przeszukiwać wskazany folder z potencjalnymi wtyczkami, i jeżeli owe znajdzie, załączać je do programu. W przypadku tego zadania, przyjmiemy uproszczoną wersję, pluginy (ich ścieżki) będą zapisane w tablicy podanej "z góry". Każda wtyczka powinna udostępniać funkcje zgodne z zaprojektowanym interfejsem, które będą wywoływane przez program główny.

3.1.2 Biblioteka dlfcn

Do obsługi wtyczek można użyć biblioteki `dlfcn`.

- `dlopen` - używamy do otwarcia pluginu, jako argument podajemy nazwę pliku wraz ze ścieżką oraz `RTLD_NOW` oznaczającym rozwiązanie wszystkich niezdefiniowanych symboli,
  np. `dlopen("./plugins/p1.so", RTLD_NOW);`

- `dlsym` - pobiera wskaźnik do funkcji o podanej nazwie, jako argumenty przyjmuje "handler"do biblioteki, zwrócony przez `dlopen` oraz nazwę funkcji
  np `dlsym(handle, "process");`
- `dlclose` - zamyka "handler"do biblioteki

3.2 Treść zadania 2

Zadaniem jest zaprojektowanie prostego systemu wtyczek i zaimplementowanie go. Zakładamy że ich nazwy (ścieżki) podane są w tablicy, i znajdują się w folderze o tej samej nazwie.
Program powinien przyjmować od użytkownika ciąg znaków (jako argument wywołania lub przez sprintf). Wtyczki powinny być dwie i realizować różne działania na owym stringu, a następnie wypisywać je na ekran. Przykładowe działania to: zamiana wszystkich liter na wielkie, mieszanie kolejności liter, wypisywanie tylko samogłosek, wypisywanie wyrazu od tyłu etc.

Należy pamiętać o uzupełnieniu `makefile`! Wskazówki w którym miejscu dokonać modyfikacji znajdują sie bezpośrednio w przygotowanym pliku.

4 Tworzenie wtyczek

4.1 Wymagania

4.1.1 Program bazowy i jego interfejs

Program, do którego zadaniem będzie napisać wtyczki, został omówiony dokładnie na seminarium. Do rozwiązania tego zadania musimy jedynie znać interfejs, jaki ma mieć plugin. Składa się na niego funkcja inicjalizująca (rejestrująca) o prototypie `int init_[nazwa] (PluginManager * )` , gdzie `[nazwa]` to nazwa pluginu.

Pozostałe dwie funkcje to funkcja wypisująca plugin jako opcja w menu oraz funkcja realizująca działanie, za które odpowiada plugin. Są zaczepiane do listy przechowywanej przez `plugin_managera` w funkcji inicjalizującej. Zaczepienie realizują funkcje `register_menu_hook` oraz `register_response_hook`

przyjmujące jako argumenty wskaźnik do `plugin_managera` oraz wskaźnik na odpowiednią funkcję menu/operacji.

4.2 Treść zadania 3

Celem zadania jest napisanie wtyczki do programu kalkulatora prezentowanego na zajęciach. Jej budowa musi być zgodna z interfejsem udostępnianym przez aplikację opisanym powyżej. Wtyczka powinna dodać obsługę mnożenia/dzielenia/potęgowania.
Należy pamiętać o uwzględnieniu nowej wtyczki w `makefile`!

5 Analiza biblioteki statycznej

5.1 Wymagania

5.1.1 Program dwarfdump

Do oglądania zawartości biblioteki statycznej możemy użyć programu `dwarfdump`. Pomoże on nam rozszyfrować deklaracje funkcji, które chcemy wykorzystać, a nie mamy plików źródłowych. Same nazwy eksportowanych funkcji możemy zobaczyć za pomocą wspomnianego już narzędzia `nm`:

```
nm --defined-only ./lib/liblib.a
```

Analizując wynik `dwarfdump` będziemy szukać fragmentów:

- `DW_AT_name`, gdzie znajduje się nazwa poszukiwanej funkcji
- `DW_TAG_formal_parameter`, gdzie będą wyszczególnione argumenty
- `DW_AT_type`, gdzie znajduje się typ zwracany

Na przykład: (output `dwarfdump`, który został odpowiednio przycięty)

```
1 < 1><0x00000062>      DW_TAG_base_type
2                         DW_AT_byte_size       0x00000004
3                         DW_AT_encoding        DW_ATE_signed
4                         DW_AT_name            int
```

```
5
6  ... skipped lines ...
7
8  < 1><0x000002d0>     DW_TAG_subprogram
9                         DW_AT_external        yes(1)
10                        DW_AT_name            fun
11                        DW_AT_decl_line       0x0000001c
12                        DW_AT_prototyped      yes(1)
13                        DW_AT_type            <0x00000062>
14                        DW_AT_sibling         <0x000002fe>
15 < 2><0x000002f1>       DW_TAG_formal_parameter
16                          DW_AT_name            i
17                          DW_AT_decl_line       0x0000001c
18                          DW_AT_type            <0x00000062>
```

Poszukujemy prototypu funkcji `fun`. Odnaleźlismy ją po nazwie w rezultacie `dwarfdump` (linia 10). Widzimy że ma jeden argument (linia 15) o nazwie `i` (linia 16) oraz typie `<0x00000062>` (linia 18), tutaj tym samym co typ zwracany funkcji (linia 13). Gdy odnajdziemy ten symbol (linia 1), okazuje się że to typ bazowy `int` (linia 4). Mamy więc gotową deklarację: `int fun(int i);`

5.2 Treść zadania 4

Dana jest biblioteka statyczna, bez plików źródłowych. Należy uzupełnić plik `main.c`, aby wyeliminować ostrzeżenia `-Wimplicit-function-declaration`, a następnie doprowadzić do kompilacji programu. Nie można zmieniać pliku `makefile`.