Linux Assembly Programlamaya Giriş

Barış Metin
<baris@metin.org>

<baris.metin@frontsite.com.tr>

Konular

- gcc / gas / ld
- Intel ve AT&T söz dizimleri
- gdb
- INT 80H
- C kütüphane fonksiyonları
- Stack Frame
- Örnek sistem çağrıları
- Örnek kütüphane fonksyonları

gcc

- GNU Compiler Collection
- gcc, orkestra şefi
- Assembly yazmak için neden bir C derleyicisi ?
  - **Linux bir C dünyasıdır !**
  - Linux'un çoğu ve işlevsel assembly programları oluşturmak için kullanılabilecek kütüphaneler C ile gcc için yazılmıştır.
  - gcc yalnızca programları derlemez, inşa eder !

gcc: inşa etme işlemi

gcc prog.c -o prog

- kaynak program için C önişlemcisi (preprocessor) çağırılır -- cpp
- ön işlemciden geçirilmiş kaynak program için gcc tarafından assembly komut setleri oluşturulur.
- assembly programı için assembler çağırılarak, object code oluşturulur. -- gas
- oluşturulan ikili kod standart C kütüphanesine bağlayıcı (linker) kullanılarak bağlanır. -- ld

Intel ve AT&T söz dizimleri

- x86 komut setleri için birden fazla söz dizimi (mnemonics) vardır.
- makina komutları saf ikili formatda olmak zorundadır.

MOV BX,AX

movw %ax,%bx

1000100111000011

AT&T söz dizimi

- Daha taşınabilir bir assembler.
- gcc AT&T söz dizimini kullanarak assembly kodu oluşturur
- gas, bu söz diziminden anlar.
- Genellikle C programcıları bu söz dizimleri ile ilgilenmezler.
- Yanız ! gdb de bu sözdizimine uygun olarak çıktılar üretir.

AT&T söz dizimi (özellikler)

- Kaynak ve hedef tanımları Intel'dekinin aksi sıradadır.
- Komutlar ve saklayıcı isimleri küçük harflidir.
- Saklayıcı isimleri % ile ifade edilir.
- Her komut işlediği değerlerin boyutunu bildirecek şekilde isim değiştirir (movb, movw, movl).
- Tanımlı değerler $ ile gösterilir ($32, $hata_str)
- Bellek referansı gösterimi farklıdır.

  -/+disp(%index,%scale), %base

gdb

- gdb C programları ile çalışan üst seviye bir hata ayıklayıcıdır.
- Fakat assembly komut setleri seviyesinde programınızı incelemenizi sağlar.
- Belleği ve saklayıcıların durumlarını incelemenize olanak verir.

gdb kullanımı

```
$ gdb program_dosyasi
(gdb) break 15
Breakpoint 1 at 0x8048548: file readdir.c, line 15.
(gdb) run
....
(gdb) continue
```

info breakpoints	enable/disable breakpoint_numarasi
break fonksiyon_adi	continue breakpoint_numarasi
run argumanlar	show args

gdb kullanımı (devam)

(gdb) info registers

eax	0x0	0
ecx	0x1	1
edx	0x40137798	1075017624
ebx	0x40136f60	1075015520
esp	0xbffff9f0	0xbffff9f0
ebp	0xbffffa08	0xbffffa08
esi	0x400098b8	1073780920
edi	0xbffffa64	-1073743260
eip	0x8048548	0x8048548

.....

.....

info all-registers

gdb kullanımı (devam)

(gdb) print $eip

$1 = (void *) 0x8048548

(gdb) print $eax

$2 = 0

(gdb) print $eip=0x804854c

(gdb) print degisken

$5 = (struct dirent *) 0x80484b8

....

(gdb) print $5

/x	HEX	/d	signed INT
/c	ASCII	/u	untigned INT
/t	Binary	/o	OCTAL

Yalnızca tek karakterlik veya integer değerlik bilgileri gösterebilir string gösterimleri için **x** komutu kullanılmalıdır.

gdb kullanımı (devam)

(gdb) until 2 — 2. brakepoint'e kadar işlet

(gdb) next — bir sonraki C satırını işlet

(gdb) step — bir sonraki C satırını dallanmadan işlet

(gdb) nexti

0x080486c2 48 printf ("Dosya Adi : %s\n", dp->d_name);

(gdb) stepi

0x080486ce 48 printf ("Dosya Adi : %s\n", dp->d_name);

(gdb) stepi

0x08048408 in printf ()

(gdb) stepi

0x40075170 in printf () from /lib/libc.so.6

Yolunuzu Belirleyin

- INT 80H : Çekirdek fonksiyonları çağırımı
  - Çekirdek sürümlerine bağımlı kod.
  - Extra kod (kütüphane) bağımlılığınız olmaz.
- Kütüphane fonksiyonları kullanımı
  - Sürüm bağımlılıklarından kurtulursunuz.
  - Taşınabilir bir kod üretilebilir
  - Programınızın karmaşıklığı yönetilebilir bir boyutlarda tutulabilir

INT 80H

- Çekirdeğin sağladığı sistem çağrılarının kullanımı.
- Çağrı numarası %eax saklayıcısında bulunmalı
- Eğer parametrelerin sayısı <6 ise, sırası ile, %ebx,%ecx,%edx,%esi,%edi
- Parametrelerin sayısı >5 ise, bellek alanına referans yapabilir veya stack kullanabilirsiniz.
- Dönüş değeri %eax'te tutulacaktır.

C Fonksiyon Çağırımları

- Dönüş değeri %eax'te, 32bit'den büyük ise %edx ve %eax'te ortak olarak tutulacaktır.
- Fonksiyonlara parametreler stack kullanılarak gönderilir.
- Parametreler stack'ta ters yönlü olarak bulunur. İlk parametre son gönderilir.

Stack Frame

- C içerisinde stack'ın genel ve geniş bir kullanımı vardır.
- Fonksiyyon parametreleri, yerel değişkenler için sürekli olarak kullanılır.
- C fonksiyonların kullanan herhangi bir assembly programı öncelikle stack frame'ini oluşturmalıdır :

```
pushl %ebp
movl %esp, %ebp
```

- Stack Frame'i yok etmek için :

```
movl %ebp, %esp
```

Sistem Çağrısı kullanımı

```
.include "defines.h"
.data
selam:
        .string "selam asm\n"
.globl  main
main:
        movl    $SYS_write,%eax
        movl    $STDOUT,%ebx
        movl    $selam,%ecx
        movl    $11,%edx
        int     $0x80
        ret
```

Kütüphane Fonksiyonları

```
.global main
.mystring :
        .string "EAX : %x\nEBX : %x\nECX : %x\nEDX : %x\n"
main :
# once bizi cagiran programin bilgilerini stack'a atiyoruz.
        pushl   %ebp                    # stackframe'in baslangici ebp ile gosteriliyor
        movl    %esp, %ebp              # stack frame'imiz ebp ve esp ile tanimlandi
        pushl   %ebx
        pushl   %esi
        pushl   %edi
```

Kütüphane Fonksiyonları

```
# buradan sonra gerekli hazirliklar
yapildi
# artik programimizi yazabiliriz.
# register bilgilerini ekrana yazdir.
# sirasi ile eax, ebx, ecx, edx
        movl    $10, %eax
        pushl   %edx
        pushl   %ecx
        pushl   %ebx
        pushl   %eax
        pushl   $.mystring
        call printf
```

Kütüphane Fonksiyonları

```
# bizi çağıran programın bilgilerini
# register'lara geri yuklemeliyiz
#     popl      %edi
#     popl      %esi
#     popl      %ebx
#     popl      %ebp
      leave
      ret
```

Teşekkürler

Sorular