Samouczek 2

Gnu Debugger (GDB) i asembler
ciąg dalszy

Prerekwizyty

Samouczek 1

Cele

Ciąg dalszy podstaw posługiwania się debuggerem i powtórzenie podstawowych instrukcji asemblera x86-64. Tłumaczenie kodu języka C na asembler. Stos i jego zastosowania, ułożenie danych w pamięci.

Forma samouczka[1]

Większość ćwiczeń polega na uruchomieniu w debuggerze gdb specjalnie przygotowanych programów i wykonywanie wypisywanych przez nie poleceń (deasemblację funkcji, wyświetlanie zawartości rejestrów lub pamięci). Do następnego etapu zadania przechodzi się najczęściej poleceniem `cont` (tzn. kolejne etapy zostały zorganizowane jako kolejne pułapki w kodzie) lub `run` (w skrócie `r`) z odpowiednim parametrem np. `r 1` (`run` oznacza uruchomienie debugowanego programu z zadanym argumentem wiersza poleceń, jeśli program jest w trakcie wykonywania to debugger zapyta, czy zakończyć aktualną instancję programu i uruchomić go z nowym argumentem).

Może przydać się ściąga z gdb dostępna tu. Składnię i semantykę komend debuggera można również poznać za pomocą komendy `help`, np.
`(gdb) help x`
spowoduje wyświetlenie opisu komendy `x`.

Podstawy

Pobierz i wypakuj[2] plik, następnie przejdź do katalogu `lec5`.

Ćwiczenie 1.

Uruchom gdb za pomocą polecenia
`$ gdb act1`

[1] Źródłem ćwiczeń jest kurs Introduction to Computer Systems na uniwersytecie Carnegie Mellon, USA http://www.cs.cmu.edu/afs/cs/academic/class/15213-m18/www/
[2] Np. poleceniem `tar xf nazwa.tar`,

```
(gdb) r 1
```
kontynuuj zgodnie z poleceniami wyświetlanymi na ekranie.

Ćwiczenie 2

Uruchom gdb za pomocą polecenia
```
$ gdb act2
(gdb) r s
```
kontynuuj zgodnie z poleceniami wyświetlanymi na ekranie.
Wskazówka: pierwszych 10 elementów tablicy liczb całkowitych arr wypiszesz poleceniem
```
(gdb) x/10d arr
```

Ćwiczenie 3

Wyświetl zawartość pliku `act3.c`, np. pisząc cat `act3.c` w wierszu poleceń powłoki.
Skompiluj ten plik do kodu asemblerowego za pomocą polecenia
```
$ gcc -Og -S act3.c
```
Powstanie plik `act3.s`. Wyświetl jego zawartość, np pisząc
```
$ cat act3.s
```
Następnie skompiluj act3.c bezpośrednio do kodu maszynowego pisząc
```
$ gcc -Og -c act3.c
```
Powstanie plik `act3.o`. Wykonaj dezasemblację pliku act3.o pisząc
$ `objdump -d act3.o`
Porównaj wydruk z zawartością `act3.s`. Co oznaczają napisy np. `.file`, `.globl`, `.type`, których nie ma w zdezasemblowanym kodzie?

Sterowanie przebiegiem programu

Pobierz i wypakuj plik. Następnie przejdź do katalogu `lec6`.

Ćwiczenie 4

Uruchom gdb za pomocą polecenia
```
$ gdb act4
(gdb) r 5
```
kontynuuj zgodnie z poleceniami wyświetlanymi na ekranie.

Ćwiczenie 5

Skompiluj plik `act5.c` poleceniem

```
$ gcc -c -Og act5.c
```
następnie zdezasembluj `act5.o` poleceniem
```
$ objdump -d act5.o
```
zobacz przykład tłumaczenia pętli `for`, `while` i `do-while` do asemblera.

Stos

Pobierz i wypakuj plik. Następnie przejdź do katalogu `lec7`.

Ćwiczenie 6

Uruchom gdb za pomocą polecenia
```
$ gdb act6
(gdb) r m
```
kontynuuj zgodnie z poleceniami wyświetlanymi na ekranie.

Ćwiczenie 7

Uruchom gdb za pomocą polecenia
```
$ gdb act7
(gdb) r rsp
```
kontynuuj zgodnie z poleceniami wyświetlanymi na ekranie.

Ułożenie danych w pamięci

Pobierz i wypakuj plik. Następnie przejdź do katalogu `lec8`.

Ćwiczenie 8

```
$ gdb act8
(gdb) r integer
```
kontynuuj zgodnie z poleceniami wyświetlanymi na ekranie.