

Blah blah

Kompilacja i linkowanie

Polecenie `as hello.s -o hello.o -g` służy do kompilacji programu.

Polecenie `ld hello.s -o hello` służy do przeprowadzenia procesu linkowania.

Połączenie tych funkcji można uzyskać poprzez użycie kompilatora gcc:

```
gcc -g -o hello hello.s
```

Użycie narzędzia GDB

Uruchomienie programu `hello` za pomocą debuggera GDB:

```
gdb -q hello
```

Po wyświetlaniu zachęty terminala (`gdb`), komendą `r` (run) uruchamiamy program. Uruchomienie GDB i samego programu daje odpowiedź:

```
Reading symbols from ./hello...done.
(gdb) r
Starting program: /home/damian_koper/Documents/GitHub/OiakLab/lab_1/hello
Hello World!
[Inferior 1 (process 7498) exited normally]
```

W celu zatrzymania i podejrzenia stanu wykonywania programy w jego trakcie możemy ustawić breakpoint (pułapkę) komendą :

```
b[reak] wskaźnik|nr linii
```

Następnie po ścieżce wykonywania programu możemy poruszać się używając komend takich jak `step`, `stepi`, `continue` itp. Niżej przedstawiono proces zatrzymania programu `hello`:

```
Reading symbols from ./hello...done.
(gdb) b 12
Breakpoint 1 at 0x8048074: file hello.s, line 12.
(gdb) b 18
Breakpoint 2 at 0x804808a: file hello.s, line 18.
(gdb) r
```

```
Starting program: /home/damian_koper/Documents/GitHub/OiakLab/lab_1/hello

Breakpoint 1, _start () at hello.s:12
12      mov $WRITE, %eax
(gdb) step
13      mov $STDOUT, %ebx
(gdb) continue
Continuing.
Hello World!

Breakpoint 2, _start () at hello.s:18
18      mov $EXIT, %eax
(gdb) continue
Continuing.
[Inferior 1 (process 8487) exited normally]
```

Ustawiono tu dwa breakpointsy w liniach 12 i 13. Program odpowiednio zatrzymał się przed i po wypisaniu tekstu na ekran. Wykonywanie wznowiono poleceniem `continue`.

Komenda `x`

Komenda `x` wyświetla zawartość pamięci odpowiednio ją formatując. Dokumentacja specyfikuje jej użycie następująco:

```
(gdb) x [Address expression]
(gdb) x /[Format] [Address expression]
(gdb) x /[Length][Format] [Address expression]
```

Przykład użycia dla programu `hello` przed wyświetleniem tekstu (13c oznacza wyświetlenie 13 kolejnych wartości sformatowanych jako `char`):

```
(gdb) x/13c &hello
0x8049096:      72 'H'   101 'e'  108 'l'  108 'l'  111 'o'  32 ' '   87 'W'   111
'o'
0x804909e:      114 'r'  108 'l'  100 'd'  33 '!'   10 '\n'
```

Disassemble

Polecenie `disassemble` wyświetla treść skompilowanego programu w postaci adresów, mnemoników. W tym przypadku strzałką zaznaczone jest polecenie, na którym przerwano wykonywanie programu:

```
(gdb) disassemble
Dump of assembler code for function _start:
0x08048074 <+0>:      mov     $0x4,%eax
0x08048079 <+5>:      mov     $0x1,%ebx
0x0804807e <+10>:     mov     $0x8049096,%ecx
```

```
0x08048083 <+15>:    mov    $0xd,%edx
0x08048088 <+20>:    int    $0x80
=> 0x0804808a <+22>:    mov    $0x1,%eax
0x0804808f <+27>:    mov    $0x0,%ebx
0x08048094 <+32>:    int    $0x80
End of assembler dump.
```