Jak program staje się procesem

Program zapisany na dysku nie jest użyteczny sam w sobie. Najpierw musi trafić do pamięci komputera, z której jego instrukcje będą pobierane i wykonywane na przydzielonym procesorze. Wykonywany program wraz ze swoim stanem, tj. przestrzenią adresową, zawartością rejestrów procesora, stosem itd., nazywany jest procesem.

W tym artykule prześledzimy, jak program napisany w języku C staje się procesem. Rozpoczniemy od jego kompilacji, następnie przeanalizujemy format, w jakim jest przechowywany na dysku, a zakończymy, przyglądając się jego ładowaniu przez system operacyjny i uruchomieniu.

PRZYKŁADOWY PROGRAM

Żeby nie wprowadzać niepotrzebnych komplikacji do i tak niełatwego zadania analizy cyklu życia programu, za przykład posłuży prosty program składający się z dwóch plików *main.c* oraz *f.c*, których zawartość zaprezentowano w Listingu 1 oraz Listingu 2.

Listing 1. Plik main.c

```
extern int f(int x);
int main(int argc, char *argv[]) {
    return f();
}
```

Listing 2. Plik f.c

```
#include <stdio.h>
int f(void) {
    return getchar();
}
```

Program po uruchomieniu czeka na pojawienie się dowolnego znaku na standardowym wejściu, po czym kończy swoją pracę, zwracając kod odczytanego znaku lub EOF (End of File) w przypadku błędu lub końca pliku. Program należy skompilować poleceniem z Listingu 3.

Listing 3. Komenda kompilacji programu

```
$ gcc main.c f.c -no-pie -static -o prog1
```

Flaga `-static` spowoduje, że GCC utworzy program statyczny, tj. taki, który zawiera już wszystkie potrzebne funkcje biblioteczne w sobie. Będzie przez to większy, ale proces ładowania do pamięci operacyjnej znacznie się uprości. Na przykład pominięte zostanie linkowanie dynamiczne, czyli program zostanie uruchomiony bez udziału ld.so (`man ld.so`). Flaga `-no-pie` (PIE – Position Independent Executable) spowoduje, że kod wygenerowany przez GCC będzie musiał znaleźć się pod wybranym przez linker adresem w pamięci. Jeśli stałoby się inaczej, to program stosunkowo szybko zakończyłby swoje działanie, ponieważ adresy bezwzględne występujące jawnie w kodzie nie wskazywałyby na poprawne miejsca w pamięci. Innymi słowy, poprawność działania programu jest m.in. determinowana przez jego położenie w pamięci.

Kompilacja programu to w zasadzie skrót myślowy. W istocie proces zamiany postaci tekstowej programu na plik wykonywalny składa się z etapów. Są to preprocessing, kompilacja, asemblacja i linkowanie – zawsze w tym porządku.

Preprocessing to faza, w której działa preprocesor. Zadaniem preprocesora jest przetworzenie kodu źródłowego przed kompilacją. W przypadku `prog1` jego rola sprowadza się do przetworzenia dyrektywy `#include`, czyli wklejenia zawartości jednego pliku w inny. Efekt pracy preprocesora można podejrzeć, wydając polecenie z Listingu 4.

Listing 4. Polecenie spowoduje wyświetlenie efektów pracy preprocesora

```
$ gcc -E main.c f.c
```

Kompilacja polega na zamianie tekstu programu na język asemblera danego procesora. Wysokopoziomowe konstrukcje są zamieniane na instrukcje procesora w formie mnemoników, a utworzony kod jest w dalszym ciągu czytelny dla programisty. Efekt pracy asemblera można podejrzeć, wydając polecenie z Listingu 5, a następnie listując zawartość plików.

Listing 5. Polecenie spowoduje utworzenie plików main.s oraz f.s

```
$ gcc -S main.c f.c
```

Asemblacja to zmiana programu w asemblerze na instrukcje binarne. Wynik asemblacji trafia do tzw. pliku obiektowego. Wydając polecenie z Listingu 6, zostaną utworzone dwa pliki relokowalne (typ pliku obiektowego). Ich zawartość można oglądać przy użyciu narzędzi readelf lub objdump.

Listing 6. Polecenie spowoduje utworzenie plików relokowalnych main.o oraz f.o

```
$ gcc -c main.c f.c
```

Ostatnim etapem jest linkowanie. Zadaniem linkera jest zebranie plików relokowalnych w całość i utworzenie biblioteki współdzielonej bądź pliku wykonywalnego. Tematu bibliotek współdzielonych nie będziemy zgłębiać, a o relokacji będzie mowa w dalszej części artykułu. Efektem pracy linkera jest plik wykonywalny (Listing 3).

FORMAT ELF

ELF jest formatem plików obiektowych, jednym z wielu dostępnych, jednak chyba najbardziej popularnym ze względu na oferowane możliwości. Pliki obiektowe to pliki zawierające oprócz kodu programu dodatkowe dane, użyteczne dla linkera, systemu operacyjnego czy debuggera – zależnie od aktualnych potrzeb.

A czy kodu nie można tak po prostu zacząć wykonywać, bez żadnego przygotowania? Gdyby wszystkie projekty składały się z pojedynczego pliku źródłowego, programy byłyby linkowane wraz ze wszystkimi zależnościami, nie istniałyby biblioteki statyczne ani współdzielone, a systemy operacyjne składałyby się wyłącznie z nie-