lab_7 - Instrukcja do ćwiczenia

Teoria:

Bliblioteki statyczne i dynamiczne:

Celem ćwiczenia jest zrozumienie zasad łączenia modułów (linkowania) w sposób statyczny i dynamiczny, a także opanowanie reguł dotyczących tworzenia kodu, tak aby mógł być wykorzystany w odpowiedniej bibliotece.

Pliki:

Do realizacji ćwiczenia niezbędne będą następujące pliki:

- lab_7.c - program główny + funkcja (język C)
- gcd_static.s - funkcja (asembler)
- gcd_dynamic.s - funkcja (asembler)
- print_rsp_static.s - funkcje (asembler)
- print_rsp_dynamic.s - funkcje (asembler)

Przydatne też mogą być dwa dodatkowe pliki:

- commands.txt - przykłady poleceń
- links.txt - przydatne linki do uzupełnienia wiedzy

Algorytm GCD (greatest common divisor) w wersji iteracyjnej:

```
unsigned int gcd_c( unsigned int a, unsigned int b )
{
     while( a != b )
     {
          if( a > b )
               a = a - b;
          else
               b = b - a;
     }
     return a;
}
```

Implementacja algorytmu w asemblerze była omówiona w trakcie zajęć lab_5:

```
gcd_a:
#
gcd:
     cmp %edi, %esi      # (a==b)?
     jz computed         # yes
     jb b_below_a        # if(b < a) goto b_below_a
     sub %edi, %esi      # else b=b-a
     jmp gcd
b_below_a:
     sub %esi, %edi      # a=a-b
     jmp gcd
computed:
     mov %edi, %eax      # result (a==b)
#
     ret                 # return gcd
```

Biblioteka statyczna:

Biblioteka statyczna tworzona jest poprzez umieszczenie skompilowanych modułów w archiwum (zbiorze plików tego samego typu). Kompilowane moduły mogą korzystać z danych w dowolny sposób (dowolnego trybu adresowania). Przykładowe polecenia pozwalające na stworzenie biblioteki o nazwie **libstat.a** (standardowo biblioteki zaczynają się od liter **lib**, a biblioteki statyczne mają rozszerzenie **.a**) są następujące:

```
gcc -c -o  gcd_static.o  gcd_static.s
gcc -c -o  print_rsp_static.o   print_rsp_static.s

ar  rcs  libstat.a  gcd_static.o  print_rsp_static.o
```

Dwa pierwsze polecenia dotyczą kompilacji modułów źródłowych, zaś trzecie jest odpowiedzialne za utworzenie biblioteki i umieszczenie w niej skompilowanych modułów.

Biblioteka dynamiczna (dzielona):

Biblioteka dynamiczna tworzona jest poprzez umieszczenie skompilowanych modułów w pliku o odpowiedniej strukturze. Kompilowane moduły muszą odwoływać się do danych w szczególny sposób (pośredni tryb adresowania względem licznika programu/instrukcji **%rip**), tak aby mogły działać niezależnie od adresu miejsca pamięci w którym się znajdą. W trakcie kompilacji konieczne jest użycie opcji **–fPIC** (Position Independent Code). Przykładowe polecenia pozwalające na stworzenie biblioteki o nazwie **libdyn.so** (standardowo biblioteki zaczynają się od liter **lib**, a biblioteki dynamiczne mają rozszerzenie **.so** (Shared Objects) są następujące:

```
gcc -fPIC -c -o  gcd_dynamic.o   gcd_dynamic.s
gcc -fPIC -c -o  print_rsp_dynamic.o  print_rsp_dynamic.s

gcc -shared -o libdyn.so  gcd_dynamic.o  print_rsp_dynamic.o
```

Dwa pierwsze polecenia dotyczą kompilacji modułów źródłowych, zaś trzecie jest odpowiedzialne za utworzenie biblioteki i umieszczenie w niej skompilowanych modułów.

Użycie bibliotek:

- Biblioteka statyczna:

  ```
  gcc  -no-pie  -o  lab_7_static   lab_7.c   -L.  -lstat
  ```

- Biblioteka dynamiczna:

  ```
  gcc -fPIC -o lab_7_dynamic  lab_7.c  -L.  -ldyn
  ```

W powyższych poleceniach wykorzystywany jest ten sam moduł źródłowy **lab_7.c** (program główny i funkcja **gcd_c**). Opcja **–L.** oznacza poszukiwanie bibliotek w bieżącym katalogu, zaś opcja **–l** wskazuje nazwę biblioteki (z pominięciem rozszerzenia i początkowego **lib**).

Jeżeli w trakcie uruchomienia programu korzystającego z biblioteki dynamicznej pojawią się komunikaty świadczące o tym, że nie została ona znaleziona, to należy wtedy przenieść plik biblioteki do katalogu systemowego przeznaczonego na biblioteki – można też zmodyfikować zmienną środowiskową **LD_LIBRARY_PATH**, tak aby zawierała katalog, w którym znajduje się biblioteka.

Praktyka (lab_7.c i pozostałe plik):

Działania:

1. Tworzymy bibliotekę w wersji statycznej:

   ```
   gcc -c -o  gcd_static.o  gcd_static.s
   gcc -c -o  print_rsp_static.o   print_rsp_static.s
   ar  rcs  libstat.a  gcd_static.o  print_rsp_static.o
   ```

2. Budujemy kod wykonywalny wykorzystując stworzoną bibliotekę:

   ```
   gcc  -no-pie  -o  lab_7_static   lab_7.c   -L.  -lstat
   ```

3. Uruchamiamy program podając jednocześnie dwie liczby będące argumentami funkcji **gcd_c** i **gcd_a**:

```
./lab_7_static    3084   1424
```

4. Program powinien zadziałać prawidłowo – obie funkcje powinny zwrócić wartość **4**.
5. Sprawdzamy zawartość pliku wykonywalnego:

```
objdump  -D  lab_7_static   >   lab_7_static.txt
```

6. Otwieramy plik **lab_7_static.txt** w edytorze.
7. Wyszukujemy napis **<.plt>** - zawartość kilku linii tekstu poniżej **section <.plt>** świadczy o tym, że funkcje **atoi** i **printf** są linkowane dynamicznie. Uwaga: jeżeli szukany symbol występuje kilka razy, to należy zlokalizować właściwe miejsce!
8. Wyszukujemy napis **<main>** - możemy zobaczyć kod (w asemblerze) programu głównego (funkcji **main**). Powyżej znajduje się kod funkcji **gcd_c**, poniżej kod funkcji **gcd_a**, **print call_rsp** i **print_ret_rsp** – moduły zostały dołączone w sposób statyczny. W kodzie funkcji **main**, przy odwołaniach do funkcji **atoi** i **printf**, ich nazwy zostały uzupełnione o fragment **@plt** – te funkcje są łączone dynamicznie.
9. Budujemy w pełni statyczny kod wykonywalny:

```
gcc -static -no-pie -o lab_7_static lab_7.c -L. -lstat
```

10. Sprawdzamy jego działanie:

```
./lab_7_full_static    3084   1424
```

11. Program powinien zadziałać identycznie jak poprzedni.
12. Ponownie sprawdzamy zawartość pliku wykonywalnego:

```
objdump -D lab_7_full_static > lab_7_full_static.txt
```

13. Otwieramy plik **lab_7_full_static.txt** w edytorze – plik jest duży, więc chwilę to zajmie.
14. Wyszukujemy napis **<.plt>** - tekst poniżej **section <.plt>** powinien wyglądać inaczej niż wcześniej.
15. Wyszukujemy napis **<main>** - ponownie możemy zobaczyć kod (w asemblerze) programu głównego (funkcji **main**). Powyżej znajduje się kod funkcji **gcd_c**, poniżej kod funkcji **gcd_a**, **print call_rsp** i **print_ret_rsp** – moduły zostały dołączone w sposób statyczny. W pliku można też znaleźć kod funkcji **atoi** oraz **_IO_printf** (wywołanie takiej funkcji pojawia się w kodzie funkcji **main**) – wszystkie funkcje zostały dołączone statycznie.
16. Porównujemy wielkości plików wykonywalnych: **lab_7_static** to ok. **8/17** kB (biblioteka standardowa jest linkowana dynamicznie, nasze moduły statycznie), **lab_7_full_static** to ok. **800** kB (moduły i biblioteka standardowa są linkowane statycznie).
17. Przechodzimy do linkowania dynamicznego.
18. Porównujemy zawartości modułów źródłowych **gcd_static.s** z **gcd_dynamic.s** oraz **print_rsp_static.s** z **print_rsp_dynamic.s** – różnice związane są z dodatkowymi wymogami, jakie muszą być spełnione aby kod mógł być wykorzystany podczas linkowania dynamicznego (nazwy wywoływanych funkcji muszą być uzupełnione o napis **@plt** oraz wykluczone są odwołania do danych w sposób bezpośredni –

konieczne jest adresowanie danych w sposób pośredni względem zawartości rejestru **%rip**).

19. Tworzymy bibliotekę w wersji dynamicznej:

```
gcc -fPIC -c -o  gcd_dynamic.o   gcd_dynamic.s
gcc -fPIC -c -o  print_rsp_dynamic.o  print_rsp_dynamic.s
gcc -shared -o libdyn.so gcd_dynamic.o print_rsp_dynamic.o
```

20. Budujemy kod wykonywalny wykorzystując stworzoną bibliotekę:

```
gcc -fPIC -o lab_7_dynamic  lab_7.c  -L.  -ldyn
```

21. Sprawdzamy jego działanie:

```
./lab_7_dynamic 3084  1424
```

22. Program powinien zadziałać tak samo jak poprzednie – jeżeli wystąpiły problemy (biblioteka nie została znaleziona), to przeczytaj jeszcze raz część teoretyczną.
23. Sprawdzamy zawartość pliku wykonywalnego (można to zrobić - nawet jeżeli samego programu nie daje się uruchomić):

```
objdump -D lab_7_dynamic  >  lab_7_dynamic.txt
```

24. Otwieramy plik **lab_7_dynamic.txt** w edytorze.
25. Wyszukujemy napis **<.plt>** - w części pliku zaczynającej się od **section <.plt>** powinny pojawić się nazwy wszystkich funkcji wykorzystywanych w programie (poza **main** i **gcd_c**) - świadczy to o tym, że wszystkie są linkowane dynamicznie.
26. Wyszukujemy napis **<main>** - ponownie możemy zobaczyć kod (w asemblerze) programu głównego (funkcji **main**). Powyżej znajduje się kod funkcji **gcd_c**, poniżej nie będzie już kodu funkcji **gcd_a**, **print call_rsp** i **print_ret_rsp** – te moduły zostały dołączone w sposób dynamiczny.
27. Ponownie porównujemy wielkości plików wykonywalnych: **lab_7_dynamic** to ok. **8/16** kB (trochę mniej niż **lab_7_static**) – teraz wszystkie moduły są linkowane dynamicznie, ale kod funkcji **gcd_a**, **print_call_rsp** i **print_ret_rsp** jest na tyle mały, że oszczędności w wielkości programu są niewielkie.
28. Świętujemy kolejny sukces!