

FORMAT BINARNY ELF

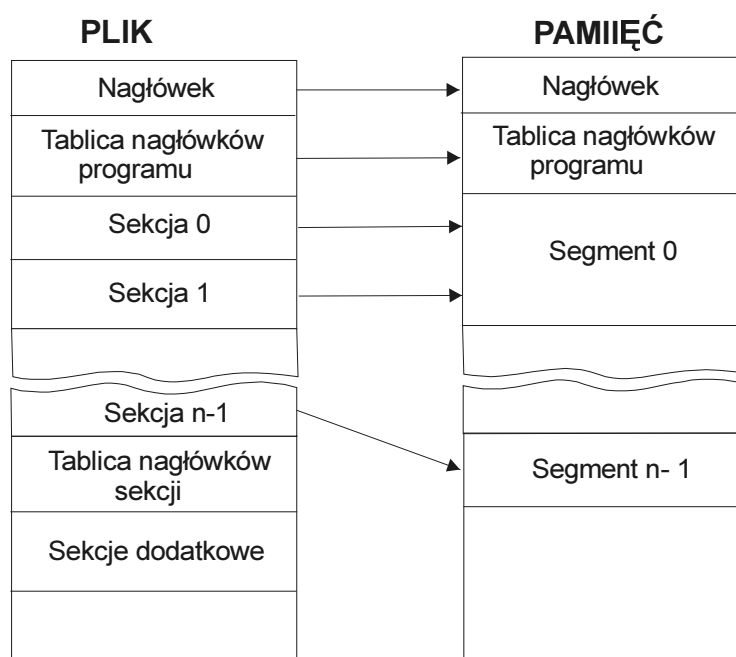
Paweł Pisarczyk

pawel@immos.com.pl

Wydział Elektroniki i Technik Informatycznych,
Politechnika Warszawska

1. Wstęp

Format ELF - **E**xecutable and **L**inkable **F**ormat - zaprojektowano na początku lat dziewięćdziesiątych w laboratoriach USL (*Unix System Laboratories*) jako część binarnego interfejsu aplikacyjnego ABI (*Application Binary Interface*). Niedługo po opracowaniu standardu, ELF stał się formatem powszechnym i chętnie stosowanym. Jego zaletą jest elastyczna budowa i doskonale pomyślana metoda dynamicznej konsolidacji. Podstawowym wymaganiem formatu jest 32-bitowa architektura sprzętowa (istnieje także 64-bitowa wersja formatu), dostarczająca mechanizmu stronicowania, dzięki czemu system operacyjny może załadować program pod z góry ustalone adresy. ELF jest zarówno formatem wykonywalnym, bibliotecznym, jak i obiektowym. Budowa każdej z wersji jest taka sama. Plik ELF składa się z nagłówka i sekcji. Sekcja jest podstawową jednostką formatu i po załadowaniu do pamięci staje się jej segmentem. Budowa pliku ELF i jego obraz wykonywalny przedstawione są na rys. 1.



Rys. 1. Budowa pliku ELF i jego obraz wykonywalny

W pracy zostaną opisane podstawowe struktury danych formatu, sposób ładowania programu do pamięci oraz metoda dynamicznej konsolidacji.

2. Nagłówek formatu ELF

Podstawową rolą nagłówka jest umożliwienie odnalezienia pozostałych elementów pliku. Nagłówek rozpoczyna się identyfikatorem formatu w postaci ciągu `ELF`, poprzedzonego bajtem `0x7F`. Po nim następują informacje dotyczące klasy formatu, sposobu interpretowania bajtów w słowie, wersji, typu pliku (relokowalny, wykonywalny, biblioteka współdzielona, plik „core”) i docelowej architektury (i386, SPARC, Alpha, Mips itp.). Pierwsze pola nagłówka służą do identyfikacji formatu, przeprowadzanej w trakcie uruchamiania i ładowania dowolnego programu przez system operacyjny.

3. Tablica nagłówków sekcji

Jeżeli mamy do czynienia z plikiem obiektywnym tzn. takim, który będzie podlegał przetworzeniu przez konsolidator i stanie się częścią pliku wykonywalnego, to najważniejszym elementem nagłówka jest informacja o położeniu i rozmiarze tablicy nagłówków sekcji (*Section Header Table*). Elementy tabeli nagłówków sekcji pozwalają na zlokalizowanie poszczególnych sekcji w pliku i zawierają podstawowe informacje o każdej z nich takie jak jej typ, nazwa, adres, pod który należy ją załadować i atrybuty. Typ sekcji określa sposób interpretacji zawartej w niej danych - kod, dane, sekcja specjalna, tablica symboli itp. Szczegółowy opis typów można znaleźć w [1]. Dzięki nagłówkom sekcji program konsolidujący może stworzyć w swoich wewnętrznych strukturach danych mapę położenia poszczególnych sekcji, załadować tablice symboli i utworzyć ścieżkę konsolidacji (ang. link path), czyli strukturę danych, która określa w jakiej kolejności należy łączyć ze sobą poszczególne sekcje i przydzielać im adresy wirtualne. Zainteresowanych szczegółami odsyłam do [2]. Tablica nagłówków sekcji znajduje się zawsze na końcu pliku, za sekcjami ładowanymi do pamięci, natomiast przed sekcjami specjalnymi takimi jak tablice łańcuchów, które nie są umieszczane w pamięci w momencie startu procesu.

4. Tablica nagłówków programu

Tablica nagłówków programu jest strukturą, która umożliwia załadowanie programu do pamięci pod ustalone adresy. Położona jest zawsze za nagłówkiem formatu. Dostęp do danych w pliku jest dualny tzn. można odwoływać się do nich przy pomocy tablicy nagłówków sekcji, co realizuje się w przypadku plików obiektywnych, jak również poprzez tablicę nagłówków programu – co ma miejsce w przypadku plików wykonywalnych. Należy jednak pamiętać, że fragment programu wskazywany przez element tablicy nagłówków programu nie musi być taki sam jak sekcja wskazywana przez element tablicy nagłówków sekcji. Elementy tablicy nagłówków programu - nagłówki programu - zawierają komplet informacji potrzebny do załadowania fragmentu programu do pamięci. W nagłówku programu zdefiniowany jest jego typ, rozmiar, adres wirtualny i atrybuty.

5. Sekcje wykorzystywane przy dynamicznej konsolidacji.

Z punktu widzenia ładowania i dynamicznej konsolidacji programu z działającym systemem, oprócz podstawowych sekcji z kodem i danymi, najważniejszą rolę odgrywają sekcje z danymi

kontrolnymi takie jak tablice symboli, tablice relokacji, tablica GOT i PLT. Budowa tych sekcji i ich przeznaczenie zostaną krótko omówione.

5.1. Tablica symboli

Tablica symboli jest strukturą danych używaną w procesie konsolidacji zarówno dynamicznej jak i statycznej. Zawiera informacje potrzebne do zlokalizowania definicji symboli i relokacji odwołań do nich. Struktura opisująca symbol zawiera jego nazwę, wartość, typ i zasięg. Wartością symbolu w większości przypadków jest jego adres lub przesunięcie względem początku sekcji, w której się znajduje. Typ symbolu informuje o tym, czy jest to obiekt pamięci (np. zmienna, tablica), czy funkcja. Każde odwołanie do symbolu spoza bieżącego pliku zaznaczane jest w tablicy symboli jako symbol typu `STT_NOTYPE`. Na przykład odwołanie do standardowej funkcji `printf()` w tablicy symboli programu oznaczone jest symbolem o nazwie `printf` i typie `STT_NOTYPE`. Na tej podstawie konsolidator „wie”, że symbol wymaga połączenia z konkretnym miejscem przestrzeni adresowej procesu, gdzie zostało umieszczone ciało funkcji. Symbol może mieć zasięg lokalny - nie jest widoczny na zewnątrz pliku obiektowego – typ `STB_LOCAL` lub globalny – `STB_GLOBAL`.

5.2 Tablica relokacji

Tablica relokacji odnosi się do konkretnej sekcji i tablicy symboli. Element tablicy zawiera adres, pod którym ma być dokonana relokacja, informację o używanym symbolu i typie relokacji. Relokacja polega na umieszczeniu pod wskazanym adresem odpowiednio obliczonej wartości, wynikającej z wartości symbolu i typu relokacji. Typy relokacji zostały wyczerpująco omówione w [1].

5.3 Tablica GOT i PLT

Tablice GOT (*Global Offset Table*) i PLT (*Program Linking Table*) są podstawowymi strukturami wykorzystywanymi w procesie dynamicznej konsolidacji. Pozwalają na przekierowanie odwołań do funkcji dynamicznych. Wejścia tablicy PLT zawierają sekwencje instrukcji, które służą do przekazywania sterowania, natomiast tablica GOT jest zbiorem adresów wykorzystywanych przez kod tablicy PLT. Współpraca tych struktur zostanie omówiona szczegółowo w dalszej części pracy.

6. Ładowanie programu do pamięci i rozpoczęcie jego wykonywania

Ładowanie programu do pamięci odbywa się w kooperacji z konsolidatorem dynamicznym. W większości implementacji konsolidator znajduje się w pliku `ld.so.x`. Ścieżka do konsolidatora znajduje się w sekcji `SHT_INTERP` i we fragmencie programu `PT_INTERP`. Program ładowany jest do pamięci w oparciu o tablicę nagłówek programu. Konsolidator dynamiczny pełni bardzo ważną rolę w procesie ładowania i wykonania programu. Odpowiada za załadowanie i relokację symboli dynamicznych, zwalniając z tego obowiązku system operacyjny. Konsolidacja dynamiczna może odbywać się od razu po załadowaniu programu do pamięci lub dopiero wtedy, gdy program

odwołuje się do danego symbolu dynamicznego tzw. „lazy binding”. Ze względów wydajnościowych zaleca się drugi ze sposobów. Program uruchamiany jest wtedy dużo szybciej i niekiedy nie jest nawet konieczna konsolidacja np. wtedy, gdy program zakończy się przedterminowo.

7. Konsolidacja dynamiczna

Konsolidacja dynamiczna jest procesem dość skomplikowanym, w trakcie którego wykorzystywane są głównie tablice GOT i PLT, służące do przekierowania odwołań do adresów absolutnych. Prześledźmy proces konsolidacji na przykładzie wywołania funkcji standardowej `printf()`. Na rys. 2 przedstawiony jest odpowiedni fragment tablic PLT i GOT.

Tablica .plt	Tablica .got:
<pre>PLT0: pushl *.GOT1 jmp *.GOT2 nop; nop nop; nop PLT11: jmp *.GOT3_printf pushl \$reloffset1 jmp .PLT0 .PLT12: jmp *.GOT4_fopen pushl \$reloffset2 jmp .PLT0 ...</pre>	<pre>.GOT0: 0x804bf98 .GOT1: [flagi] .GOT2: [adres konsolidatora] .GOT3_printf: .PLT11 + 5 .GOT4_fopen: .PLT12 + 5 ...</pre>

Rys. 2. Przykładowy fragment tablic PLT i GOT.

W momencie wywołania przez proces funkcji `printf` sterowanie przekazywane jest do tablicy PLT, w miejsce określone etykietą `.PLT11`. Znajduje się tam kod, który wykonuje skok pod adres zawarty w odpowiednim wejściu tablicy GOT (etykieta `.GOT3_printf`). Adres ten wskazuje początkowo następną instrukcję w tablicy PLT. W tym przypadku jest to instrukcja `pushl $reloffset1`. Po przekazaniu sterowania z powrotem do tablicy PLT program umieszcza na stosie adres elementu tablicy relokacji, odpowiadającego za relokację wywołania `printf()`. Offset relokacji wskazuje na wejście tablicy GOT oznaczone etykietą `.GOT3_printf`. Po odłożeniu na stosie adresu relokacji, proces przekazuje sterowanie do zerowego wejścia tablicy PLT (etykieta `.PLT0`). Na stosie umieszczane są flagi zawarte w pierwszym wejściu tablicy GOT i wykonywany jest skok pod adres zawarty w drugim wejściu, będący adresem wirtualnym konsolidatora. Konsolidator, po otrzymaniu sterowania, ściąga ze stosu flagi i adres relokacji, odszukuje żądany symbol w jednej z bibliotek, włącza bibliotekę w

przeźren adresową procesu i relokuje wejście tablicy GOT, wpisując w nie adres symbolu. Dzięki temu, przy następnym odwołaniu do funkcji, sterowanie zostanie przekazane bezpośrednio do niej. Ostatnim etapem konsolidacji jest przekazanie sterowania do procesu, w miejsce skąd została wywołana funkcja (na stosie nadal istnieje ślad funkcji). Należy podkreślić, że pierwsze i drugie wejście tablicy GOT, z uwagi na swoje specjalne znaczenie, muszą być odpowiednio ustawione po załadowaniu programu do pamięci.

8. Podsumowanie

Format ELF jest bardzo elastyczny i otwarty. Pliki o tej samej budowie są jednocześnie programami wykonywalnymi, plikami obiektowymi i bibliotekami. Na uwagę zasługuje sposób dynamicznej konsolidacji, który nie wymaga żadnego wstępnego inicjowania bibliotek, ani umieszczania ich pod ustalonymi adresami wirtualnymi.

9 . Literatura

1. "*Executable and Linkable Format. Portable Format Specification, Version 1.1*" Tools Interface Standards (TIS)
2. S. Chamberlain, "*libbfd – The Binary File Descriptor Library*", Free Software Foundation, 1991