Linux ile Sembolik Makina Dili Kullanımı

Yazan:
Phillip
<phillip (at) ussrback.com>

Çeviren:
Fehmi Noyan İSİ
<fnoyanisi (at) yahoo.com>

8 Ocak 2001

Özet

Bu belgede Linux altında sembolik makina dili kullanımını anlatılacaktır. Intel ve AT&T sözdizimi arasında bir karşılaştırma, sistem çağrılarını kullanım yolları ve GCC ile satıriçi sembolik makina diline giriş bilgileri bu belgeye dahil edilen konulardır.

Bu belgenin özgün sürümünü `http://www.linuxassembly.org/articles/linasm.html` adresinde bulabilirsiniz.

Konu Başlıkları

Yasal Açıklamalar

1. Giriş

Bu belge (özellikle satıriçi sembolik makina dili konusunda) programlama alanındaki eksikliklerden dolayı hazırlanmıştır ve şunu belirtmeliyim ki bu belge size bir çekirdek kodu yazma becerisi kazandırmak için yazılmamıştır, çünkü bu konuda zaten yeteri kadar belge mevcuttur.

Bu belgenin çeşitli kısımlarını deneyimlerimden yararlanarak yazdım ve bu sebepten herhangi bir hata bulunması olasıdır. Eğer herhangi bir bölümde böyle bir hataya rastlarsanız bana e–posta yolu bildirmek ve konu hakkında beni bilgilendirmek için tereddüt etmeyin.

Bu belgeden yaralanabilmek için temel bir x86 sembolik makina dili ve C bilgisi gereklidir.

2. Intel ve AT&T Sözdizimi

Intel ve AT&T Sembolik Makina Dili sözdizimleri görünüşte birbirlerinden çok farklıdırlar ve bu durum ilk başta Intel sözdizimi kullanıp daha sonra AT&T sözdizimi ile karşılaşan ya da önce AT&T daha sonra Intel sözdizimi kullanan birisi için karışıklık yaratmaktadır. Sözü fazla uzatmadan temellere başlayalım.

2.1. Önekler

Intel sözdiziminde yazmaçların ve sabit değerlerin önekleri yoktur. Buna karşın AT&T sözdiziminde yazmaçlar bir '%' ve sabit değerler de bir '$' öneki alırlar. Intel sözdiziminde hexadecimal (onaltılık) veya binary (ikilik) sistemdeki sabit değerler sırasıyla 'h' ve 'b' öneki alırlar. Eğer ilk hexadecimal basamak bir harf ise değer bir '0' öneki alır. Örnek:

```
  Intel Sözdizimi          AT&T Sözdizimi
------------------       --------------------
  mov eax,1                movl $1,%eax
  mov ebx,0ffh             movl $0xff,%ebx
  int 80h                  int  $0x80
```

2.2. Terimlerin Yönü

Intel sözdizimindeki terimlerin yönü AT&T sözdizimindekinin tersidir. Intel sözdiziminde ilk terim hedef (destination) ve ikinci terim kaynak (source) olur, buna karşın AT&T sözdiziminde ilk terim kaynak ve ikinci terim hedeftir. Bu durumda AT&T sözdiziminin yararı açıktır. Soldan sağa okuruz, soldan sağa yazarız, öyleyse soldan sağa olan sözdizimi zaten doğal olandır. Örnek:

Intel Sözdizimi	AT&T Sözdizimi
komut *hedef*,*kaynak* mov eax,[ecx]	**komut** *kaynak*,*hedef* movl (%ecx),%eax

2.3. Bellek Terimleri

Yukarıda da görüldüğü gibi bellek terimleri de farklıdır. Intel sözdiziminde temel yazmaç **[** ve **]** karakaterleri arasına yazılırken buna karşın AT&T sözdiziminde **(** ve **)** karakterleri arasına yazılır. Örnek:

Intel Sözdizimi	AT&T Sözdizimi
mov eax,[ebx] mov eax,[ebx+3]	movl (%ebx),%eax movl 3(%ebx),%eax

Karmaşık işlemler gerektiren yönergelerde AT&T sözdiziminin Intel sözdizimine göre anlaşılması daha güçtür. Bu tür işlemler için Intel sözdizimi

segreg:[*base*+*index***scale*+*disp*]

şeklindedir. AT&T sözdizimi ise

%*segreg*:*disp*(*base*,*index*,*scale*)

şeklindedir.

index, *scale*, *disp* ve *segreg* isteğe bağlıdır ve istenirse kullanılmayabilir. *scale* tanımlı değilse ve *index* tanımlı ise, *scale* **1** kabul edilir. *segreg* yönergeye ve uygulamanın gerçek kipte mi yoksa korumalı kipte mi çalıştığına bağlıdır. Gerçek kipte komuta bağlı olmasına karşın, korumalı kipte kullanımı gerekli değildir. Kullanılan sabit değer, AT&T'de *scale* veya *disp* için kulanıldığında **$** öneki kullanılmaz. Örnek:

Intel Sözdizimi	AT&T Sözdizimi

komut *foo*,*segreg*:[*base*+*index***scale*+*disp*]
mov eax,[ebx+20h]
add eax,[ebx+ecx*2h]
lea eax,[ebx+ecx]
sub eax,[ebx+ecx*4h-20h]
	komut %*segreg*:*disp*(*base*,*index*,*scale*),*foo*
	movl 0x20(%ebx),%eax
	addl (%ebx,%ecx,0x2),%eax
	leal (%ebx,%ecx),%eax
	subl -0x20(%ebx,%ecx,0x4),%eax

Gördüğünüz gibi AT&T'nin anlaşılması güçtür.

[*base*+*index***scale*+*disp*]

kalıbı,

disp(*base*,*index*,*scale*)

kalıbına göre daha anlamlı bir görünüm sağlar.

2.4. Sonekler

Sizin de fark ettiğiniz gibi AT&T sözdiziminde komutların sonekleri vardır. Bu sonekler kullanılan terimin boyutunu verir. Uzun için (long: 32 bit) **l**, sözcük için (word: 16 bit) **w** ve bayt (byte: 8 bit) için **b** kullanılır. Intel sözdiziminde bellek terimleri için benzer yönergeler kullanılır, örneğin `byte ptr`, `word ptr`, `dword ptr`. Tabii ki `dword`, **long**'a karşılık gelmektedir. Bu işlem C dilindeki tür dönüştürmeye benzemektedir, fakat kullanılan yazmaçların kapasitesi varsayılan tür genişliği olarak düşünüldüğü zaman gerekli olmamaktadır. Örnek:

Intel Sözdizimi	AT&T Sözdizimi

mov al,bl
mov ax,bx
mov eax,ebx
mov eax, dword ptr [ebx]
	movb %bl,%al
	movw %bx,%ax
	movl %ebx,%eax
	movl (%ebx),%eax

BURADAN SONRAKİ TÜM ÖRNEKLER AT&T SÖZDİZİMİNDE OLACAKTIR

3. Sistem Çağrıları

Bu bölümde linux sistem çağrılarının sembolik makina dilinde kullanımını kısaca anlatılacaktır. Sistem çağrıları, `/usr/man/man2` dizini altında bulunan kılavuz dosyalarındaki bütün işlevleri içerir. Ayrıca `/usr/include/sys/syscall.h` başlık dosyasında da listelenmişlerdir. Bu işlevlerin büyük bir listesi `http://www.linuxassembly.org/syscall.html` adresinde bulunabilir. Bu işlevler Linux **int $0x80** ile de çalıştırılabilir.

3.1. Altıdan Az Argüman Alan Sistem Çağrıları

Tüm sistem çağrıları için sistem çağrı numarası **%eax** içine atılır. Altıdan az argüman alan sistem çağrıları için argümanlar sırasıyla **%ebx**, **%ecx**, **%edx**, **%esi**, **%edi** yazmaçlarına atılır. Sistem çağrısından dönen değer **%eax** içerisinde saklanır.

Sistem çağrı numarasına `/usr/include/sys/syscall.h` başlık dosyasından bakılabilir. Macrolar **SYS_*sistem_çağrısının_adı*** şeklinde ifade edilir , örneğin `SYS_exit`, `SYS_close`, vb.

Örnek 1. Hello world

```
/* write.s */

.data
  hello:
  .string "hello world\n"
.text
  .globl _start
_start:
  movl $SYS_write,%eax  // SYS_write = 4
  movl $STDOUT,%ebx     // fd = fileno(stdio)
  movl $hello,%ecx      // buf = str
  movl $12,%edx         // count = 0x6
  int  $0x80

  movl $SYS_exit
```

write(2) man sayfasına göre, **write** işlevi

```
ssize_t write(int fd, const void *buf, size_t count)
```

şeklinde belirtilir.

Bundan dolayı *fd* **%ebx** içerisine, *buf* **%ecx** çerisine, *buf* **%edx** içerisine ve `SYS_write` da **%eax** içerisine atılır. Bu işlem sistem çağrısının çalıştırılması için bir `int $0x80` yönergesi tarafından takip edilir. Sistem çağrısından geri dönen değer **%eax** içerisinde saklanır.

Aynı işlem beş argümandan az argümanı bulunan sistem çağrılarına uygulanabilir. Sadece kullanılmayan yazmaçlara dokunmayın. **open** veya **fcntl** gibi isteğe bağlı fazladan argüman bulunduran sistem çağrıları kendileri atama yaparlar.

3.2. Beşten Fazla Argüman Alan Sistem Çağrıları

Argüman sayısı beşten fazla olan sistem çağrılarında da yine sistem çağrı numarası **%eax** içerisine saklanır fakat argümalar bellekte tutulur ve ilk argümanı gösteren bir gösterici **%ebx**'e atılır.

Eğer yığıtı kullanıyorsanız parametreler yığıtın içerisine geriye doğru **push** edilmelidir, örneğin son parametreden ilkine doğru. Daha sonra yığıt gösterici **%ebx** taban yazmacı içerisine kopyalanmalıdır. Aksi taktirde parametreleri ayrılmış bir bellek alanına kopyalayın ve ilk parametrenin adresini **%ebx** içerisine koyun.

Örnek 2. sistem çağrısı mmap

```
/* mmap.s */

.data
  fd:
    .long 0
  fdlen:
    .long 0
```

```
  mappedptr:
    .long 0

.text
.globl _start
_start:
  subl $24,%esp

// open(file, O_RDONLY);
  movl $SYS_open,%eax
  movl 32(%esp),%ebx     // argv[1], %esp+8+24 de
  xorl %ecx,%ecx         // %ecx'i O_RDONLY, yani 0 yap
  int  $0x80

  test %eax,%eax         // dönen değer < 0 ise çık
  js   exit

  movl %eax,fd           // fd'i sakla

// lseek(fd,0,SEEK_END);
  movl %eax,%ebx
  xorl %ecx,%ecx         // göreli bellek konumunu sıfır yap
  movl $SEEK_END,%edx
  movl $SYS_lseek,%eax
  int   $0x80

  movl %eax,fdlen        // dosya uzunluğunu sakla

  xorl %edx,%edx

// mmap(NULL,fdlen,PROT_READ,MAP_SHARED,fd,0);
  movl %edx,(%esp)
  movl %eax,4(%esp)
  movl $PROT_READ,8(%esp)
  movl $MAP_SHARED,12(%esp)
  movl fd,%eax
  movl %eax,16(%esp)
  movl %edx,20(%esp)

  movl $SYS_mmap,%eax
  movl %esp,%ebx
  int  $0x80

  movl %eax,mappedptr    // göstericiyi sakla

// write(STDOUT, mappedptr, fdlen);
  movl $STDOUT,%ebx
  movl %eax,%ecx
  movl fdlen,%edx
  movl $SYS_write,%eax
  int  $0x80

// munmap(mappedptr, fdlen);
  movl mappedptr,%ebx
  movl fdlen,%ecx
  movl $SYS_munmap,%eax
  int  $0x80
```

```
// close(fd);
  movl fd,%ebx            // dosya tanımlayıcıyı yükle
  movl $SYS_close,%eax
  int  $0x80
exit:
// exit(0);
  movl $SYS_exit,%eax
  xorl %ebx,%ebx
  int  $0x80

  ret
```

Örnek 3. mmap()'i C içerisinde kullanmak

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>

#define STDOUT 1

void
main(void)
{
  char file[] = "mmap.s";
  char *mappedptr;
  int fd, filelen;

  fd = fopen(file, O_RDONLY);
  filelen = lseek(fd, 0, SEEK_END);
  mappedptr = mmap(NULL, filelen, PROT_READ, MAP_SHARED, fd, 0);
  write(STDOUT, mappedptr, filelen);
  munmap(mappedptr, filelen);
  close(fd);
}
```

mmap() parametrelerinin bellekte dizilişleri:

%esp	%esp+4	%esp+8	%esp+12	%esp+16	%esp+20
00000000	filelen	00000001	00000001	fd	00000000

ASM Karşılığı:

```
$ cat mmap.s
.include "defines.h"
.data
  file:
    .string "mmap.s"
  fd:
    .long 0
  filelen:
    .long 0
  mappedptr:
    .long 0
.globl main
```

```
main:
  push %ebp
  movl %esp,%ebp
  subl $24,%esp

// open($file, $O_RDONLY);
  movl $fd,%ebx          // fd kaydediliyor
  movl %eax,(%ebx)

// lseek($fd,0,$SEEK_END);
  movl $filelen,%ebx     // dosya uzunluğu kaydediliyor
  movl %eax,(%ebx)

  xorl %edx,%edx

// mmap(NULL,$filelen,PROT_READ,MAP_SHARED,$fd,0);
  movl %edx,(%esp)
  movl %eax,4(%esp)      // dosya uzunluğu hala %eax içerisinde
  movl $PROT_READ,8(%esp)
  movl $MAP_SHARED,12(%esp)
  movl $fd,%ebx          // dosya tanımlayıcısı yükleniyor
  movl (%ebx),%eax
  movl %eax,16(%esp)
  movl %edx,20(%esp)
  movl $SYS_mmap,%eax
  movl %esp,%ebx
  int  $0x80

  movl $mappedptr,%ebx   // ptr kaydediliyor
  movl %eax,(%ebx)

// write($stdout, $mappedptr, $filelen);
// munmap($mappedptr, $filelen);
// close($fd);

  movl %ebp,%esp
  popl %ebp

  ret
$
```

Bilgi

Yukarıdaki kaynak kodu *sistem çağrısı mmap* (sayfa: 5) örneğindeki kaynak kodundan farklıdır. Yukarıda bulunan kaynak kodu diğer sistem çağrılarını, bu bölümün konusu olmadıkları için, göstermemektedir. Yukarıdaki kod sadece `mmap.s` dosyasını açar, oysa *sistem çağrısı mmap* (sayfa: 5) kaynak kodu komut satırı parametrelerini okumaktadır. **`mmap`** örneği dosya uzunluğunu okumak için de **`lseek`**'i kullanmaktadır.

3.3. Soket Sistem Çağrıları

Soket sistem çağrıları, **`%eax`** içerisine giden, sadece bir sistem çağrı numarası kullanır: **`SYS_socketcall`**. Soket işlevleri `/usr/include/linux/net.h` başlık dosyası içerisinde bulunan ve **`%ebx`**'e kaydedilen bir altişlev numarası yolu ile tanımlanır. Sistem çağrı argümanını gösteren bir gösterici **`%ecx`** içerisinde saklanır. Soket sistem çağrıları da `int $0x80` çağırılarak çalıştırılabilir.

```
/* socket.s */

.text
.globl _start
_start:
 sub $12,%esp

// socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
  movl $AF_INET,(%esp)
  movl $SOCK_STREAM,4(%esp)
  movl $IPPROTO_TCP,8(%esp)

  movl $SYS_socketcall,%eax
  movl $SYS_socketcall_socket,%ebx
  movl %esp,%ecx
  int  $0x80

  movl $SYS_exit,%eax
  xorl %ebx,%ebx
  int  $0x80
  ret
```

4. Komut Satırı Argümanları

Linux çalışabilir dosyalarında komut satırı parametreleri yığıt üzerinde dizilirler. `argc` ilk olarak gelir ve bir boş gösterici tarafından takip edilen, komut satırındaki dizgeleri gösteren göstericilerin oluşturduğu bir dizi tarafından izlenir (`**argv`). Daha sonra çevreye ait göstericilerin tutulduğu bir dizi gelir (`**envp`). Bunlar sembolik makina dili altında kolayca elde edilir ve bu örnek kod ile de gösterilmiştir.

Örnek 4. args.s

```
/* args.s */

.text

.globl _start

_start:
  popl %ecx              // argc

lewp:
  popl %ecx              // argv
  test %ecx,%ecx
  jz   exit

  movl %ecx,%ebx
  xorl %edx,%edx

strlen:
  movb (%ebx),%al
  inc  %edx
  inc  %ebx
  test %al,%al
  jnz  strlen
  movb $10,-1(%ebx)
```

```
// write(1, argv[i], strlen(argv[i]));
  movl $SYS_write,%eax
  movl $STDOUT,%ebx
  int  $0x80

  jmp  lewp

exit:
  movl $SYS_exit,%eax
  xorl %ebx,%ebx
  int  $0x80

  ret
```

5. GCC Satıriçi Sembolik Makina Dili

GCC satıriçi sembolik makina dili hakındaki bu bölüm sadece x86 uygulamalarını kapsayacaktır. Terimler diğer işlemcilerde farklılık göstereceklerdir. Başvurulabilecek diğer belgeler bu belgenin *sonundadır* (sayfa: 13).

GCC içerisinde temel sembolik makina dili sözdizimi oldukça açıktır. Temel sözdizimi ile aşağıdaki gibidir:

```
__asm__("movl %esp,%eax");     // bildik görünüyor ?
```

veya

```
__asm__
("
  movl $1,%eax                 // SYS_exit
  xor  %ebx,%ebx
  int  $0x80
");
```

Sembolik makina dili için girdi ve çıktı olarak kullanılacak bilgilerin tanımlanması ile daha etkili bir kullanım mümkündür. Özel bir giriş/çıkış/değişiklik alanı zorunlu değildir. Kullanım şekli:

```
__asm__("asm_deyimleri" : çıkış : giriş : değişiklik);
```

giriş ve *çıkış* alanları terim belirteçleri ile gösterilir ve parantez içinde gösterilen C değişkeni onu takip eder. *çıkış* teriminden önce, onun bir *çıkış* terimi olduğunu belirtmek için **=** kullanılmalıdır. Birden çok *çıkış*, *giriş* ve *değişiklik* yazmacı bulunabilir. Her bir "değer" virgül (',') ile birbirinden ayrılmalıdır ve toplamda 10'dan fazla değer girilmemelidir. Terim dizgesi ya yazmacın tam adını ya da kısaltmasını içermelidir.

Kısaltma Tablosu

Kısaltma	Yazmaç
a	%eax %ax %al
b	%ebx %bx %bl
c	%ecx %cx %cl
d	%edx %dx %dl
S	%esi %si
D	%edi %di
m	bellek (memory)

Örnek 5.

```
__asm__("test %%eax,%%eax", : /* çıktı yok */ : "a"(foo));
```

veya

```
__asm__("test %%eax,%%eax", : /* çıktı yok */ : "eax"(foo));
```

__asm__'den sonra **__volatile__** komutunu kullanabilirsiniz:

> " bir **__asm__** komutunun silinmesini, taşınmasını ve birleştirilmesini **__asm__** den sonra **__volatile__** yazarak önleyebilirsiniz."

Örnek 6.

(Konsol komut satırından `info gcc "C Extensions" "Extended Asm"` yazarak erişilen "Assembler Instructions with C Expression Operands" sayfasından alınmıştır.)

```
$ cat inline1.c
#include <stdio.h>

int
main(void)
{
  int foo = 10, bar = 15;

  __asm__ __volatile__
    ("addl %%ebxx,%%eax"
      : "=eax"(foo)                          // çıktı
      : "eax"(foo), "ebx"(bar)               // girdi
      : "eax"                                // değişiklik
    );
  printf("foo + bar = %d\n", foo);

  return 0;
}
$
```

Yazmaçların **%** yerine **%%** ile öneklendirildiğini fark etmişsinizdir. Giriş/çıkış/değişiklik alanları kullanıldığında bu zorunludur çünkü genişletilmiş alanlara sahip yazmaçların kullanılması da olasıdır. Bunu daha sonra kısaca açıklayacağım.

"eax" yazmak veya **"ax"**, **"al"** gibi özel bir yazmaç tanımlamak yerine sadece **"a"** yazabilirsiniz. Bu tüm genel amaçlı yazmaçlar için geçerlidir (*Kısaltma tablosuna* (sayfa: 10) bakınız). Bu kullanım kod yazımında her zaman kullanışlı olmadığı için GCC size yazmaç kısaltmaları sağlar. Maksimum 10 (**%0**–**%9**) tane kısaltma olduğundan sadece 10 girdi/çıktı tanımlayabilirsiniz.

```
$ cat inline2.c
int
main(void)
{
  long eax;
  short bx;
  char cl;

  __asm__("nop;nop;nop"); // kodun geri kalanını satıriçi
                          // sembolik makina dilinden ayırmak için

  __volatile__ __asm__(" test %0,%0 test %1,%1 test %2,%2"
```

```
    : /* çıktı yok */
    : "a"((long)eax), "b"((short)bx), "c"((char)cl) );

   __asm__("nop;nop;nop");

   return 0;
}
$ gcc -o inline2 inline2.c
$ gdb ./inline2
GNU gdb 4.18
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnulibc1"...
(no debugging symbols found)...
(gdb) disassemble main
Dump of assembler code for function main:
... start: inline asm ...
0x8048427: nop
0x8048428: nop
0x8048429: nop
0x804842a: mov 0xfffffffc(%ebp),%eax
0x804842d: mov 0xfffffffa(%ebp),%bx
0x8048431: mov 0xfffffff9(%ebp),%cl
0x8048434: test %eax,%eax
0x8048436: test %bx,%bx
0x8048439: test %cl,%cl
0x804843b: nop
0x804843c: nop
0x804843d: nop
... end: inline asm ...
End of assembler dump.
$
```

Gördüğünüz gibi, üretilen satıriçi sembolik makina dili kodu girdi bölümünde belirtilen değişkenlerin değerlerini yazmaçlara atar ve sonra gerçek kodu üretir. Derleyici, terim boylarını değişkenlerin boylarından yararlanarak otomatik olarak bulur ve böylece `%0`, `%1` ve `%2` kısaltmaları ile gösterilen yazmaçlar değerlerini alır. (Yazmaç kısaltmalarını kullanırken komut kısmında terim boyutunu belirtmek derleme sırasında hatalara neden olabilir).

Kısaltmalar terim belirteçlerinde de kullanılabilir. Bu işlem, yazılımcının 10'dan fazla girdi/çıktı tanımlaması yapmasına izin vermez. Bu işlemin aklıma gelen tek kullanım biçimi, terim belirteci olarak **"q"** kullanılan durumlardır. Terim belirteci olarak **"q"** kullanıldığı zaman derleyici kendisi **a**, **b**, **c** veya **d**'den uygun olanı seçer. Eğer yazmaç ataması **"q"** ile yapılmışsa hangi yazmacın kullanıldığını tahmin edemeyiz ve değişiklik alanıda belirtemeyiz, bu sebepten sadece yazmaç ile ilgili rakam yazılır. Örnek

```
$ cat inline3.c
#include <stdio.h>

int
main(void)
{
  long eax=1, ebx=2;

  __asm__ __volatile__ (
    "add %0,%2"
```

```
    : "=b"((long)ebx)
    : "a"((long)eax), "q"(ebx)
    : "2"
  );

  printf("ebx = %x\n", ebx);

  return 0;
}
$
```

6. Derleme

Sembolik Makina Dili yazılımlarını derlemek normal C programlarını derlemeye çok benzer. Eğer yazılımınız "Liste 1"deki gibi ise normal bir C programı gibi derleyebilirsiniz. Eğer `main` yerine `_start` kullanmışsanız, "Liste 2"deki gibi, derleme işlemi birazcık değişecektir:

Liste 1

```
$ cat write.s
.data
hw:
  .string "hello world\n"
.text
.globl main
main:
  movl $SYS_write,%eax
  movl $1,%ebx
  movl $hw,%ecx
  movl $12,%edx
  int  $0x80
  movl $SYS_exit,%eax
  xorl %ebx,%ebx
  int  $0x80
  ret
$ gcc -o write write.s
$ wc -c ./write
  4790 ./write
$ strip ./write
$ wc -c ./write
  2556 ./write
```

Liste 2

```
$ cat write.s
.data
hw:
  .string "hello world\n"
.text
.globl _start
_start:
  movl $SYS_write,%eax
  movl $1,%ebx
  movl $hw,%ecx
  movl $12,%edx
  int  $0x80
  movl $SYS_exit,%eax
  xorl %ebx,%ebx
  int $0x80

$ gcc -c write.s
$ ld -s -o write write.o
$ wc -c ./write
  408./write
```

`-s` parametresi isteğe bağlıdır ve sadece boyutu kısaltılmış çalışabilir bir ELF dosyası yaratır. Ayrıca bu yöntem (Liste 2'deki),derleyicinin, normalde olduğu gibi, fazladan giriş ve çıkış deyimleri eklemediği için, daha küçük çalışabilir dosyalar üretir.

7. Bağlantılar

- http://www.linuxassembly.org/
- GNU Assembler Manual[B9]
- GNU C Compiler Manual[B10]
- GNU Debugger Manual[B11]

- Constraints for asm Operands[B12]
- AT&T Syntax Reference[B13]

A. Örnek Kodlar

Örnek 7. defines.h

```
/* defines.h */

SYS_exit              = 1
SYS_fork              = 2
SYS_write             = 4
SYS_open              = 5
SYS_close             = 6
SYS_execve            = 11
SYS_lseek             = 19
SYS_dup2              = 63
SYS_mmap              = 90
SYS_munmap            = 91
SYS_socketcall        = 102
SYS_socketcall_socket = 1
SYS_socketcall_bind   = 2
SYS_socketcall_listen = 4
SYS_socketcall_accept = 5

SEEK_END              = 2
PROT_READ             = 1
MAP_SHARED            = 1

AF_INET               = 2
SOCK_STREAM           = 1
IPPROTO_TCP           = 6

STDOUT                = 1
```

Örnek 8. daemon.s

```
/* daemon.s */

BIND_PORT = 0xff00   // 255

.data
SOCK:
  .long 0x0
LEN:
  .long 0x10
SHELL:
  .string "/bin/sh"

.text
.globl _start
_start:
  subl $0x20,%esp

// socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

  movl $SYS_socketcall,%eax
  movl $SYS_socketcall_socket,%ebx
  movl $AF_INET,(%esp)
  movl $SOCK_STREAM,0x4(%esp)
  movl $IPPROTO_TCP,0x8(%esp)
```

```
  movl %esp,%ecx
  int  $0x80

// save sockfd
  movl %eax,SOCK

  xorl  %edx,%edx
// bind(%eax, %esp+0xc, 0x10);
  movw $AF_INET,0xc(%esp)
  movw $BIND_PORT,0xe(%esp)
  movl %edx,0x10(%esp)
  movl %eax,(%esp)
  leal 0xc(%esp),%ebx
  movl %ebx,0x4(%esp)
  movl $0x10,0x8(%esp)
  movl $SYS_socketcall,%eax
  movl $SYS_socketcall_bind,%ebx
  int  $0x80

  movl  SOCK,%eax

// listen(%eax, 0x1);
  movl %eax,(%esp)
  movl $0x1,0x4(%esp)
  movl $SYS_socketcall,%eax
  movl $SYS_socketcall_listen,%ebx
  int  $0x80

  movl  SOCK,%eax

// accept(%eax, %esp+0xc, LEN);
  movl  %eax,(%esp)
  leal  0xc(%esp),%ebx
  movl  %ebx,0x4(%esp)
  movl  $LEN,0x8(%esp)
  movl  $SYS_socketcall,%eax
  movl  $SYS_socketcall_accept,%ebx
  int   $0x80

// for(i=2;i>-1;;i--) dup2(%eax,i)
  movl $0x2,%ecx
DUP2LOOP:
  pushl %eax
  movl  %eax,%ebx
  mov   $SYS_dup2,%eax
  int   $0x80
  dec   %ecx
  popl  %eax
  jns   DUP2LOOP

// execve(SHELL, { SHELL, NULL }, NULL );
  movl $SYS_execve,%eax
  movl $SHELL,%ebx
  movl %ebx,(%esp)
  movl %edx,0x4(%esp)
  movl %esp,%ecx
  int  $0x80
```

```
// _exit(0)
  movl $SYS_exit,%eax
  movl %edx,%ebx
  int  $0x80

  ret
```

Notlar

a) Belge içinde dipnotlar ve dış bağlantılar varsa, bunlarla ilgili bilgiler bulundukları sayfanın sonunda dipnot olarak verilmeyip, hepsi toplu olarak burada listelenmiş olacaktır.

b) Konsol görüntüsünü temsil eden sarı zeminli alanlarda metin genişliğine sığmayan satırların sığmayan kısmı ¬ karakteri kullanılarak bir alt satıra indirilmiştir. Sarı zeminli alanlarda ¬ karakteri ile başlayan satırlar bir önceki satırın devamı olarak ele alınmalıdır.

(B9) http://www.gnu.org/software/binutils/manual/gas-«2.9.1/as.html

(B10) http://gcc.gnu.org/onlinedocs/

(B11) http://www.gnu.org/software/gdb/documentation/

(B12) http://gcc.gnu.org/onlinedocs/gcc-«4.0.2/gcc/Constraints.html

(B13) http://www.gnu.org/software/binutils/manual/gas-«2.9.1/html_mono/as.html#SEC196

Bu dosya (asm–syntax.pdf), belgenin XML biçiminin TEXLive ve belgeler-xsl paketlerindeki araçlar kullanılarak PDF biçimine dönüştürülmesiyle elde edilmiştir.

6 Şubat 2007