Wydział Elektrotechniki, Informatyki i Telekomunikacji

Instytut Informatyki i Elektroniki

Instrukcja do zajęć laboratoryjnych

Język ANSI C (w systemie LINUX)

wersja: 1.32

Nr ćwiczenia:	5	
Temat:	**Tworzenie oraz korzystanie z bibliotek programistycznych**	
Cel ćwiczenia:	Celem ćwiczenia jest przedstawienie zasad tworzenia oraz korzystania z bibliotek statycznych oraz współdzielonych (używana jest też nazwa biblioteki dzielone) w systemie LINUX.	
Wymagane przygotowanie teoretyczne:	Informacje podane na wykładzie. Podstawy pracy z systemem LINUX.	
Sposób zaliczenia:	Sprawozdanie w formie pisemnej.	[ ]
	Pozytywna ocena ćwiczenia przez prowadzącego pod koniec zajęć.	[X]

1. Konwencje przyjęte w instrukcji

`Czcionka o stałej szerokości`

Nazwy programów, poleceń, katalogów, wyniki działania wydawanych poleceń.

`Czcionka o stałej szerokości pogrubiona`

Podaje tekst, który należy dosłownie przepisać. W przypadku plików źródłowych wyróżnia istotniejsze fragmenty.

`Czcionka o stałej szerokości kursywą`

Tekst komentarza w przykładowych sesjach przy terminalu.

`Czcionka o stałej szerokości kursywą pogrubiona`

Wyróżnia istotniejsze fragmenty wyników działania wydawanych poleceń.

2. Uwagi wstępne

Biblioteki programistyczne (ang. *program library*) są plikami zawierającymi skompilowany kod (a czasami też i dane), który może być następnie wykorzystany przez inne programy. Dzięki takiemu rozwiązaniu tworzone oprogramowanie staje się bardziej modularne, łatwiej jest wprowadzać do niego zmiany i poprawki.

Dobrym przykładem bibliotek są standardowe biblioteki samego języka C. Zwierają one bardzo dużą liczbę różnych funkcji, jak przykładowo `printf` oraz `getchar`, które używane są w naszych programach bardzo często i oczywiście pisanie ich za każdym razem gdy tworzymy jakiś program byłoby męczące i nieefektywne. Oprócz oczywistych zalet używania bibliotek standardowych, programista ma możliwość tworzenia własnych bibliotek i tylko od niego samego zależy jakie funkcje w nich zawrze i jak później wykorzysta te biblioteki w praktyce.

W systemie LINUX programista ma możliwość tworzenia trzech rodzajów bibliotek:

- statycznych (ang. *static libraries*),
- współdzielonych (ang. *shared libraries*)
- bibliotek ładowanych dynamicznie (ang. *dynamically loaded libraries*).

Dwie pierwsze są omówione dokładnie, trzecia tylko pobieżnie. Zainteresowane osoby mogą samodzielnie pogłębić wiadomości zapoznając się z literaturą.

3. Biblioteki statyczne

3.1. Uwagi wstępne oraz prosty przykład

Biblioteki statyczne (jak można łatwo domyślić się z ich nazwy) są dołączane do programów w czasie ich kompilacji, a nie dynamicznie w czasie działania programu. Można powiedzieć, że biblioteki statyczne są przeciwieństwem znanych z systemu Windows bibliotek DLL.

Aby użyć kodu z biblioteki statycznej należy wstawić do kodu źródłowego naszego programu plik nagłówkowy biblioteki statycznej i w czasie kompilacji dokonać połączenia kodu aplikacji z kodem biblioteki.

Biblioteka statyczna jest niczym innym jak plikiem binarnym zawierającym jeden lub też wiele plików obiektowych (czyli tych z rozszerzeniem `.o`). Tworzymy ją poleceniem `ar`, które służy do tworzenia, modyfikacji oraz rozpakowywania archiwów. Najczęściej program ten jest jednak używany do tworzenia bibliotek statycznych. Dokładną składnię polecenia znajdziemy w dokumentacji systemowej (`man ar`). Proszę samodzielnie sprawdzić do czego służą najczęściej używane opcje `-c, -s, -r, -q`. Biblioteki statyczne zazwyczaj mają nazwy z rozszerzeniem `.a` (skrót od 'archiwum').

Poniżej pokazano cztery bardzo proste pliki źródłowe. W pierwszym kroku z plików źródłowych `pl.c` oraz `eng.c` zostanie zbudowana *biblioteka statyczna*, a następnie biblioteka ta zostanie użyta w pliku `pleng.c`.

```
/* pl.c */

#include<stdio.h>

void hello_pl(void) {
  printf("Witaj w swiecie bibliotek !\n");
```

```
}

void linux_pl(void) {
  printf("Witaj w LINUX-ie !\n");
}
```

```
/* eng.c */

#include<stdio.h>

void hello_eng(void) {
  printf("Hello in the library world !\n");
}

void linux_eng(void) {
  printf("Welcome to LINUX !\n");
}
```

```
/* pleng.h */

void hello_pl(void);
void hello_eng(void);
void linux_pl(void);
void linux_eng(void);
```

```
/* pleng.c */

/* wstawiamy kod nagłówkowy naszej biblioteki */
#include "pleng.h"

int main (void) {
  hello_pl();
  hello_eng();
  linux_pl();
  linux_eng();
  return(0);
}
```

```
W katalogu bieżącym mamy obecnie tylko cztery pliki źródłowe.
$ ls -l
-rw-r-----    1 artur    students      163 Oct 28 12:38 eng.c
-rw-r-----    1 artur    students      164 Oct 28 12:36 pl.c
-rw-r--r--    1 artur    students      158 Oct 28 13:13 pleng.c
-rw-r--r--    1 artur    students      101 Oct 28 13:13 pleng.h

Pliki źródłowe kompilujemy do postaci obiektowej .o
$ gcc -c eng.c -o eng.o
$ gcc -c pl.c -o pl.o

Następnie za pomocą programu ar tworzymy archiwum (czyli de facto
bibliotekę statyczną libpleng_static.a).
$ ar rcs libpleng_static.a pl.o eng.o

Rozglądamy się po dysku. Odnajdujemy naszą bibliotekę oraz dwa pliki
obiektowe (w tym momencie, gdy zbudowaliśmy już z nich bibliotekę, nie są
one już potrzebne – możemy je wykasować).
```

Zwróćmy uwagę, że bibliotece nadaliśmy nazwę rozpoczynającą się od przedrostka 'lib'. W świecie UNIX-a (LINUX-a) jest to powszechna praktyka i nie powinniśmy jej łamać.

```
$ ls -l
-rw-r-----    1 artur    students      163 Oct 28 12:38 eng.c
-rw-r--r--    1 artur    students     1000 Oct 28 13:13 eng.o
-rw-r--r--    1 artur    students     2246 Oct 28 13:14 libpleng_static.a
-rw-r-----    1 artur    students      164 Oct 28 12:36 pl.c
-rw-r--r--    1 artur    students     1000 Oct 28 13:14 pl.o
-rw-r--r--    1 artur    students      158 Oct 28 13:13 pleng.c
-rw-r--r--    1 artur    students      101 Oct 28 13:13 pleng.h
```

Kompilujemy nasz plik źródłowy pleng.c i w trakcie kompilacji zlecamy kompilatorowi, aby dołączył plik biblioteczny (opcja -lpleng_static). Opcja -L. wskazuje kompilatorowi, aby szukał biblioteki w katalogu bieżącym. Uwaga: kropka po '-L' jest bardzo ważna i mówi ona, że biblioteki maja być poszukiwane w katalogu bieżącym.

```
$ gcc pleng.c -o pleng -L. -lpleng_static
```

UWAGA:
Gdy używamy przełącznika -l to biblioteka MUSI być zapisana w pliku libpleng_static.a (czyli nie podajemy przedrostka 'lib' oraz rozszerzenia '.a').

Można tez nie używać przełącznika -l i podać pełna nazwę biblioteki, ale zwyczajowo tak się nie robi. Wówczas byłoby następująco:

```
  $ gcc pleng.c -o pleng -L. libpleng_static.a
```

lub też tak (bez przełącznika -l oraz -L):

```
  $ gcc pleng.c -o pleng $(PWD)/libpleng_static.a
```

Na dysku powstaje plik wykonywalny pleng.

```
$ ls -l
-rw-r-----    1 artur    students      163 Oct 28 12:38 eng.c
-rw-r--r--    1 artur    students     1000 Oct 28 13:13 eng.o
-rw-r--r--    1 artur    students     2246 Oct 28 13:14 libpleng_static.a
-rw-r-----    1 artur    students      164 Oct 28 12:36 pl.c
-rw-r--r--    1 artur    students     1000 Oct 28 13:14 pl.o
-rwxr-xr-x    1 artur    students     6752 Oct 28 13:21 pleng
-rw-r--r--    1 artur    students      157 Oct 28 13:20 pleng.c
-rw-r--r--    1 artur    students      101 Oct 28 13:13 pleng.h
```

Wykonujemy program.

```
$ ./pleng
Witaj w swiecie bibliotek !
Hello to the library world !
Witaj w LINUX-ie !
Welcome to LINUX !
$
```

Uwaga. Nie cała zawartość biblioteki standardowej języka C jest automatycznie dołączana przez kompilator. Kompilator automatycznie dołącza jej podstawową część (niejawne użycie przełącznika -lc *powodujące „łączenie się" ze standardową biblioteką języka C* libc.a*). Przykładowo domyślnie (ze względów wydajnościowych) nie jest dołączana biblioteka funkcji matematycznych. Programista musi jawnie dołączyć ją w przypadku, gdy korzysta z zawartych w niej funkcji. Aby poprawnie skompilować poniższy program* należy wydąć polecenie:

```
$ gcc mymath.c -o mymath -lm
```

```
#include <stdio.h>
#include <math.h>
```

```
int main(void) {
  double f1 = 2 * 3.1415926;
  double f2 = 3.1415926;
  printf("%f, %f\n", sin(f1), cos(f2)); }

Gdy o tym zapomnimy, to uzyskamy:

$ gcc mymath.c -o mymath

/tmp/ccq288q8.o: In function `main':
/tmp/ccq288q8.o(.text+0x28): undefined reference to `cos'
/tmp/ccq288q8.o(.text+0x42): undefined reference to `sin'
collect2: ld returned 1 exit status
```

3.2. Program nm

Program `nm` wypisuje wszystkie symbole zakodowane w pliku obiektowym lub binarnym (np. biblioteka `.a`). Pozwala nam w prosty sposób zorientować się, czy dany plik obiektowy lub biblioteka zawiera interesujące nas funkcje. Poniżej użyto polecenia `nm` do pokazania symboli zawartych w utworzonej przez nas wcześniej bibliotece `libpleng_static.a`:

```
$ nm libpleng_static.a

pl.o:
00000000 t gcc2_compiled.
00000000 T hello_pl
00000018 T linux_pl
         U printf

eng.o:
00000000 t gcc2_compiled.
00000000 T hello_eng
00000018 T linux_eng
         U printf

Obejrzyjmy również zawartość wspomnianej wcześniej standardowej biblioteki
języka C libc.a. Z uwagi na jej dużą objętość, ograniczymy się tylko do
wpisów dotyczących funkcji printf

$ nm /usr/lib/libc.a | grep printf

...
printf.o:
00000000 T _IO_printf
00000000 T printf
         U vfprintf
...

Widzimy, że funkcja printf zawarta jest w bibliotece libc.a (musi tak być,
gdyż jest to funkcja z biblioteki standardowej języka C). Domyślamy się
dalej, że twórcy biblioteki zaimplementowali kiedyś funkcję printf() w
pliku printf.c, skompilowali go do postaci obiektowej printf.o i ... dalej
postępowali analogicznie jak my w dzisiejszym ćwiczeniu.
```

Widzimy, że nasza biblioteka powstała z dwóch plików obiektowych (`pl.o` oraz `eng.o`). Na listingu odnajdujemy również zdefiniowane w plikach źródłowych funkcje (`hello_pl`, `hello_eng`, `linux_pl`, `linux_eng`). Litera `T` oznacza, że funkcja jest zdefiniowana w tej bibliotece. Litera `U` oznacza, że funkcja jest używana w bibliotece, jednak nie jest w niej

zdefiniowana (w naszym przykładzie chodzi o funkcję `printf`, która oczywiście jest zdefiniowana w jednej ze standardowych bibliotek kompilatora `gcc`).

4. Biblioteki współdzielone

4.1. Uwagi wstępne

Biblioteki współdzielone mają kilka zasadniczych zalet w porównaniu do bibliotek statycznych. Po pierwsze programy z nich korzystające zajmują mniej miejsca na dysku, gdyż kody biblioteki nie są wkompilowywane w każdy plik binarny. Zamiast tego kod biblioteki jest dynamicznie łączony z programem, który tej biblioteki potrzebuje w trakcie uruchamiania programu. Są one ponadto nieco szybsze w działaniu, gdyż ich kod jest ładowany do pamięci tylko raz, przy pierwszym użyciu biblioteki.

Zasadniczą jednak zaletą bibliotek współdzielonych jest to, że nie stanowią one integralnej i nierozłącznej części aplikacji. Zmiana kodu biblioteki (np. usunięcie zauważonych błędów) nie pociąga za sobą konieczności kompilacji wszystkich programów korzystających z tej biblioteki. Wystarczy tylko przekompilować bibliotekę i wszystkie korzystające z niej programy będą „widziały" wprowadzone zmiany.

4.2. Nazewnictwo bibliotek współdzielonych

Chcąc tworzyć biblioteki współdzielone należy przede wszystkim zrozumieć specyficzne zasady tworzenia nazw tych bibliotek (nie mogą one być tworzone dowolnie!). Pojawia się tutaj tzw. *nazwa so* (*soname* - ***s**hared **o**bject name*). *Soname* posiada:

- przedrostek `lib`,
- nazwę biblioteki,
- znak kropki,
- frazę `so`,
- znak kropki,
- numer wersji.

Uwaga: *nazwa so* zwykle jest dowiązaniem symbolicznym (polecenie `ln`) do nazwy fizycznego pliku dyskowego z kodem biblioteki.

Kolejne elementy, które mogą (ale nie muszą) pojawić się to: znak kropki oraz podrzędny numer wersji (ang. *minor number*), znak kropki oraz numer wydania (ang. *release number*). Numer wersji oraz wydania nie są obowiązkowe. Przykładowo, gdy mamy podane: `libnasza.so.1.0.5` napis ten zależy „rozszyfrować" następująco:

- `libnasza.so` - nazwa biblioteki współdzielonej,
- `libnasza.so.1` - nazwa *so* (*soname*) biblioteki współdzielonej (numer wersji: `1`),
- `libnazsza.so.1.0.5` - nazwa pliku, zawierającego kod biblioteki współdzielonej (podrzędny numer wersji: `0`, numer wydania: `5`).

Zmiana w głównym numerze wersji wskazuje na znaczącą zmianę biblioteki i zwykle takie dwie wersje biblioteki nie są ze sobą zgodne. Z kolei zmiana numeru podrzędnego oraz numeru wydania oznacza, że usuwane są mniejsze lub większe błędy w bibliotece, nie skutkujące jednak utratą kompatybilności z poprzednimi wersjami.

4.3. Jak używać i instalować biblioteki współdzielone

W systemie LINUX za dynamiczne łączenie programów z bibliotekami współdzielonymi odpowiedzialny jest program `ld` (czyli tzw. *linker*). Program używający funkcji zdefiniowanych w bibliotece współdzielonej zna oczywiście tylko nazwę funkcji - nic nie wie natomiast o szczegółach jej implementacji. Te szczegóły są oczywiście zakodowane w bibliotece. Zadaniem programu `ld` jest więc połączenie nazwy funkcji z właściwym kodem, który tą funkcję implementuje.

W systemie LINUX domyślnym miejscem, gdzie poszukiwane są biblioteki współdzielone są katalogi `/usr/lib` oraz `/lib`. Jeżeli chcemy, aby linker poszukiwał bibliotek również w innych katalogach musimy je wpisać do zmiennej systemowej `$LD_LIBRARY_PATH`, która zawiera oddzielone przecinkami nazwy katalogów, które program `ld` będzie przeszukiwać w czasie działania programu, oprócz domyślnych katalogów `/usr/lib` oraz `/lib`.

Istnieje też druga zmienna `$LD_PRELOAD`, która jest oddzieloną spacjami listą dodatkowych, określonych przez użytkownika, bibliotek współdzielonych, które są ładowane przed wszystkimi innymi bibliotekami (czasami taka możliwość przydaje się w praktyce).

W systemie LINUX istnieje również plik `/etc/ld.so.conf`, który zawiera listę katalogów, które przeszukuje linker w poszukiwaniu bibliotek współdzielonych wymaganych przez nasz program. Mając uprawnienia administratora możemy dopisać tam nowe katalogi. Wówczas nie ma potrzeby wpisywania ich do zmiennej `$LD_LIBRARY_PATH`.

Kolejnym ważnym plikiem jest `/etc/ld.so.cache`. W pliku tym przechowywane są wszystkie zarejestrowane nazwy *so*. Aby uaktualnić zawartość tego pliku używamy programu `ldconfig` (aby go użyć musimy posiadać uprawnienia administratora), który tworzy *dowiązanie symboliczne* od właściwej nazwy pliku z biblioteką (np. `libnasz.so.1.0.5`) do nazwy *so* (czyli `libnasz.so.1`) i składuje te informacje w pliku `/etc/ld.so.cache`. Gdy program `ldconfig` wywołamy bez żadnych parametrów przeglądane są wszystkie katalogi zapisane w `/etc/ld.so.conf`.

W czasie swojego działania program `ld` przegląda ten plik, odnajduje wymaganą nazwę *so* i ponieważ jest to dowiązanie symboliczne, ładuje odpowiednia bibliotekę do pamięci i wywołania funkcji dowiązuje do odpowiednich symboli w załadowanej bibliotece. Należy być w pełni świadomym nazwy *so* biblioteki. One to bowiem, a nie nazwy plików zawierających kody bibliotek, są używane przez program linkera `ld`.

Jeżeli nie posiadamy uprawnień administratora, a mimo wszystko chcemy napisać program, który korzysta z biblioteki ładowanej dynamicznie, musimy posłużyć się zmienną `$LD_LIBRARY_PATH`, do której wpiszemy katalog, gdzie program linkera (`ld`) będzie mógł odnaleźć naszą bibliotekę, np. tak: `export LD_LIBRARY_PATY=$(PWD)` - każemy wówczas szukać bibliotek w katalogu bieżącym (ale można wskazać oczywiście każdy inny katalog).

4.4. Program ldd

Polecenie `ldd` przydaje się w sytuacji, gdy chcemy wiedzieć jakich bibliotek dynamicznych wymaga do prawidłowego działania pewien program wykonywalny. Wypisuje on po prostu nazwy tych bibliotek. Przykład użycia - patrz kolejny podpunkt.

4.5. Przykład tworzenia i używania biblioteki współdzielonej

Chcąc utworzyć bibliotekę współdzieloną należy wykonać następujące czynności:

- dla zadanych plików źródłowych utworzyć pliki obiektowe. W czasie kompilacji do postaci obiektowej należy używać opcji `-fPIC`, powodującej wygenerowanie tzw. kodu PIC (ang. *Position Independent Code* - kod niezależny od położenia)[1]. Kod ten jest wymagany do późniejszego prawidłowego utworzenia biblioteki współdzielonej,
- tak przygotowane pliki obiektowe użyć do utworzenia biblioteki współdzielonej. W tym celu używamy opcji `-Wl`, `-shared` oraz `-soname` kompilatora `gcc`.

Aby użyć tak przygotowaną bibliotekę współdzieloną należy program z niej korzystający skompilować z opcją `-l` (analogicznie jak przy kompilacji z użyciem bibliotek statycznych).

Poniżej, bazując na plikach źródłowych zamieszczonych w rozdziale 2, pokazano przykład tworzenia oraz wykorzystywania biblioteki współdzielonej. Najpierw budowane są dwa pliki obiektowe, następnie na bazie tych plików tworzona jest biblioteka współdzielona i w końcu tworzone są DWA dowiązania symboliczne od pełnej nazwy pliku biblioteki do:

- nazwy *so,*
- nazwy współdzielonej biblioteki.

Po utworzeniu biblioteki współdzielonej jest ona użyta w programie testowym `pleng.c`.

```
W katalogu bieżącym mamy obecnie tylko cztery pliki źródłowe.
$ ls -l
-rw-r-----    1 artur    students      163 Oct 28 12:38 eng.c
-rw-r-----    1 artur    students      164 Oct 28 12:36 pl.c
-rw-r--r--    1 artur    students      158 Oct 28 13:13 pleng.c
-rw-r--r--    1 artur    students      101 Oct 28 13:13 pleng.h

Pliki źródłowe kompilujemy do postaci obiektowej .o. Ponieważ zamierzamy
utworzyć bibliotekę współdzieloną używamy opcji -fPIC
$ gcc -fPIC -c eng.c -o eng.o
$ gcc -fPIC -c pl.c -o pl.o

Następnie tworzymy bibliotekę współdzieloną. Używamy do tego następującego
formatu:
gcc -shared -Wl,-soname,nazwa_so_biblioteki -o nazwa_pliku
lista_plikow_obiektowych
!!!UWAGA!!!
Poniższy znak ukośnika „\" NIE JEST częścią wpisywanego polecenia. Po
wpisaniu tego znaki i naciśnięciu Enter przechodzimy do nowej linii i
możemy kontynuować wpisywania plecenia. Dopiero kolejne naciśnięcie Enter
(gdy nie jest oczywiście poprzedzone kolejnym znakiem ukośnika) powoduje
wykonanie wpisanego polecenia. Metody tej używany zwykle wtedy, gdy
wpisywane polecenie jest długie i nie mieści się w jednej linii.
$ gcc -shared -Wl,-soname,libpleng_shared.so.1 -o \
  libpleng_shared.so.1.0.0 pl.o eng.o

Biblioteka współdzielona jest w tym momencie utworzona ale jeszcze nie
można z niej korzystać (biblioteka nie jest jeszcze zarejestrowana).
```

[1] *Position Independent Code* oznacza kod niezależny od pozycji. Funkcje biblioteki współdzielonej mogą być załadowane pod różne adresy w różnych programach, tak więc kod współdzielonego obiektu nie może być zależny od adresu (albo pozycji), pod który jest załadowany. Nie wpływa to w żaden sposób na naszą pracę przy programowaniu, musimy jedynie pamiętać, aby użyć flagi -fPIC przy kompilowaniu kodu, który zostanie użyty w bibliotece współdzielonej.

```
$ gcc pleng.c -o pleng_shared -L. -lpleng_shared
/usr/bin/ld: cannot find -lpleng_shared
collect2: ld returned 1 exit status
```

Ponieważ nie posiadamy uprawnień root-a nie możemy użyć programu ldconfig do 'zarejestrowania' utworzonej biblioteki w systemie. W katalogu bieżącym tworzymy więc DWA dowiązania symboliczne - jedno do nazwy so oraz jedno do nazwy biblioteki współdzielonej. Program łączący użyje tej drugiej nazwy łącząc nasz program z biblioteką libplend_shared.so używając opcji -lpleng_shared.

Tworzymy pierwsze dowiązanie.

```
$ ln -s libpleng_shared.so.1.0.0 libpleng_shared.so.1
```

Powyższe dowiązanie będzie potrzebne dopiero w momencie uruchamiania programu. Aby jednak go uruchomić, trzeba wcześniej go skompilować. Tak więc mając tylko to jedno dowiązanie kompilacja nie powiedzie się! Opcja -l wymaga bowiem, aby istniał plik o nazwie zaczynającej się od przedrostka 'lib' i mającej rozszerzenie .so (lub dowiązanie symboliczne o takiej nazwie).

```
$ gcc pleng.c -o pleng_shared -L. -lpleng_shared
/usr/bin/ld: cannot find -lpleng_shared
collect2: ld returned 1 exit status
```

Tworzymy więc drugie dowiązanie. Tym razem kompilacja powiodła się.

```
$ ln -s libpleng_shared.so.1.0.0 libpleng_shared.so
$ gcc pleng.c -o pleng_shared -L. -lpleng_shared
```

Rozglądamy się po dysku.

```
$ ls -l
-rw-r-----    1 artur    students      169 Oct 28 13:24 eng.c
-rw-r--r--    1 artur    students     1100 Oct 30 09:43 eng.o
lrwxrwxrwx    1 artur    students       24 Oct 30 10:12 libpleng_shared.so ->
libpleng_shared.so.1.0.0
lrwxrwxrwx    1 artur    students       24 Oct 30 10:09 libpleng_shared.so.1 ->
libpleng_shared.so.1.0.0
-rwxr-xr-x    1 artur    students     5822 Oct 30 09:48 libpleng_shared.so.1.0.0
-rw-r-----    1 artur    students      164 Oct 28 13:24 pl.c
-rw-r--r--    1 artur    students     1092 Oct 30 09:44 pl.o
-rw-r--r--    1 artur    students      158 Oct 28 13:26 pleng.c
-rw-r--r--    1 artur    students      101 Oct 28 13:13 pleng.h
-rwxr-xr-x    1 artur    students     5371 Oct 30 10:12 pleng_shared
```

Próbujemy uruchomić program. Ponieważ domyślnie linker poszukuje bibliotek tylko w katalogach systemowych (zwykle /usr oraz /usr/lib) więc biblioteka nie zostaje odnaleziona. Musimy więc próbować inaczej ...

```
$ ./pleng_shared
./pleng_shared: error while loading shared libraries:
libpleng_shared.so.1: cannot open shared object file: No such file or
directory
```

Tym razem używamy zmiennej środowiskowej LD_LIBRARY_PATH. Program linkera szuka teraz potrzebnej biblioteki najpierw w katalogu (katalogach) określonych przez tą zmienna środowiskową.

```
$ export LD_LIBRARY_PATH="."
$ ./pleng_shared
Witaj w swiecie bibliotek !
Hello in the library world !
Witaj w LINUX-ie !
Welcome to LINUX !
$
```

Alternatywą do nieporęcznego uruchamiania programu jest dodanie ścieżki, gdzie znajduje się biblioteka do pliku /etc/ld.so.conf i uaktualnienie

```
bufora (/etc/ld.so.cache) za pomocą programu ldconfig (trzeba mieć prawa
root-a!)

Inną alternatywą (też potrzebne są uprawnienia root-a!) jest umieszczenie
biblioteki w jednym z katalogów systemowych (zwykle /usr oraz /usr/lib) a
następnie uruchomienie polecenia ldconfig. Zaletą tej metody jest to, że
nie trzeba wówczas specyfikować ścieżki poszukiwania za pomocą
przełącznika -L.

Na koniec zobaczmy jeszcze jak działa polecenie ldd. Widzimy, że do
prawidłowego działania naszego programu wymagane są dwie biblioteki
systemowe oraz nasza biblioteka. Napis 'not found' pojawia się, gdyż
biblioteka nie jest zarejestrowana w systemie.

$ ldd pleng_shared
        libpleng_shared.so.1 => not found
        libc.so.6 => /lib/libc.so.6 (0x40020000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)

Gdy biblioteka zostanie prawidłowo zarejestrowana w systemie za pomocą
ldconfig (aby to zrobić wymagane są uprawnienia administratora) wynik
działania polecenia ldd będzie następujący (nie pojawia się już napis 'not
found').

$ ldd pleng_shared
        libpleng_shared.so.1 => ./libpleng_shared.so.1 (0x40018000)
        libc.so.6 => /lib/libc.so.6 (0x40020000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)

I to by było na tyle!
```

5. Biblioteki ładowane (i wyładowywane) dynamicznie

Biblioteki ładowane dynamicznie to w istocie inny sposób używania, znanych już z poprzednich rozdziałów niniejszej instrukcji, plików obiektowych (z rozszerzeniem `.o`) oraz bibliotek statycznych / współdzielonych (z rozszerzeniem `.a` lub `.so`). Tworzy się je dokładnie tak samo jak pokazano powyżej - nie ma więc żadnej różnicy w ich wewnętrznej (binarnej) strukturze. Różnica polega jedynie na tym, że nie muszą one być ładowane w czasie uruchamiania programu, mogą natomiast był ładowane w dowolnym momencie PO uruchomieniu programy, wtedy gdy są potrzebne. To rozwiązanie nadaje się idealnie do implementacji różnego rodzaju modułów typu *plugin* zwiększających funkcjonalność istniejących już aplikacji.

Oczywiście program wykorzystujący ten mechanizm musi być napisany w taki sposób, aby było możliwe ładowanie tych modułów na żądanie. Zwykle odbywa się to tak, że programista tworzy plik konfiguracyjny, gdzie z kolei użytkownik aplikacji może wskazać czy i jakie moduły mają być ładowane. Niestety kod źródłowy aplikacji staje się przez to bardziej złożony, jednak z drugiej strony zyskujemy znacznie na funkcjonalności i modułowości finalnego programu.

W tym podejściu biblioteki ładowane są wprost za pomocą tzw. interfejsu API o nazwie `dl` (ang. *dynamic loading*). Interfejs ten (sam zaimplementowany jako biblioteka `libdl`) zawiera funkcje do ładowania, przeszukiwania oraz usuwania z pamięci obiektów współdzielonych (bibliotek). Aby użyć tych funkcji należy do kodu źródłowego wstawić nagłówek `<dlfnc.h>` i połączyć się z biblioteką `libdl`. Nie musimy łączyć się z biblioteką, której chcemy użyć. Co więcej, pisząc program, w ogóle nie musimy wiedzieć o istnieniu tej biblioteki, gdyż może ona jeszcze nie być wcale zaimplementowana!

Interfejs `dl` udostępnia cztery funkcje do pracy z obiektami współdzielonymi. Są to: `dlopen`, `dlclose`, `dlsym`, `dlerror`.

Poniżej pokazano kod źródłowy programy, który korzysta z interfejsu `dl`. Zwróćmy uwagę, że nie ma tutaj wstawionego pliku nagłówkowego `pleng.h`. Oznacza to tyle, że w czasie kompilowania programu funkcje zdefiniowane w tym pliku nie muszą być wcale dostępne (bo mogą np. jeszcze nie być zaimplementowane). Ładowanie biblioteki odbywa się „w locie", w trakcie działania programu. Za pomocą funkcji `dlopen` ładowana jest biblioteka `libpleng_shared.so` a za pomocą funkcji `dlsym` odszukiwana jest w tej bibliotece funkcja `hello_pl`, która następnie jest wykonywana. Program ten kompiluje się standardowo. Zwróćmy tylko uwagę, że program łączony jest (opcja `-l`) z biblioteką `libdl`, która implementuje interfejs `dl`.

Więcej szczegółów na temat użycia interfejsu `dl` można znaleźć na przykład w opracowaniu *Program Library HOWTO* (patrz literatura).

```
Kompilacja do postaci obiektowej.
$ gcc -c dl_demo.c

Zwróćmy uwagę na -ldl. Ten interfejs zapewni mam możliwość ładowania 'w
locie' biblioteki libpleng_shared.so.
$ gcc -o dl_demo dl_demo.o -ldl

Wykonujemy program.
$ export LD_LIBRARY_PATH="."
$ ./dl_demo
```

```
/*
dl_demo.c

Przykład zaczerpnięty z opracowania:
  Program Library HOWTO
  David A. Wheeler
  version 1.00, 22 March 2002
  http://www.dwheeler.com/program-library

Zmieniono tylko nazwę wołanej biblioteki (libpleng_shared.so) oraz
nazwę wołanej funkcji (hello_pl).
*/

/* Need dlfcn.h for the routines to dynamically load libraries */
#include <dlfcn.h>
#include <stdlib.h>
#include <stdio.h>

/* Note that we don't have to include "pleng.h".
   However, we do need to specify something related;
   we need to specify a type that will hold the value
   we're going to get from dlsym(). */

/* The type "simple_demo_function" describes a function that
   takes no arguments, and returns no value: */

typedef void (*simple_demo_function)(void);


int main(void) {
```

```
  const char *error;
  void *module;
  simple_demo_function demo_function;

 /* Load dynamically loaded library */
 module = dlopen("libpleng_shared.so", RTLD_LAZY);
 if (!module) {
   fprintf(stderr, "Couldn't open libpleng_shared.so: %s\n", dlerror());
   exit(1);
 }

 /* Get symbol */
 dlerror();
 demo_function = dlsym(module, "hello_pl");
 if (error = dlerror()) {
   fprintf(stderr, "Couldn't find hello_pl: %s\n", error);
   exit(1);
 }

 /* Now call the function in the DL library */
 (*demo_function)();

 /* All done, close things cleanly */
 dlclose(module);
 return 0;
}
```

6. Zadania do samodzielnego wykonania

a) dla przykładowego programu (składającego się z kilku plików źródłowych) podanego w rozdziale 2 instrukcji nr 4 utworzyć plik *makefile*, w którym zdefiniowane będą następujące główne cele (być może będzie wygodnie zdefiniować również inne cele pomocnicze):

- `lib_stat` - tworzenie biblioteki statycznej
- `lib_dyn` - tworzenie biblioteki współdzielonej
- `calc_stat` - kompilacja programu z użyciem biblioteki statycznej
- `calc_dyn` - kompilacja programu z użyciem biblioteki współdzielonej
- `all` - wykonuje automatycznie cztery powyższe cele
- `clean` - usuwanie z katalogu bieżącego wszystkich plików za wyjątkiem plików źródłowych

Wszędzie tam, gdzie jest to możliwe używać zmiennych automatycznych programu `make` (takich jak na przykład: `$@`, `$*`, `$<`, `$^`, `$?`) oraz predefiniowanych zmiennych i nazw programów (takich jak na przykład: `AR`, `CC`, `CFLAGS`, `RM`).

b) utworzyć drugi plik *makefile* analogicznie jak w poprzednim podpunkcie, ale dla przykładów z bieżącej instrukcji.

Zadanie dla PRAWDZIWYCH PROGRAMISTÓW ! (nieobowiązkowe):

c) program `calc.c` podany w rozdziale 2 instrukcji nr 4 przerobić w taki sposób, aby wykorzystywał interfejs `dl`.

Literatura

1. Polecenia systemowe: `man ld(1)`, `man ldd(1)`, `man ra(1)`, `man nm(1)`

2. *Program Library HOWTO* (na przykład pobrany ze strony autora opracowania: http://www.dwheeler.com/program-library)

3. Mark Michell, Jeffrey Oldham, Alex Samuel: LINUX. Programowanie dla zaawansowanych. Wydawnictwo RM, Warszawa 2002