piątek/tydzień nieparzysty/10:30 Wrocław, dn. 15. marca 2012

Artur Zochniak, nr albumu 184725

Łączenie kodu programu napisanego w języku c/c++ z kodem assemblera na platformie Linux/x86

sprawozdanie z laboratorium kursu „Architektura komputerów 2”

rok akademicki 2011/2012, kierunek INF

Prowadzący:

prof. dr hab. inż. Janusz Biernat

Spis treści

Cele laboratorium

Celem laboratorium było:

- napisanie programu w assemblerze i odwołanie się w nim do funkcji napisanej w C/C++,
- napisanie programu w C/C++ i użycie weń funkcji napisanej w assemblerze,
- napisanie programu w assemblerze i odwołanie się w nim do funkcji zdefiniowanej w standardzie C/C++.

Wstęp

Kompilacja to proces zamiany kodu czytelnego dla człowieka na kod czytelny dla komputera. Zarówno programy napisane w języku C lub C++, jak i programy napisane w assemblerze podlegają kompilacji, która umożliwia późniejsze linkowanie. Wszystko to po to, aby program mógł zostać uruchomiony na tej samej maszynie.
Musi zatem istnieć sposób, aby połączyć kody programów napisanych, fragmentami, w dwóch językach. Na tworzenie programu składają się: kompilacja i linkowanie.

Łączenie plików wynikowych

W wyniku kompilacji, np. poleceniem `gcc` lub `c++` czy `as`, powstają pliki wynikowe. Wynikiem kompilacji jest plik zawierający kod programu, lecz w formie przystępnej dla komputera. Rezultatem kompilacji jest plik kodu maszynowego oraz zbioru powiązań, które

są wymagane, aby program działał prawidłowo. Połączenie programu napisanego w języku C/C++ z programem napisanym w assemblerze jest możliwe po otrzymaniu plików wynikowych. Następny etap – linkowanie – łączy oba programy w spójną całość, dołączając również wymagane statyczne importy.

Wywołanie w asm funkcji napisanej w C++

Kompilator C/C++ zajmuje się zarządzaniem stosem (tj. jeśli zostaje zdeklarowana zmienna lokalna, to programista nie musi się martwić o to, aby zmniejszać wartość wskaźnika stosu `%esp`, lecz pracuje na wyższym poziomie abstrakcji, mając dzięki temu więcej czasu na tworzenie algorytmu.
Jeśli jednak te operacje są automatyzowane, to musi istnieć usestymatyzowany sposób przekazywania argumentów do funkcji.
Funkcja skompilowana w C++ oczekuje, że argumenty będą umieszczone na stosie począwszy od argumentu o największym indeksie kończąc na pierwszym.

Plik defineAdd.cpp

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

int add(int a, int b)
{
     printf("wynik=%d+%d=%d\n",a,b,a+b);
     return a+b;
}
```

Plik callAdd.asm

W wywołaniu funkcji w assemblerze argumenty wędrują na stos i wywoływana jest funkcja. Funkcja dodawania sama w sobie wyświetla wynik.

```
.section .data

.globl main
.extern add
main:
     push $2
     push $3
     call add
     mov $1, %eax
     mov $0, %ebx
     int $0x80
```

Program można skompilować poleceniem `Make prog` po uprzednim stworzeniu pliku `Makefile` zawierającego:

```
asm.o:
```

```
	as -o asm.o callAdd.asm --32
cpp.o:
	gcc -m32 -o cpp.o defineAdd.c -c
prog: asm.o cpp.o
	gcc -m32 -o prog cpp.o asm.o
run:
	./prog
clean:
	rm *.o && rm ./prog
```

Po uruchomieniu programu można zaobserwować przebieg obsługiwania argumentów przez funkcji - pierwszy argument to ten, który został umieszczony na stosie jako ostatni.

```
s184725@lak:~/03lab/call_my_cpp_function_from_asm_code$ ./prog
wynik=3+2=5
```

Wywołanie w C++ funkcji napisanej w asm

Każda funkcja w języku C/C++ musi zostać (co najmniej) zadeklarowana. Korzystając z funkcji zdefiniowanej w assemblerze z pomocą przychodzi słowo kluczowe extern. Oznacza ono dla kompilatora tyle, że funkcja (lub zmienna), przy której zostało ono użyte, na pewno będzie już znana w procesie linkowania, jednakże teraz (w procesie kompilacji) jeszcze jej nie znamy. Dodatkowo, każda funkcja w C/C++ musi mieć swój jasno sprecyzowany typ, dlatego niebywale istotne jest, aby funkcja napisana w assemblerze była kompatybilna z deklaracją funkcji w C/C++ (i vice versa).
Funkcja napisana w asm to funkcja dodająca dwie liczby. Przyjmuje ona jako argumenty 2 liczby 32-bitowe w U2 (`int`), a zwraca ich sumę w postaci liczby dodatniej. Deklaracja funkcji w C/C++ wygląda następująco:

```
extern int dodaj_liczby(int a, int b);
```

W momencie implementacji funkcji w asm należy pamiętać, że na stosie będą znajdować się dwie liczby przeznaczone do wykorzystania w tej funkcji, a język C/C++ będzie oczekiwał otrzymania wyniku w rejestrze akumulatora `%eax`.

Plik `cpp.cpp`:

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

extern int dodaj_liczby(int a, int b);
int main(int argc, char *argv[])
{
  int a=2;
  int b=18;
  printf("wynik funkcji zdefiniowaniej w asm wywolanej z
poziomu c=%d\noczekiwany wynik to %d\n"
  ,dodaj_liczby(a,b), a+b);
```

```
    return 0;
}
```

Plik z kodem źródłowym assemblera jedyne co robi to zdejmuje argumenty ze stosu [linie oznaczone przez `(A)`] oraz je dodaje, a wynik przechowuje w rejestrze `%eax` [linia oznaczona w kodzie jako `(B)`].

Plik asm.asm**:**

```
.section .data

.globl dodaj_liczby

.type dodaj_liczby @function
dodaj_liczby:
    push %ebp
    mov %esp, %ebp
    mov 8(%esp),%eax (A)
    mov 12(%esp),%ecx (A)
    add %ecx,%eax #(B)
    mov %ebp,%esp
    pop %ebp
    ret
```

Program można skompilować poleceniem

`Make prog`

Po uprzednim stworzeniu pliku Makefile zawierającym:

```
prog: asm.o cpp.o
        gcc -m32 -g -o prog cpp.o asm.o
cpp.o:
        gcc -m32 -g -x c -o cpp.o call.c -c
asm.o:
        gcc -m32 -g -x assembler -o asm.o asm.asm -c
clean:
        rm *.o && rm ./prog
```

Wynik działania programu wygląda następująco:

```
s184725@lak:~/03lab/call_asm_from_c_32bit$ ./prog
wynik funkcji zdefiniowaniej w asm wywolanej z poziomu c=20
oczekiwany wynik to 20
```

Wywołanie w asm funkcji zdefiniowanej jako standardowa w C/C++

Linker `g++` wykorzystywany również do kompilowania i linkowania doskonale daje sobie radę z odnalezieniem definicji funkcji standardowych.

Celem programu będzie uruchomienie funkcji `printf` wyświetlającej liczbę podaną przez użytkownika (dzięki funkcji `scanf`).

```
.section .data
read:     .ascii "%u\0"
write:    .ascii "%u\n\0"

.global main
.extern printf
.extern scanf
main:
     pushl %ebp
     movl %esp, %ebp
     pushl $n #ostatni argument funkcji (w postaci adresu)
     pushl $read #podanie pierwszego argumentu funkcji
before:
     call scanf
done:
     pushl n #ostatni argument printf (w postaci wartości)
     pushl $write #ciąg znaków formatujący wywołanie printf
     call printf
     movl %ebp, %esp
     popl %ebp
     mov $0, %eax
ret
```

Między etykietami before i done wywoływana jest funkcja scanf, której wywołanie jest równoznaczne z wywołaniem zapisanym w języku C/C++ jako: `scanf((char*)read, &n);`. Argumenty w assemblerze, zgodnie z konwencją. Następnie wywołujemy funkcję printf w sposób, który w C/C++ zostałby zapisany jako: `printf((const char*)write, n);`.

`([const ]char*)` informuje w tym przypadku tylko o tym, że oczekiwany argument funkcji to wskaźnik na ciąg znaków.

Program można skompilować poleceniem `make prog` po uprzednim utworzeniu pliku `Makefile` zawierającego

```
asm.o:
     gcc -g -m32 -x assembler -o asm.o call.asm -c
#    as -o asm.o call.asm --32 #alternatywne rozwiazanie
prog: asm.o
     gcc -g -m32 -o prog asm.o
run: prog
     ./prog
clean:
     rm *.o && rm ./prog
debug: prog
     gdb ./prog
```

Podsumowanie

Łączenie kodu napisanego w C/C++ oraz assemblerze jest możliwe dzięki dwuetapowym tworzeniu programu: kompilacja i linkowanie. Kompilacja pozwala na sprowadzenie obu fragmentów kodu do „wspólnego mianownika”, a linkowanie umożliwia połączenie tych części w jedną całość, dodając przy tym wszystkie potrzebne powiązania (statycznie).

Bibliografia

1) http://pl.wikibooks.org/wiki/Asembler_x86
2) Programming from the Ground Up
   http://download.savannah.gnu.org/releases/pgubook/