

Biblioteki statyczne oraz pluginy

Aleksandra Poręba

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
AGH University of Science and Technology

8 kwietnia 2019

Agenda

- » **Biblioteki statyczne**
- » Działanie bibliotek statycznych
- » Tworzenie bibliotek statycznych – Archiver
- » Użycie bibliotek statycznych
- » Analizowanie bibliotek statycznych
- » Wady i zalety

Agenda

- » **Pluginy**
- » Projektowanie - koncept
- » Narzędzia
- » Tworzenie własnych systemów wtyczek
- » Tworzenie pluginów do gotowych aplikacji

Biblioteki statyczne

Czym są biblioteki statyczne?

- » Bibliotekami nazywamy zbiory funkcji, typów danych, etc, dostarczanych w plików, który możemy wykorzystać w naszym programie
- » Biblioteka statyczna jest zbiorem plików obiektowych, które są spakowane do jednego archiwum
- » Powstała biblioteka jest niezależna od plików, z których powstała
- » Dzięki nim unikamy konieczności każdorazowej kompilacji używanego kodu

Działanie

- » Biblioteki statyczne są łączone z programem na stałe w czasie linkowania
- » Jeśli podczas kompilacji linker nie znajdzie definicji danego symbolu, będzie jej szukał w załączonych bibliotekach
- » Odnaleziony kod zostanie dołączony
- » Każdy dołączony plik obiektowy jest czytany tylko raz

Tworzenie i używanie bibliotek statycznych

- » Jednym z narzędzi do tworzenia bibliotek statycznych jest program `Archiever`
- » Opcje `c` – tworzenia biblioteki, `r` – dodawanie plików

```
np. ar rc libbib.a plik1.o plik2.o
```

- » Aby użyć biblioteki statycznej należy dodać flagi:

- `-L` – ze wskazaniem położenia biblioteki
- `-l [NAZWA]` – z nazwą biblioteki

```
np. gcc main.c -L./lib -lbib
```

Tworzenie i używanie bibliotek statycznych

- » Do archiwum dodawana jest tabela symboli – możemy ją odczytać np. na pomocą narzędzia `nm`
- » Tworzenie bibliotek jest odwracalne – pliki obiektowe można wypakować za pomocą `ar x`
- » Przykład 1
- » Gdy dołączamy więcej niż jedną bibliotekę i są one od siebie zależne, należy zwrócić uwagę na kolejność dołączania
- » Przykład 2

Analiza bibliotek statycznych

- » W tabeli symboli znajdują się nazwy funkcji – nie mamy typów zwracanych ani argumentów
- » Jeśli pliki obiektowe zostały stworzone z flagą `-g` lub `-gdwarf` zostały do nich dodane dodatkowe informacje dla debuggera
- » Pozwolą nam one odtworzyć prototypy funkcji

Analiza bibliotek statycznych - DWARF^{GH}

- » Do analizy można użyć programu `dwarfdump`
- » Korzystając z formatu DWARF możemy odnaleźć interesujące nas elementy:
 - `DW_AT_name` – nazwa funkcji
 - `DW_TAG_formal_parameter` – argumenty funkcji
 - `DW_AT_type` – typ zwracany
- » Przykład 3

Wady i zalety bibliotek statycznych

- » Program wynikowy jest przenośny - nie wymaga obecności użytych bibliotek na urządzeniu z którego korzystamy
- » Niedostępny kod źródłowy
- » Pojedynczy plik binarny zamiast wielu
- » Program wynikowy zajmuje więcej pamięci

Pluginy

Czym są pluginy

- » Pluginem, czyli inaczej wtyczką, nazywamy dodatkowe moduły do programu, które rozszerzają jego możliwości
- » Wymagają obecności programu głównego, ale on sam może działać bez ich obecności
- » Dodawanie lub usuwanie wtyczek nie powinno wymagać rekompilacji programu
- » Np. wtyczki w przeglądarkach (Addblock), Wireshark, Wordpress, do edytorów (Notepad++, Eclipse, IntelliJ), Valgrind

Zastosowanie

- » Pluginy są często używane przez programistów do rozszerzania otwartego oprogramowania
- » Dzięki nim możemy łatwo dodawać nowe funkcjonalności, nie zwiększając rozmiaru aplikacji bazowej

Koncepcja systemu

- » Aplikacja udostępnia **interfejs** zarządzający wtyczkami, jeśli takie są
- » Projektując system wspierający wtyczki należy wziąć pod uwagę trzy etapy:
 - Poszukiwanie dostępnych pluginów
 - Rejestracja (inicjalizacja)
 - Wywoływanie poszczególnych funkcji

System pluginów

- » Wtyczki są zazwyczaj implementowane jako biblioteki dynamiczne
- » Jednym ze schematów tworzenia wtyczek w C jest eksportowanie przez plugin wskaźników do odpowiednich funkcji, nazwanych zgodnie z przyjętym podczas planowania założeniem
- » Wskaźniki przyjmuje część odpowiedzialna za zarządzanie wtyczkami i wywołuje funkcje w odpowiednich momentach

Narzędzia

- » Istnieją frameworki ułatwiające tworzenie wtyczek (C-Pluff, Pluga, Boost.DLL)
- » Można też tworzyć własne systemy
- » Biblioteka `dlfcn.h`
- » Służy do dynamicznego linkowania
- » `dlopen()`, `dlsym()`, `dlclose()`
- » linux.pl/man/index.php?command=dlsym
- » Przykład 4 - prosty system plugin

Przykład 5

- » Aplikacja kalkulatora, która posiada operacje stworzone jako pluginy
- » Interfejs dla wtyczek składa się z części poszukującej wtyczki `plugin_discovery` oraz managera `plugin_manager`
- » Zakładamy że każda wtyczka posiada funkcję inicjalizującą `init_[nazwa]`, która rejestruje odpowiednie funkcje w managerze

Przykład 5

- » Funkcja inicjalizująca wywoływana jest podczas odkrywania wtyczki
- » Manager posiada dwie listy z zaczepionymi funkcjami – jedną do wypisywania opcji w menu i drugą do wykonywania działań
- » Dwie wtyczki – `add` oraz `sub`, są tworzone jako biblioteki dynamiczne (`makefile`)

Pisanie wtyczek do programów

- » Na przykładzie valgrinda
- » <http://www.valgrind.org/docs/manual/writing-tools.html>
- » Udostępniona jest instrukcja jak skonfigurować nową wtyczkę
- » Informacja o interfejsie:

A tool must define at least these four functions:

```
pre_clo_init()  
post_clo_init()  
instrument()  
fini()
```

- » Przykład 6

Bibliografia

- » wikipedia.org
- » geeksforgeeks.org
- » linux.pl/man
- » eli.thegreenplace.net
- » hackaday.com/
- » Milan Stevanovic - C and C++ compiling
- » <https://sourceware.org/binutils/docs/binutils/>
- » <https://developer.ibm.com/articles/au-dwarf-debug-format/>

Dziękuję za uwagę