# Linux Makina Dili NASIL

Yazan:
**Konstantin Boldyshev**
Linux Assembly [B1]
<konst (at) linuxassembly.org>

Yazan:
**Francois-Rene Rideau**
Tunes project [B2]
<fare (at) tunes.org>

Çeviren:
**Oğuz Yarımtepe**
<oguzy (at) comu.edu.tr>

Mart 2006

**Özet**

Bu, Linux Sembolik Makina Dili NASIL, sürüm 0.6g belgesidir. Bu belge *özgür* programlama araçları kullanarak, genelde IA–32 (i386) platformunda, Linux işletim sistemi için veya Linux işletim sistemi kaynaklı, sembolik makina dilinde nasil programlama yapılacağını anlatmaktadır. İçerik diğer yazılım veya donanıma uygulanamayabilir.

## Konu Başlıkları

## Yasal Açıklamalar

# 1. Giriş

**Bilgi**

Eğer NASIL'lara yabancı değilseniz veya sadece buradaki sembolik makina diliyle ilgisiz ıvır zıvırları okumaktan nefret ediyorsanız bu bölümü geçebilirsiniz.

## 1.1. Belgenin son sürümü

Bu belgenin en son resmi sürümleri Linux Sembolik Makina Dili (Linux Assembly)[B5] ve LDP[B6] sitelerinden elde edilebilir.

## 1.2. Önsöz

Bu belge, özellikle Linux işletim sistemi altında, 32–bit x86 sembolik makina dili kullanarak *özgür yazılım* altında programlayan ya da programlamak isteyenlerin sorularına cevap olmayı amaçlamaktadır. Pek çok yerde Evrensel Kaynak Konumlayıcılar (Universal Resource Locators – URL) bazı yazılım veya belge depoları için verilmiştir. Bu belge ayrıca, temel amacı olmamakla beraber, özgür olmayan (non–free), x86 tabanlı olmayan (non–x86) veya 32 bit olmayan (non–32–bit) derleyicilerle ilgili diğer belgelere de işaret etmektedir. Aynı zamanda, doğrudan sembolik makina diliyle ilgili olmayan platforma bağlı meselelerde, favori platformunuzda (artık her ne ise) programlama yapmak için SSS (Sıkça Sorulan Sorular) ve belgeler olduğuna da dikkat edin.

Sembolik makina dili ile programlamak temel ilgi alanı, C derleyicilerinin ihtiyaç duyulan ifadeleri sağlamakta yetersiz kaldığı (performans oldukça nadir bir meseledir), işletim sisteminin sindirim sistemi, yorumlayıcı, derleyici ve oyunları olduğundan, bu tür yazılımları geliştirmek üzerine odaklanıyoruz.

Eğer *özgür yazılımın*[B7] ne olduğunu bilmiyorsanız, pekçok yazılımda kullanılan ve pekçoğunun lisansının modeli olan, GNU Genel Kamu Lisansı[B8]'nı (`GPL` veya `copyleft`) lütfen *dikkatle* okuyunuz. Genelde adı `COPYING` (veya `COPYING.LIB`) olan bir dosya içerisinde gelmektedir. Özgür Yazılım Vakfı[B9] (`FSF`) tarafından yayınlanmış eserler de size yardımcı olabilir. Özellikler, özgür yazılımın en ilgi çekici özelliği başvurabileceğiniz veya düzeltebileceğiniz veya hatta bazen ödünç alabileceğiniz kaynak koduyla gelmesidir. Size özel lisansı okuyun ve ona bağlı kalın.

## 1.3. Katkıda Bulunanlar

Bu, etkileşimli olarak gelişen bir belgedir: Sorular sormak, sorulara cevap vermek, verilen cevapları düzeltmek, yeni yazılımlara işaret etmek, şu anki sürdürücüyü sayfadaki hatalar veya eksiklikler ile ilgili uyarmak için özellikle davet ediliyorsunuz. Tek kelimeyle, katılın!

Katılımda bulunmak için, *geliştirici* (sayfa: 1) ile iletişime geçin.

**Bilgi**

Yazım sırasında, bu kişi *Konstantin Boldyshev* (sayfa: 1) idi, *Francois–Rene Rideau* (sayfa: 1) artık yok (versiyon 0.5'den beri). Ben (*Fare* (sayfa: 1)) uzun bir süreden beri bu belgeyi sürdürecek, yerime geçecek bir üstat (hacker) aramaktaydım ve sizlere değerli halefim olarak *Konstantin* (sayfa: 1)'i sunmaktan memnunum.

## 1.4. Tercümeler

Bu NASIL belgesinin Kore diline çevirisi `http://kldp.org/HOWTO/html/Assembly-«HOWTO/` adresinden, Türkçe çevirisi ise `http://belgeler.org/howto/assembly-«howto.html` adresinden

edinilebilir. Tamamlanmamış Rusça çevirisi ise
http://volgograd.lug.ru/wiki/GrableVodstvo/articles/AssembleInLinux/
adresindedir. Ayrıca, eski NASIL sürümleri için Fransızca çeviriler vardı, fakat bulamadım.

# 2. Sembolik makina diline ihtiyacınız var mı?

Aslında, her ne yapıyorsanız bölmek istemem, ama burada zor kazanılmış tecrübelerin sonucu olarak bazı tavsiyelerde bulunacağım.

## 2.1. Artılar ve Eksiler

### 2.1.1. Sembolik makina dilinin (Assembly) avantajları

Sembolik makina dili (Assembly) pekçok düşük seviyeli şeyi ifade edebilir:

- makinaya bağlı yazmaç ve Giriş/Çıkışlara (I/O) erileşilirsiniz
- çoklu yazılım parçalarının ya da donanım aygıtlarının ölümcül kilitlenmesini içeren kritik bölümlere özgü kod davranışlarını kontrol edebilirsiniz.
- alışıldık derleyicinizin herkesçe kabul görmüş kurallarını kırabilirsiniz, ki bu da bazı eniyilemelere izin vermektedir (bellek tahsisiyle, evrelerle (threading), çağrı kurallarıyla ilgili kuralları geçici olarak kırmak gibi).
- kod parçaları arasında, uygun olmayan (örn. değişik derleyiciler tarafından üretilen veya düşük seviye arayüzlerle ayrılan) kurallar kullanarak arayüzler tasarlayabilirsiniz.
- işlemcinizin alışılmadık programlama kiplerine erişebilirsiniz (örn. açılış arayüzü için 16 bitlik kip, aygıt yazılımları, Intel bilgisayarlardaki kalıtsal kodlar).
- kötü ve eniyileme yapmayan derleyicilerle uyumlu çalışacak sıkı döngüler için oldukça hızlı kodlar üretebilirsiniz. (Ancak, eniyileme yapabilen özgür derleyiciler var!)
- her ne kadar başka herhangi birininkine ait olmayacak olsa da, size özel aygıt ayarlarınız için el ile eniyilenmiş mükemmel bir kod üretebilirsiniz
- yeni dilinizin eniyileme yapabilen derleyicisi için bir kod yazabilirsiniz (bu, çok azımızın yapacağı ve hatta pek de sıklıkla yapmayacağı bir iştir).
- örn. kendi kodunuzu tamamiyle kontrol altına alabilirsiniz

### 2.1.2. Sembolik makina dilinin (Assembly) dezavantajları

Sembolik makina dili, oldukça düşük seviyeli bir dildir (bundan daha aşağıda ikilik komutları el ile kodlamak bulunmaktadır). Bu da şu manalara gelmektedir:

- ilk başlarda yazmak, oldukça uzun ve can sıkıcıdır
- hata yapmaya oldukça meyillidir
- hatalarınızı takip etmek oldukça zor olabilir
- kodunuzu anlamak ve değiştirmek oldukça zordur, örn. bakımını yapmak
- sonuç, şu anda veya gelecekte var olacak mimarilere taşınabilir değildir

- kodunuz aynı mimarinin sadece belli bir gerçekleştirimi için eniyilenecektir: mesela, Intel uyumlu platformlar arasında, herbir CPU tasarımı ve türevi (işlemci birimlerinin nispi gecikme süresi, üretilen iş (through–output) ve kapasitesi, önbellekler (cache), RAM, taşıt (bus), diskler, FPU, MMX, 3DNOW, SIMD uzantılarının varlıkları, v.b.) tamamiyle farklı eniyileme tekniklerini ifade etmektedir. İşlemci (CPU) tasarımları halihazırda şunları içermektedir: Intel 386, 486, Pentium, PPro, PII, PIII, PIV; Cyrix 5x86, 6x86, M2; AMD K5, K6 (K6–2, K6–III), K7 (Athlon, Duron). Yeni tasarımlar yukarılara doğru tırmanmayı sürdürmektedir, dolayısıyle ne bu listenin ne de kodunuzun güncel kalacağını ummayın.
- çok az bir ayrıntı üzerinde fazlaca zaman harcarsınız, hızlanmanın büyük kısmını oluşturan küçük ve geniş algoritmik tasarımlar üzerine odaklanamazsınız (örn. liste/diziler üzerinde değişikler yapan hızlı ve ilkel nesneler oluşturmak için çokça zaman harcayabilirsiniz; oysa sadece hesaba dayalı bir adresleme (hash) tablosu veya başka bir yaklaşımda ikilik ağaç ya da pekçok CPU kümesine dağıtılmış olan yüksek seviyeli bir yapı programınızı çok daha hızlandırırdı).
- algoritmik tasarımınızdaki ufak bir değiliklik var olan sembolik makina kodunuzu tamamiyle geçersiz hale getirebilir. Bu durumda ya tamamen tekrar yazmaya (yazabilemeye) hazırsınızdır veya belirli bir algoritmik tasarımı yapmaktan sıkılmışsınızdır.
- Standart karşılaştırmalı değerlendirmeden (benchmark) uzak olmayan kod üzerinde, eniyileme yapan ticari derleyiciler elle kodlanmış sembolik makina dili gerçekleştirirler (aslında, RISC mimarisine göre bu durum x86 mimarisi üzerinde daha az geçerlidir ve belki de geniş bir şekilde bulunan/kullanılan derleyiciler için de daha az doğrudur; herneyse, tipik bir C kodu için GCC oldukça iyidir);
- Ve her durumda, comp.compilers[B18]'da bulunan yönetici John Levine'nin dediği gibi,

  > "derleyiciler karmaşık veri yapılarının kullanımını oldukça kolay hale getirmektedir ve derleyiciler işin yarısından sonra sıkılmamakta ve güvenilir oldukça güzel kodlar üretmektedir."

  Prosedür ve modül sınırları arasında kodu eniyilerken, aynı zamanda da tüm (büyük) program boyunca *düzgün* biçimde kod dönüşümleri üreteceklerdir.

### 2.1.3. Değerlendirme

Tüm bunlardan sonra, sembolik makina dili kullanmanın bir ihtiyaç olduğunu düşünebilirsiniz ve hatta ihtiyaç olmadığı bazı yerlerde kullanmak çok da faydalı olabilir. Şunları yapmak isteyeceksiniz:

- sembolik makina dilinin kullanımını küçültmek
- iyi tanımlanmış arayüzler içine bu kodları koymak (encapsulate)
- sembolik makine dili dışında yüksek seviyeli dillerle tanımlanmış yapılar tarafından sembolik makine dili kodunuzun otomatik üretilmesi (örn. GCC satıriçi (inline) makrolar)
- bu programları otomatik araçların sembolik makine dili koduna dönüştürmesi
- bu kodun eğer mümkünse eniyilenmesi
- yukarıdakilerin tümü, örn. bir derleyici (ya da derleyiciye bir eklenti) arka ucu (back–end)

Sembolik makina diline gerek duyulsa (örn. İşletim sistemi geliştirmek) bile, yukarıdakilerin çok daha fazlasına gerek duyulmayacağını göreceksiniz ve de üstteki prensipler de varlığını koruyacaktır.

Bununla ilgili olarak Linux çekirdek kaynaklarına bakınız: ne kadar az sembolik makina diline gerek duyulursa, neticesinde hızlı, güvenilir, taşınabilir ve sürdürülebilir işletim sistemi oluşmaktadır. Hatta DOOM gibi başarılı bir oyun dahi yoğun şekilde C ile yazılmıştır, sadece küçük bir kısmı hızı arttırmak için sembolik makine dili ile yazılmıştır.

## 2.2. Sembolik makina dili nasıl kullanılmaz

### 2.2.1. Başarılı kodu gerçeklemek için adımlar

comp.compilers[B19]'daki Charles Fiterman'ın bilgisayara karşın insanın ürettiği sembolik makina kodlarıyla ilgili söylediği gibi:

```
İnsanoğlu her zaman kazanmak zorundadır ve işte bu da nedenidir.

Birincisi insanoğlu herşeyi yüksek seviyeli dilde yazar.
İkincisi üzerinde zaman harcadığı sıcak noktaları bulacak
şekilde programın taslağını hazırlar.
Üçüncüsü elinde bu kısımlar için derleyicinin ürettiği kod vardır.
Dördüncüsü makina kodu üzerinde ufak gelişmeler sağlayarak onlara
bir canlılık kazandırabilir.

İnsanoğlu kazanır çünkü makinaları kullanabilmektedir.
```

### 2.2.2. Eniyileme yapan derleyiciler ile dilleri

Diğerleri arasında ObjectiveCAML, SML, CommonLISP, Scheme, ADA, Pascal, C, C++ gibi dillerin programınızın şişkinliğini eniyileyecek özgür derleyicileri vardır ve genellikle, sıkı döngüler için bile elle yazılmış sembolik makina kodundan daha iyisini üretirler, bu arada da daha yüksek seviyeli ayrıntılar üzerinde odaklanmanızı sağlarlar ve de belli bir kararlı düzeye geldikten sonra da yukarıda bahsedilen şekilde belli miktar başarıma el koymanızı yasaklamazlar. Elbette, bu dillerin çoğu için eniyileme yapan ticari derleyiciler de vardır!

Bazı diller C kodu üreten ve sonrasında eniyilemesini C derleyicisine yaptıran derleyicilere sahiptir: LISP, Scheme, Perl ve diğer pekçoğu. Hız oldukça iyidir.

### 2.2.3. Kodunuzu hızlandıracak için genel adımlar

Kodunuzun hızlı çalışmasını sağlamak için, analiz araçlarından birinin kodunuzun belli bir yerini sürekli bir performans darboğazı olarak tanımlaması gerekmektedir.

Bu nedenle, bir kod parçasını çok yavaş olarak tanımladıktan sonra, şunları yapmalısınız:

- öncelikle daha iyi bir algoritma kullanmaya çalışın;
- yorumlamak yerine onu derleyin;
- daha sonra derleyicinizdeki eniyilemeyi etkinleştirip onunla oynayın
- daha sonra derleyiciye nasıl eniyileme yapacağına dair ipuçları verin (LISP'de bilgi girmek; GCC ile yazmaç kullanmak; pekçok derleyicideki seçenekler, v.b)
- bundan sonra ancak son çare sembolik makina dilidir.

Son olarak, sembolik makina diliyle yazmayı bitirmeden önce, üretilen kodu incelemelisiniz, problemin gerçekten de kötü kod üretiminden kaynaklandığını görmelisiniz, ki bu herzaman sanılan durum olmayabilir: derleyicinin ürettiği kod sizin yazdığından daha iyi olabilir, özellikle modern çoklu ardışık düzen (multi–pipelined) mimarilerinde! Programın yavaş olan kısımları esas olarak böyle olabilir. Hızlı işlemcili modern mimarilerde temel sorun, bellek erişim gecikmeleri, önbellek atlamaları, TLB (TLB–misses) kayıpları ve sayfa hatalarından kaynaklanmaktadır; yazmaç kullanımı gereksiz olmaktadır ve veri yapılarını daha kazançlı bir şekilde ve bellek erişimini daha iyi yapmanın yollarını tekrar düşüneceksiniz. Belki de tamamen farklı bir yaklaşım sorunun çözümüne yardımcı olabilir.

### 2.2.4. Derleyicinin ürettiği kodu incelemek

Derleyicinin ürettiği kodu incelemek için pek çok neden vardır. İşte size bu kodla neler yapacağınız:

- üretilen kodun el yordamıyla geliştirilip geliştirilemeyeceğini kontrol edin (veya derleyici seçenekleriyle oynayarak)
- durum böyle ise, o zaman üretilmiş kod ile başlayın, onu yeniden yazmak yerine değiştirin
- genel olarak, dış dünyaya karşı sizin sembolik makine rutinlerinizi anlayan üretilen kodu değişiklik yapmak için bir yama gibi kullanın
- derleyicinizdeki hataları takip edin (tahminim en az sıklıkla)

Sembolik makina dilinizin oluşması için standart yol derleyicinizi `-S` parametresiyle çalıştırmaktır. Bu, GCC C derleyicisini de içeren (GCC) pekçok Unix derleyicinde çalışmaktadır. GCC'ye gelince, `-fverbose-asm` komut satırı parametresiyle çok daha anlaşılabilir sembolik makina kodları üretecektir. Elbette, iyi sembolik makina kodu elde etmek istiyorsanız, herzamanki eniyileme seçeneklerinizi ve ipuçlarını unutmayınız!

## 2.3. Linux ve Assembly

Muhtemelen farkettiğiniz üzere, genel olarak, Linux programlamada sembolik makine dilini kullanmaya ihtiyaç duymazsınız. DOS'takinin aksine, Linux sürücülerini sembolik makine dili ile yazmanız zorunlu değildir (aslında, eğer gerçekten istiyorsanız yapabilirsiniz). Ve günümüzün eniyileme yapan derleyicileriyle, farklı işlemcilerdeki hızı dikkate alıyorsanız, C ile yazmak çok daha basittir. Yine de, bunu okuyorsanız, C/C++ yerine sembolik makine dili kullanmak için bir nedeniniz olabilir.

Sembolik makine dilini kullanmak *ihtiyacında* olabilirsiniz veya sadece kullanmak *isteyebilirsiniz*. Kısaca, sembolik makine dili krallığına dalmaktaki temel pratik sebepler (*ihtiyaç*) *kısa kodlar* ve *libc bağımsızlıklarıdır*. Pratik olarak ve en sık karşılaşılan nedense (*istek*), 20 yıllık ve herşeyi sembolik makine dili ile yapma alışkanlığı olan eski bir bilgisayar kurdu olmaktır.

Yine de, eğer Linux'u gömülü bir sisteme yerleştiriyorsanız, tüm sistemin büyüklüğüne göre kısa olabilirsiniz: çekirdeği, **`libc`** ve tüm diğer şeylerin **`(file|find|text|sh|v.b)`** uygulamalarını birkaç yüz kilobyte'a oturtmalısınız ve her kilobyte'ın değeri çok fazladır. Bundan dolayı, mümkün olan yollardan birisi, sistemin bazı (veya tüm) kısımlarını sembolik makine dili ile yazmaktır ve bu durum da pekçok yer tasarrufu sağlayacaktır. Mesela, basit bir, sembolik makine dili ile yazılmış, **`httpd`** 600 byte'tan az tutmaktadır; kernel, httpd ve ftpd içeren bir sunucuyu 400kb veya daha az boyutta ayarlayabilirsiniz... Bunu düşünün.

# 3. Çeviriciler (Assemblers)

## 3.1. GCC Satıriçi Sembolik Makina Dili

GNU projesinin kalbinde bulunan 32 bitlik eniyileme yapan, iyi bilinen GNU C/C++ derleyicisi (GCC), x86 mimarisini oldukça iyi bir şekilde desteklemektedir ve C kodları içerisine, yazmaç tahsisini özel olarak belirtilerek veya GCC'ye bırakılabilecek şekilde, sembolik makina kodları gömülmesine olanak sağlamaktadır. GCC pekçok platformda, *BSD, VSTa, OS/2, *DOS, Win*, v.b., özellikle Linux'da çalışır.

### 3.1.1. GCC'yi nereden bulabiliriz

GCC ana sayfası `http://gcc.gnu.org/` adresindedir.

GCC'nin DOS tabanlı çalışan sürümüne DJGPP[B21] denir.

GCC'nin Win32 tabanında çalışan iki sürümü vardır: cygwin[B22] ve mingw[B23]

CC'nin bir de OS/2 tabanlı EMX olarak isimlendirilen bir sürümü vardır; DOS altında da çalışır ve unix–benzeri kütüphane yordamlarının pekçoğunu içerir. Şu siteye bir göz atın: ftp://ftp.leo.org/pub/comp/os/os2/leo/gnu/emx+gcc/.

### 3.1.2. GCC satır içi sembolik makina dili için belgeler nerede

GCC'nin belgelendirmesi belge dosyalarını TeXinfo biçiminde içerir. TeX ile bunları derleyip çıktılarını görebilirsiniz, .info'ya çevirip emacs ile göz atabilirsiniz veya .html'ye çevirebilirsiniz ya da hemen hemen istediğiniz herhangi bir biçime; istediğiniz herhangi bir şeye çevirin (düzgün araçlarla) veya olduğu haliyle okuyun. .info dosyaları GCC için yapılan herhangi bir iyi yüklemede bulunur.

Bakmak için doğru yer C Extensions::Extended Asm::'dir.

Invoking GCC::Submodel Options::i386 Options:: kısmı da size yardımcı olabilir. Özellikle, yazmaçlara i386'ya özgü kısıtlandırılmış isimler verir: abcdSDB sırasıyla %eax, %ebx, %ecx, %edx, %esi, %edi ve %ebp (%esp için bir harf yok)'ye karşılık gelmektedir.

DJGPP Oyun kaynağı (sadece oyun ustaları için değil) özellikle sembolik makina diliyle ilgili bir sayfaya sahipti, fakat artık erişilebilir değil. Herşeye rağmen oradaki veriler *DJGP* (sayfa: 9) ve diğer faydalı bilgilerin madenini içeren belgeler tekrar bir araya getirilmiştir: http://www.delorie.com/djgpp/doc/brennan/ ve http://www.castle.net/~avly/djasm.html

GCC, makina diline çeviri için GAS'a (aşağıyı inceleyiniz) dayanır; şunu unutmayın ki satır içi asm yüzdelik karakterlerinin çift tırnak içerisine alınmasına ihtiyaç duyar ve GAS'a onlar aktarılacaktır. Aşağıdaki *GAS* (sayfa: 11) bölümüne bakınız.

Linux çekirdeğinin kaynağının alt dizini olan linux/include/asm-i386/ altında *çokça* örnek bulabilirsiniz.

### 3.1.3. GCC'yi uygun satıriçi sembolik makina kodu üretmesi için çağırmak

Çekirdek başlık dosyalarındaki (ve büyük ihtimal sizin kendi başlığınızda da, eğer kendi sembolik makina programlamanızı Linux çekirdeğindeki gibi temiz yaparsanız) sembolik makina yordamları *harici satıriçi (extern inline)* işlevlere gömülü olduğundan, GCC, kullanabilir olduğu bu yordamlar için, -O seçeneği ile çağrılmalıdır (veya -O2, -O3, v.b.). Aksi halde, kodunuz derlenebilir, fakat, programınızın bağlı olduğu kütüphanelere karşın satıriçi olmayan harici kütüphanelere bakıyor olacağından, düzgün şekilde bağlanmaz. Başka bir yöntem ise yordamların sonçare sürümlerini içeren kütüphanelere bağlantı yapmaktır.

Satıriçi çeviri (sembolik makine diline) -fno-asm ile devre dışı bırakılabilir, böylece derleyici, genişletilmiş satıriçi kodlar kullanılırken çalışmayacaktır veya bağlayıcının algılıyamacağı asm() isimli işleve çağrılarda bulunacaktır. Bu bayrağın çalışmasına karşı durum için, -fasm bayrağı ile asm anahtar kelimesine karşı olan davranış geri kazandırılır.

Daha genel olarak x86 platformunda GCC için iyi derleme seçenekleri şu şekildedir:

```
gcc -O2 -fomit-frame-pointer -W -Wall
```

-O2 pekçok durumda iyi bir eniyileme seviyesidir. Bunun yanında eniyileme uzun zaman alır ve de kodun olduğundan daha büyümesine sebep olur, fakat neticede olduğundan biraz daha hızlı çalışır hale gelir; böylesi bir aşırı eniyileme sembolik makina dilinde her halükarda yaptığınız sıkı döngülerde (eğer varsa) faydalı olabilir. Eğer gerçekten de bazı seyrek dosyalar için oldukça güçlü bir eniyilemeye ihtiyaç duyarsanız, -O6'yı kullanmayı deneyin.

-fomit-frame-pointer kullanımı, üretilen kodun aptal çerçeve gösterici onarımını (frame pointer maintenance) atlamasını sağlar, bu da kodun daha küçük ve hızlı olmasını sağlar ve de bir yazmacı sonraki kullanımlar için boşaltır. (**gdb**)'nin kolay kullanımına müsade etmez, fakat bunları kullandığınız zaman, boyut ve hızı artık dert etmezsiniz.

`-W -Wall` ile tüm faydalı uyarıları etkinleştirilir ve aleni yapılan aptalca hataları yakalamanızı sağlar.

İşlemciye özel, `-m486` gibi komutlar ekleyerek üretilen kodun size özel işlemciye daha uygun halde olmasını sağlayabilirsiniz. Modern GCC'nin komut sadırı seçenekleri `-mpentium` ve benzeri şeklindedir (PGCC[B29]'de daha da fazladır), oysa ki GCC 2.7.x ve daha eski sürümleri böyle değildir. İşlemciye özel en iyi seçeneklerin neler olacağı Linux çekirdeği içindedir. Daha ayrıntılı bilgi için şu anki GCC'nizin belgelendirmesini inceleyiniz.

`-m386` boyutta eniyilemeyi sağlamaktadır, bu da belleğin sıkı (tight) olmasını ve/veya yüklenmiş olan bilgisayarlarda hızlanmayı sağlamaktadır, çünkü büyük programlar takas alanı kullanımına sebep olmaktadır, bu da daha büyük kodlar için sayılandan daha çok eniyilemeye meyledilecek demektir. Bu tür ayarlarda, C dilini kullanmayı bırakmak faydalı olabilir, bunun yerine işlevsel bir dil ve/veya FORTH gibi kod çözümlemeye meyilli bir dil kullanın ve bayt seviyesi (bytecode) – veya sözcük seviyesi (wordcode) – tabanlı bir gerçekleme kullanın.

Kod üretim seçeneklerini dosyadan dosyaya değiştirebilirsiniz, böylece başarımın önemli olduğu dosyalar azami eniyileme kullanacaklardır, oysa ki diğerleri boyutlarına göre eniyileneceklerdir.

Daha çok eniyileme için, `-mregparm=2` seçeneği ve/veya buna karşılık gelen işlev davranışları yardım edebilir, fakat yabancı kodlara, **`libc`** *dahil*, ilintileme yaparken pekçok soruna sebep olabilir. Yabancı işlevlerin bildirimi için yollar vardır, böylece doğru çağrı sırası üretilmiş olur veya yabancı kütüphanelerinizi aynı yazmaç tabanlı çağrı uzlaşımlarını kullanacak şekilde tekrar derlemeyi isteyebilirsiniz.

`/usr/lib/gcc-lib/i486-linux/2.7.2.3/specs` dosyasını (veya her nerede ise) düzenleyerek bu secenekleri öntanımlı olarak ekleyebilirsiniz (`-W -Wall` seçeneklerini eklememek daha iyi olacaktır). GCC'ye özel dosyalarınızın sisteminizde nerede olduğunu **`gcc -v`** ile öğrenebilirsiniz.

### 3.1.4. Makro desteği

GCC, satır içi sembolik makina dilinde yazmaç kısıtlamalarını belirlemenizi sağlar (ve gerektirir), böylece eniyileyicinin bunlar hakkında herzaman bilgisi olur; satır içi sembolik makina kodu gerçekten bir takım kalıplardan oluşmuştur, zorunlu şekilde oluşturulan kodlardan değil.

Böylece sembolik makina kodunuzu CPP makrolarının ve satır içi C fonsiyonlarının içine koyabilirsiniz ve böylece herhangi biri onu herhangi bir C işlevi/makrosu olarak kullanabilir. Satıriçi işlevler makrolara çok benzer, fakat kullanmak için daha temizdir. Tüm bu durumlarda, kodun tekrar edeceğine (dublicate) dikkat edin, bundan dolayı asm kodunda sadece yerel etiketler (`1:` tanımlanmış olmalıdır. Yine de, bir makro yerel olmayan etiketlerin parametre olarak aktarılmasına izin verecektir (veya sizin bazı ek meta programlama yöntemleri kullanmanız gerekecektir). Ayrıca şuna da dikkat edin, satıriçi sembolik makina kodu üretmek, içerisinde olası hataların da yayılmasına sebep olacaktır; bundan dolayı böylesi bir satıriçi asm kodundaki yazmaç kısıtlamalarını iki kere kontrol edin.

Son olarak, C dilinin kendisi, sembolik makina dili programlama için iyi bir soyutlama katmanı olabilir ve bu da sizi sembolik makina dilinin pekçok sorunlarından kurtarır.

## 3.2. GAS

GAS, GCC'nin güvendiği GNU Çeviricisidir (Assembler).

### 3.2.1. Nereden Bulurum

GCC'yi bulacağınız yerdeki ikilik uygulama paketleri (binutils paketi) içerinde bulabilirsiniz. İkilik uygulama paketinin son sürümü `http://sources.redhat.com/binutils/` adresinden temin edilebilir.

### 3.2.2. Şu AT&T sözdizimi de ne ola ki

GAS 32 bit Unix derleyicilerini desteklemek için icat edildiğinden, standart AT&T sözdizimini kullanmaktadır. Bu söz dizimi UNIX dünyasında bir standart olan m68k çeviricilerinin sözdizimine benzemektedir. Bu sözdizimi Intel'in söz diziminden ne daha iyi ne de daha kötüdür. Alıştıktan sonra, Intel sözdiziminden daha sıradan bulursunuz, hatta biraz daha sıkıcı.

GAS sözdizimi ile ilgili temel uyarılar:

- Yazmaç isimleri `%` ile başlar, bunun için de eax, dl, v.b isimler yerine, yazmaçlar `%eax, %dl`, v.b. şekilde isimlendirilir. Bu da, harici C sembollerinin, herhangi bir karıştırma riski olmadan veya herhangi bir çirkin görünümlü alt çizgi kullanma ihtiyacı olmadan, sembolik makina kodu içerisine dahil edilmesine olanak sağlar.
- Intel'deki geleneksel önce hedef sonra kaynak kuralının tersine, terimlerin sırası kaynak–önce/hedef–sonra şeklindedir. Böylece, Intel'de `mov eax,edx` (`edx` yazmacının içeriğini `eax` içerisine taşı) şeklindeki bir sözdizimi GAS'da `mov %edx,%eax` haline gelecektir.
- Terim genişliği, komut adına bir sonek olarak belirtilir. Sonek olarak kullanılan `b` (8 bit) bayt için, `w` (16 bit) kelime (word) için ve `l` (32 bit) long içindir. Mesela, yukarıdaki ifade için doğru sözdizimi `movl %edx,%eax` şeklinde olacaktır. Yine de, **`gas`** katı bir AT&T sözdizimi gerektirmemektedir, bunun için de yazmaç terimlerinin genişliği tahmin edilebildiğinde sonek isteğe bağlıdır, aksi durumlarda öntanımlı olarak 32 bittir (bir uyarı ile birlikte).
- Anlık değerler önlerine bir `$` konarak belirtilir, `addl $5,%eax` örneğinde olduğu gibi (uzun tamsayı olarak anlık 5 değerini `%eax` yazmacına ekle).
- Terim öneklerinin olmaması, terimin bir bellek içeriği olduğu anlamına gelmektedir; burada `movl $foo,%eax` foo'nun *adresini* %eax'e koyarken, `movl foo,%eax` foo'nun *içeriğini* `%eax` yazmacına koymaktadır.
- Sıralama veya yönlendirme, referans yazmacını veya yönlendirme bellek hücresi adresini parantez içerisine almakla sağlanır, `testb $0x80,17(%ebp)`'de olduğu gibi (`%ebp` ile işaret edilen hücreden 17 birim uzaklıktaki baytın en yüksek anlamlı bitini sına).

Not: AT&T ve Intel çevirici sözdizimleri arasında kaynak kodu dönüşümü için *birkaç programın* (sayfa: 27) size yardımı dokunabilir, bazıları her iki yönde de dönüşüm gerçekleştirebilmektedir.

GAS'ın TeXinfo biçiminde, en azından kaynak dağıtımıyla gelen, kapsamlı bir belgelendirmesi vardır. `.info` sayfalarını Emacs veya herhangi birşeyle gözden geçirin. GAS kaynak paketi içerisinde `gas.doc` veya `as.doc` isimli bir dosya olsa gerek, fakat TeXinfo belgeleri arasına gömülmüştür. Elbette, herhangi bir şüphe durumunda en iyi belgelendirme kaynağın kendisidir! Sizi özellikle ilgilendiren kısım `Machine Dependencies::i386-Dependent::`'dır.

Yine, Linux kaynağı (işletim sistemi çekirdeği) mükemmel bir örnek olarak gelmektedir; `linux/arch/i386/` altındaki şu dosyalara bakınız: `kernel/*.S, boot/compressed/*.S`, `math-emu/*.S`.

Eğer bir çeşit dil, bir evre (thread) paketi, v.b. yazıyorsanız diğer dillerin (OCaml[B32], Gforth[B33], v.b.) veya evre kütüphanelerinin (QuickThreads, MIT pthreads, LinuxThreads, v.b.) bu işleri nasıl yaptıklarına bakabilirsiniz.

Son olarak, bir C kodunu sembolik makina diline derlemek de aradığınız komut çeşitleriyle ilgili sözdizimini size gösterebilir. Yukarıdaki *Sembolik makina diline ihtiyacınız var mı?* (sayfa: 6) kısmına bakınız.

### 3.2.3. Intel sözdizimi

İyi haber şu ki 2.10 sürümünden itibaren, GAS Intel sözdizimini de desteklemektedir. `.intel_syntax` komutu ile bu kipe geçiş sağlanabilir. Ne yazık ki, resmi binutils kılavuzunda bu belgelenmemiştir (henüz?), dolayısıyle

kullanmak istiyorsanız, binutils'in 2.11 AMD64 tabanlı sürümünü, http://www.lxhp.in-«berlin.de/lhpas86.html adresinden inceleyiniz.

### 3.2.4. 16 bitlik kip

Binutils (2.9.1.0.25+) i386 kişisel bilgisayarlarda tam olarak 16 bitlik kipi desteklemektedir (yazmaçlar *ve* adreslemeler). `.code16` ve `.code32`'yi çevirme kipleri arasında geçiş için kullanın.

Ayrıca, pekçok kimse (işletim sistemine kitaplık (kit) hazırlayan yazarlar da dahil) tarafından kullanılan zeki hilelerden birisi de GCC'yi 16 bitlik gerçek kip için kod üretmeye zorlamaktır, bu da `asm(".code16\n")` şeklinde bir satıriçi ifade ile sağlanır. GCC halen 32 bitlik adresleme kipleri üretirken, GAS onlar için uygun 32 bitlik önekleri ekleyecektir.

### 3.2.5. Makro desteği

AS, texinfo belgelerinde de belirtildiği gibi makro desteği de barındırmaktadır. Üstelik, GCC `.s` dosyalarını GAS'a göndermek için işlenmemiş birer sembolik makina kodu olarak algılarken, aynı zamanda `.S` dosyalarını GAS'a göndermeden önce CPP süzgecinden de geçirmeyi algılıyabilmektedir. Tekrar tekrar, örnekler için Linux kaynaklarına bakınız.

GAS'ın ayrıca GASP önişlemcisi de vardır, ki bu da makrosembolik makine dili hilelerinin GAS'a dahil olmasını sağlamaktadır. GASP, GAS binutils paketi içerisinde gelmektedir. *CPP* (sayfa: 17) ve *M4* (sayfa: 18) gibi filtre olarak çalışmaktadır. Ayrıntılar hakkında hiç bilgim yok, fakat kendi texinfo dosyasıyla beraber gelmektedir, gözatmak (**`info gasp`**), çıktısını almak isteyebilirsiniz. GAS ve GASP bana sıradan makro–çeviriciler gibi gelmektedir.

## 3.3. NASM

Netwide Assembler Projesi, C ile yazılmış, bilinen tüm sözdizimlerine ve nesne biçimlerine uyması gereken, hoş bir i386 çeviricisi sunmaktadır.

### 3.3.1. NASM'ı nereden bulurum

http://nasm.sourceforge.net, http://sourceforge.net/projects/nasm/ adreslerinde bulabilirsiniz.

Kullandığınız olağan yansınızdaki ikilik sürümü `devel/lang/asm/` dizini altındadır. Aynı zamanda `.rpm` ve `.deb` biçiminde Linux dağıtımlarının 'contrib' kısmında bulunması lazım.

### 3.3.2. Ne yapar

Söz dizimi Intel sözdizimidir. Kapsamlı makro işleme desteği eklenmiştir.

Desteklenen nesne dosyaları `bin`, `aout`, `coff`, `elf`, `as86`, `obj` (DOS), `win32`, `rdf` (onların kendi biçimi).

NASM, özgür LCC derleyicisi için bir arka uç (backend) olarak kullanılabilir (destek dosyaları eklenmiştir).

16 bitlik derleyici olarak BCC kullanmadığınız sürece (ki bu da bu 32 bitlik NASIL belgesinin kapsamı dışındadır), AS86 veya MASM yerine kesinlikle NASM kullanmalısınız, çünkü her platformda çalışır.

**Bilgi**

NASM, NDISASM isimli bir tersine çevirici (disassembler) ile beraber gelmektedir.

El yazım çözümleyicisi onu GAS'tan daha hızlı hale getirmektedir, buna rağmen elbette ki, 3 bazilyon (1 zilyondan büyük oldukça büyük bir sayı) değişik mimariyi desteklememektedir. Eğer GAS sözdiziminin aksine, Intel tarzı sözdizimini seviyorsanız, bu tercih edeceğiniz çevirici olacaktır.

Not: AT&T ve Intel çeviricileri arasında kaynak kod dönüşümünü sağlayan *çok az program* (sayfa: 27) bulunmaktadır; bazıları her iki yönde de çeviri yapabilmektedir.

## 3.4. Diğer Sembolik Makina Çeviricileri

Değişik ve göze çarpan özellikleriyle ilginizi çekebilecek başka sembolik makina çeviricileri de bulunmaktadır.

**Bilgi**

Klasik olmayan/Yüksek seviyeli gibi değişik geliştirilme seviyelerinde olabilirler.

### 3.4.1. AS86

AS86 makro desteğine sahip 16 bit ve 32 bitlik çalışma kipleri olan 80*86 çeviricisidir. Intel sözdizimine sahiptir fakat adresleme kiplerinde ufak farklılıkları vardır. Bir süre önce Linux çekirdeğinin de içinde bulunduğu pekçok projede kullanılıyordu. Daha sonra bu projeler GAS ve NASM içerisinde kullanılmaya başlandı. Bildiğim kadarıyla ELKS halen kullanmaya devam etmektedir.

AS86 `http://www.cix.co.uk/~mayday/` adresinde bin86 paketi içerisinde bir bağlayıcı ile (ld86) veya ayrı bir dosya olarak bulunabilir. Kılavuz sayfalarında ve kaynak paketteki asm.doc içerisinde belgelendirmesi bulunmaktadır. Şüpheye düştüğünüzde kaynağın kendisi iyi bir kılavuzdur: İyi şekilde açıklama satırları olmasa da programlama şekli oldukça doğrudur. ELKS, LILO veya Tunes 0.0.0.25 içerisinde AS86 nasıl kullanıldığına bakmak isteyebilirsiniz.

**Bilgi**

Çekirdeğin 2.4 sürümünden önce tamamen modası geçmiş bir sürümü, HJLu tarafından, bin86 adı ile, şu an herhangi bir Linux GCC deposunda bulunabilir, sadece Linux çekirdeğini derlemek için dağıtılıyordu. Fakat hiç kimseye bunu Linux çekirdeğini derlemek dışında başka bir şeyde kullanmasını önermem. Bu sürüm sadece elden geçirilmiş bir minix nesne dosyası içermektedir ve bu da GNU binutils ya da başka herhangi bir şey tarafından desteklenmediği gibi 32 bitlik kipinde bazı hatalar vardır, dolayısiyle onu sadece Linux çekirdeğini derlemekte kullanırsanız daha iyi olur.

#### 3.4.1.1. Belgeleri nerden bulurum

Kaynak paketteki kılavuz sayfalarına ve `as.doc` belgesine bakınız. Şüpheye düşüldüğünde, kaynağın kendisi en iyi belgedir: yorum satırları iyi yazılmamıştır, fakat yazılım geliştirme tarzı çok açıklayıcıdır. as86'nın ELKS, LILO veya Tunes 0.0.0.25... uygulamalarında nasıl kullanıldığına bakabilirsiniz.

#### 3.4.1.2. AS86'nın BCC ile kullanımı

`.s` asm'yi a.out `.o` nesne ve `.l` listeleme dosyasına dönüştürmek için BCC kullanımıyla ilgili GNU Makefile dosya satırı şöyledir:

```
%.o %.l:        %.s
        bcc -3 -G -c -A-d -A-l -A$*.l -o $*.o $<
```

Eğer herhangi bir listeleme istemezseniz, `%.l`, `-A-l` ve `-A$*.l` kısımlarını kaldırın. a.out dışında başka bir şey isterseniz, BCC belgelerini diğer desteklenen dillerle ilgili olarak inceleyebilirsiniz ve/veya GNU binutils **`objcopy`** uygulamasını kullanabilirsiniz.

### 3.4.2. YASM

YASM, NASM çeviricisinin tamamen yeniden GNU GPL (bazı kısımları "yeni" BSD Lisansı altındadır) altında tekrar yazılmış halidir. Temelden itibaren çoklu söz dizimine (örn. NASM, TASM, GAS, v.b.) ve çoklu çıktı nesne biçimlerine izin vermek için tasarlanmıştır. Tüm tasarım içerisindeki önemli modüllerden biri de eniyileme gerçekleştiren modüldür.

Umut verici görünmektedir; yoğun şekilde bir gelişim sürecindedir, siz de bunda yer almak isteyebilirsiniz. http://www.tortall.net/projects/yasm/ adresine bakınız.

### 3.4.3. FASM

FASM (düz çevirici–flat assembler) 'düz gerçek modda' ('flat real mode') çalışan, verimli ve hızlı bir 80x86 çeviricisidir. Diğer pekçok 80x86 çeviricisinin tersine, ihtiyaç duyduğu bilgiyi eklemek için sadece kaynak koda gereksinim duyar. Kendi diliyle yazılmıştır, çok küçük ve hızlıdır. DOS/Windows/Linux altında çalışabilir, DOS EXE, Win32 PE ve COFF çıktıları gibi, düz ikilikler üretir (flat binary). http://fasm.sourceforge.net/ adresine bakınız.

### 3.4.4. OSIMPA (SHASM)

osimpa, 80x86 ve sonraki kuşakları için, tamamen GNU Bash komut yorumlama kabuğunda yazılmış olan bir çeviricidir. osimpa'nın atası shasm idi. osimpa daha iyi ayıklanmış, faydalı Linux ELF çalıştırılabilir dosyaları üretebilir ve pekçok HLL–benzeri eklentisi ve programcıya uygun komutları vardır.

Elbetteki, diğer çeviricilerden daha yavaştır. Kendi söz dizimi vardır (x86 opkodları için kendi isimlerini kullanır). Oldukça iyi bir belgeleleme eklenmiştir. ftp://linux01.gwdg.de/pub/cLIeNUX/interim/ adresine bir göz atın. Muhtemelen sıradan işler için kullanmayacaksınız, fakat en azından ilginç bir alan olarak ilginizi hak etmektedir.

### 3.4.5. AASM

Aasm değişik hedef mimarileri desteklemek için tasarlanmış gelişmiş bir çeviricidir. Kolayca genişletilebilmesi için tasarlanmıştır ve her bir işlemci ve ikilik dosya biçimi için yapılan yekpare çevirici tasarımım için bir alternatif olarak düşünülmelidir.

Aasm sunduğu sembol alanı, ifade makinası, büyük tamsayı desteği, makro yeteneği ve çok sayıdaki ve doğru uyarı mesajları gibi ileri özellikleri ile sembolik makina programlamayı kolay hale getirmelidir. Değişken modüler yapısı dinamik kütüphanelerin eklenmesine imkan sağlamaktadır.

Girdi modulü nasm, tasm, masm, v.b gibi söz dizimlerini destekler. x86 çevirici modulü MMX, SSE ve 3DNow'ı da içeren P6'ya kadar uzanan opkod uzantılarını destekler. F–CPU ve SPARC çevirici modulleri geliştirilme aşamasındadır.

http://savannah.nongnu.org/projects/aasm/

### 3.4.6. TDASM

Table Driven Assembler (TDASM) herhangi bir sembolik makina dili için *özgür*, taşınabilir, platformlar arasında uygunluk gösteren bir çeviricidir. Derleme işlemini tanımlayan bir tablo kullanılarak herhangi bir hedef işlemci için onu bir derleyici olarak kullanmak mümkün olmalıdır.

http://www.penguin.cz/~niki/tdasm/ adresinden temin edilebilir.

### 3.4.7. HLA

HLA[B47] Yüksek Seviyeli Sembolik Makina Dilidir (**H**igh **L**evel **A**ssembly). Değişken tanımlama, yordam tanımlama ve yordam çağrıları için yüksek seviyeli dile benzer (Pascal, C/C++ ve diğer yüksek seviyeli dillere benzer) bir sözdizimi kullanmaktadır. Standart makina dili komutları için değiştirilmiş bir sözdizimi kullanmaktadır. Aynı zamanda yüksek seviyeli dile özgü denetim yapıları da (`if, while, repeat..until`, v.b) sağlayarak, daha okunabilir kodlar yazılmasını sağlamaktadır.

HLA özgürdür ve kaynak kodlarıyla beraber gelmektedir, Linux ve Win32 sürümleri de mevcuttur. Win32 için MASM'a ve de MS–link'in Win32 de çalışan 32 bitlik sürümüne ihtiyacınız vardır, Linux üzerinde GAS'a ihtiyacınız vardır, çünkü HLA özelleştirilmiş sembolik makina kodu üretir ve en son aşamadaki çevirme ve bağlama işlemlerinde bu çeviriciyi kullanır.

### 3.4.8. TALC

TALC[B48] bir diğer MASM/Win32 tabanlı derleyicidir (yine de ELF çıktısını desteklemektedir, değil mi?).

TAL, **T**yped **A**ssembly **L**anguage sözcüklerinin kısaltmasıdır. Yazım açıklamalı geleneksel türsüz sembolik makina dilini, bellek yönetim ilkellerini ve yazım kurallarının bir ses kümesini içermesi yanında bellek güveliğini sağlama, akış güvenliği denetimi ve TAl programlarının yazım güvenliği konusunda zenginleştirilmiştir. Bunun yanında yazma yapıları, kayıtlar ve yapılar içeren, diziler, polimorfik ve yüksek seviyeden işlevler içeren, olağandışılıklar, soyut veri yapıları, altyazımlar ve modüller içeren kaynak programlama dili özelliklerini çözümleyecek kadar, yeterince açıklayıcıdır. TAL düşük seviyeli derleyici eniyilemelerini kabul etme esnekliğiyle tek başına önemli bir yere sahiptir. Sonuç olarak, TAL tür yönlendirmeli derleyiciler, taşınabilir kod uygulamaları içerisinde doğrulanmış güvenli kod veya genişleyebilir işletim sistemi çekirdekleri üretmek isteyen biri için ideal bir platformdur.

### 3.4.9. Özgür Pascal (Free Pascal)

Free Pascal[B49] dahili bir 32 bitlik çeviriciye (NASM tablolarına dayanan) sahiptir ve seçeneklerle ayarlanabilen çıktıları şunlardır:

- İkilik (çapraz olarak .o derlendiğinde ELF ve coff) çıktı
- NASM
- MASM
- TASM
- AS (aout,coff, elf32)

MASM ve TASM çıktılarının hataları diğer ikisi kadar güzel ayıklanmamıştır, fakat bu da bazen yararlı olabilir.

Çeviricinin görünüşü ve kullanımı Turbo Pascal'ın dahili BASM'ına dayanmaktadır, IDE benzeri renklendirmeleri desteklemektedir, FPC'de tam olarak gcc ile içiçe geçebilir (C düzeyinde, C++ değil).

Aptal bir RTL kullanarak bile, birisi saf sembolik makina kodu üretebilir.

### 3.4.10. Win32Forth derleyicisi

Win32Forth, Win32s, Win95, Win/NT altında çalışan, *özgür* bir 32–bit ANS FORTH sistemidir. FORTH dilini yansıtacak 32 bitlik özgür bir derleyici (önekli veya sonekli bir sözdizimine sahip olabilir) barındırır. Makro işleme FORTH dilinin gücüyle gerçeklenir; yine de, girdi ve çıktı içeriğini tek destekleyen Win32For'un kendisidir (`.obj` dosyalarının dökümlenmesi yoktur, fakat bu özelliği ekleyebilirsiniz). `ftp://ftp.forth.org/pub/Forth/Compilers/native/windows/Win32For/` adresine bakınız.

### 3.4.11. Terse

TERSE[B51], x86 ailesi için en öz sembolik makine dili sözdizimini sağlayan yazılım geliştirme aracıdır! Yine de şeytani bir yazılımdır. Bir yerlerde, orjinal yazarın sözdizimine sahip çıktığı, değersiz hileler içerdiğinden sonradan ortadan kaldırılan, özgür bir kopyasının olduğu söylenir. Eğer sembolik makina dili korsanlığıyla ilgili kendinize ilgi çekici bir proje arıyorsanız, sizi NASM için bir terse–sözdizimi ucu geliştirmeye davet ediyorum, tabii eğer o sözdizimini severseniz.

comp.compilers[B52]'da tarihi bir not:

```
999/07/11 19:36:5, moderator şöyle yazmış:

"Çeviricilerin oldukça kötü bir söz diziminin olması zorunluluğunun
bir sebebi yok. Yaklaşık 30 yıl önce, Niklaus Wirth'in PL360'ını
kullandım, temelde S/360 çeviricisi olan, Algol sözdizimine sahip ve
döngülerin açık dallanmalara dönüşmesinden dolayı biraz sentetik şeker
gibi bir çeviriciydi. Aslında gerçekten de bir çeviriciydi, örn.
ifadenizi yazmaçlara gidecek açık değer atamalarıyla yazmalıydınız,
fakat oldukça hoştu. Gene de Algol W ile, Algol'un hızlı bir alt kümesi,
Pascal'ın atası ile yazmaya değerdi. Her zaman söylendiği gibi,
atalarına nazaran önemli bir gelişmeydi. -John"
```

### 3.4.12. Özgür olmayan ve/veya 32–bit olmayan x86 çeviricileri

Bunların pekçoğunu, x86 sembolik makina dili programlamanın temelleriyle beraber, *Raymond Moon'un x86 sembolik makina dili SSS* (sayfa: 27)'inde bulabilirsiniz.

DOS tabanlı tüm çeviricilerin, Linux DOS emülatöründe, aynı zamanda benzer diğer emüilatörlerde çalışması gerektiğini unutmayın, dolayısiyle eğer elinizde bir tane varsa, onu hala gerçek bir işletim sistemi içerisinde kullanabilirsiniz. DOS tabanlı çeviriciler, GNU BFD kütüphanesi tarafından desteklenen, COFF ve/veya diğer nesne dosyalarını destekler, dolayısiyle bunları kendi özgür 32 bitlik araçlarınızla kullanabilirsiniz ve belki de GNU **objcopy**'i de (binutils'in bir parçası) bir dönüştürme aracı olarak kullanabilirsiniz.

# 4. Düşük seviye programlama

Sembolik makina dili ile programlama, programın kritik kısımları için bir can sıkıntısıdır.

Doğru görev için uygun araçları kullanmalısınız, dolayısiyle eğer uygun değilse sembolik makina dilini kullanmak için seçmeyin; pekçok durumda C, OCaml, Perl, Scheme daha iyi birer seçim olabilir.

Yine de, makina üzerinde bu araçların makina üzerinde yeterince iyi olanaklar sunmadığı anlar vardır ve sembolik makina dili böylesi durumlar için faydalı veya gereklidir. Böyle bir durumda, sonsuz kere kullanılabilir tanımlamalar içerisine yerleştirilmiş tekrar eden kalıplara izin veren makroprogramlama ve metaprogramlama sistemini takdir edeceksiniz, ki bu sistem aynı zamanda, daha güvenli programlama, model değişiminin otomatik üretimi, vs. sağlamaktadır. Yalın çeviriciler genelde yetersizdir, hatta sadece C ile ilintilenecek küçük yordamlar yaparken bile.

## 4.1. Harici Filtreler

Sizin çeviricinizde her ne makro desteği varsa veya her ne dilini (hatta C!) kullanıyorsanız, eğer dil yeterince sizin için ifadesel değilse, bir Makefile kuralı ile dosyalarınızı harici bir filtreden şu şekilde geçirebilirsiniz:

```
%.s:    %.S other_dependencies
        $(FILTER) $(FILTER_OPTIONS) < $< > $@
```

### 4.1.1. CPP

CPP gerçekten çok anlamlı, ifadesel değildir, fakat kolay şeyler için yeterlidir ve GCC tarafından şeffaf şekilde çağrılır.

Kısıtlamalarına bir örnek olarak, yokediciler de (destructors), bildirim bloğunun sonunda çağrıldıklarından içinde nesne bildirimleri yapamazsınız; bölümlere veya etki alanlarına, vs. sahip olamazsınız.

CPP herhangi bir C derleyicisi ile birlikte gelir. Yine de, ne kadar vasat olduğunu düşünürsek, eğer onu C'siz kullanmayı başarabilirseniz de, ondan uzak durun.

### 4.1.2. M4

M4 size, Turing dengi bir dil, yinelemeler (recursions), düzenli ifadeler (regular expression) ile, macro işlemenin tüm gücünü vermektedir. CPP'nin yapamadığı herşeyi onunla yapabilirsiniz.

M4 kullanarak yapılan ileri düzey programlama örnekleri için macro4th (this4th)[B54] veya Tunes 0.0.0.25 kaynaklarına[B55] bakınız.

Yine de, işlevsel olmayan tırnak kullanımı (quoting) ve bunun kaldırılması (disquoting) şeklindeki anlamsal yapı (semantics), eğer *ileri* düzey makro programlama yapmak istiyorsanız sizi süreklilik gösteren içiçe makro tarzı (continuation–passing tail–recursive macro style) kullanmaya zorlamaktadır (ki bu da insana TeX'i hatırlatıyor – herneyse, TeX'i yazı yazmak dışında bir makroişlemci olarak kullanan kimse var mı?). Tırnak ve yineleme (recursion) kullanımına hiç izin vermeyen CPP'den daha kötü değildir.

M4'ü kullanmak için doğru sürüm, en çok özelliği, en az hatası ve kısıtlamaları olan `GNU m4 1.4`'tür (varsa daha sonraki sürümleri). m4 herhangi bir şey için yavaş çalışacak şekilde tasarlanmıştır fakat en kolay kullanıma sahiptir, ki bu da çoğu sembolik makina dili için makul bir durumdur (milyonlarca satır sembolik makina kodu yazmıyorsunuz değil mi?).

### 4.1.3. Kendi filtreleriniz ile makroprogramlama

Bilindik araçlarla kendi basit makro genişleme filtrelerinizi yazabilirsiniz: perl, awk, sed, v.b. Biraz daha hızlı gerçeklenebilir ve herşeyi denetmersiniz. Fakat, elbetteki, makroişlemede güç, "zor yol" anlamına gelmektedir.

## 4.2. Metaprogramlama

Makroları genişleten harici bir filtre kullanmak yerine, işleri yapmanın bir yolu da tüm diğer programların bir kısmını ya da tamamını yazan programlar yazmaktır.

Örneğin, çıktı olarak aşağıdakileri gerçeklemek için kaynak kod üreten bir program kullanabilirsiniz:

- sinüs/kosinüs/vs. arama tablolarını üretmek için,
- bir ikilik dosyanın kaynak kodu gösterimini elde etmek için,
- hızlı ekran yordamlarına biteşlemleri derlemek için,
- normal kaynak koddan başka, belgelendirme, başlangıç/bitiş kodları, tanımlama tablolarını elde etmek için,
- perl/shell/scheme betiğinden üretilen rastgele işlem yapan özelleştirilmiş sembolik makina koduna sahip olmak için,
- çapraz başvuru tabloları ve kod bölümlerinden sadece tek noktada tanımlı veri etkileşimi sağlamak için,
- v.b.

Bunun hakkında düşünün!

### 4.2.1. Derleyicilerdeki arka uçlar

GCC, SML/NJ gibi derleyiciler, Objective CAML (OCAML), MIT–Scheme, CMUCL, v.b.'nin, kullanmayı tercih edebileceğiniz, kendi dahili çevirici arkaçları (backend) vardır, eğer yarı otomatik olarak bu dillere göre ya da elden geçirdiğiniz dillerden kod üretmeye niyetlenirseniz: uzun sembolik makina kodları yazmak yerine, bir derleyiciyi değiştirebilirsiniz, böylece uzun sembolik makina kodu çöpe gider!

### 4.2.2. New–Jersey makina kodu araç seti

Sembolik makina kodunu değiştirecek bir kod üretim tabanı oluşturmak için Icon programlama dilini kullanan (deneme açamasındaki ML sürümü ile) bir proje vardır. `http://www.eecs.harvard.edu/~nr/toolkit/` adresine bakınız.

### 4.2.3. TUNES

Özgür Fikirli Bilgisayar Sistemi (Free Reflective Computing System) için TUNES Projesi[B57], kendi gelişim sürecinin bir parçası şeklinde, Scheme dilinin bir uzantısı olarak kendi çeviricisini geliştirmektedir. Henüz hiç çalışmamıştır, fakat yardımlar kabul edilmektedir.

Çevirici soyut sözdizim ağaçlarını (abstract syntax trees) değiştirmektedir, böylece sembolik makina sözdizimi çeviricisi, tersine çevirici (disassembler), genel sembolik makina dili/derleyici arka ucu, v.b. tabanı olarak eşit şekilde hizmet verebilmektedir. Aynı zamanda, gerçek bir dilin, Scheme, tüm gücü, makroprogramlama ve metaprogramlama için onu karşı konulmaz hale getirmektedir.

# 5. Çağrı Uzlaşımları

## 5.1. Linux

### 5.1.1. GCC'ye ilintileme

C–asm karışımı projeler üretiyorsanız bu tercih edilen yoldur. **GAS** açıklamaları olan Linux çekirdeği **.s** dosyalarından örnekleri ve GCC belgelerini inceleyiniz (**as86** ile ilgili olanları).

32 bitlik geridönüş adresi üzerinde, 32 bitlik bağımsız değişkenler yığıta (stack) ters sözdizimsel sırada itilirler (push) (böylece doğru sırada erişilir/çıkarılırlar (pop)). `%ebp`, `%esi`, `%edi` ve `%ebx` çağrılan tarafından kullanılır, diğer yazmaçlar ise çağırıcı tarafından kullanılır; `%eax` sonuçları tutmak içindir veya `%edx:%eax` 64 bitlik sonuçlar için kullanılır.

FP yığıtı (FP stack): Emin değilim ama, sanırım tüm sonuç, saklanmış çağrıcıların tamamı `st(0)` içindeydi. Eğer daha fazla ayrıntı istiyorsanız `http://www.caldera.com/developer/devspecs/` adresindeki SVR4 i386 ABI belirtimi iyi bir başvuru kaynağıdır.

GCC'nin çağrı uzlaşımlarını değiştirmek için, yazmaçları önceden ayırtan, yazmaçlarda bağımsız değişkenler barındırılmasını sağlayan, FPU'yu dikkate almayan, v.b. seçenekleri olduğunu unutmayın. i386 `.info` sayfalarına bakınız.

Standart GCC çağrı uzlaşımlarını izleyecek bir işlev için `cdecl` ve `regparm(0)` bildirimlerini yapmanız gerektiğine dikkat edin. GCC info sayfasından `C Extensions::Extended Asm::` kısmına bakınız. Aynı zamanda Linux'un kendi `asmlinkage` makrosunu da nasıl tanımladığına bakınız.

### 5.1.2. ELF ve a.out arasındaki sorunlar

Bazı C derleyicileri, diğerleri yapmadığı halde, her sembolden önce bir alt çizgi yerleştirirler.

Özellikle, Linux a.out GCC böylesi bir ön yerleştirmeyi yapar, oysa ki Linux ELF GCC yapmamaktadır.

Eğer her iki davranış biçimiyle de aynı anda uğraşmak isterseniz, varolan paketlerin bunu nasıl yaptıklarına bakınız. Mesela, Elk, qthreads, OCaml... gibi eski bir Linux kaynak ağacı edinin.

Ayrıca, örneğin, `foo()` işlev çağrısının gerçekte bir makine dili kodu olan `bar`'ın çağrılmasını sağlaması için şu şekilde bir ek ifade ile örtük C–>asm isim değişikliğini zorlayabilirsiniz:

```
void foo asm("bar") (void);
```

Binutils paketindeki **objcopy** uygulamasının a.out nesnelerinizi ELF nesnelerine dönüştürmeyi mümkün kılması gerektiğini unutmayın, hatta bazı durumlarda tam tersine de imkan tanır. Daha genel olarak, pekçok dosya biçimi arasında dönüşüm gerçekleştirir.

### 5.1.3. Doğrudan Linux sistem çağrıları (syscalls)

Genelde **C kütüphanesi** (`libc`) kullanmanın tek yol olduğu ve doğrudan sistem çağrıları yapmanın kötü olduğu söylenir. Bu doğrudur. Bir bakıma... Genel olarak, **libc** kütüphanesinin kutsal olmadığını bilmelisiniz ve *pekçok* durumda sadece bazı denetimler yapar, sonra çekirdeğe çağrı yapar ve ardından errno'ya atama yapar. Bunu kendi programınızda da yapabilirsiniz (eğer ihtiyacınız varsa) ve programınız bir düzine kat daha küçük olacaktır, bu da gelişmiş bir başarım artışına sebep olacaktır, bu da sırf paylaşımlı kütüphaneleri kullanmadığınızdan kaynaklanacaktır (durağan (static) kütüphaneler daha hızlıdır). Sembolik makina dili ile programlamada **libc** kullanımı pratik birşeyden çok zevk/inanış meselesidir. Linux'un POSIX standartlarına uygun olmayı hedeflediğini unutmayın, benzer şekilde **libc** de. Bu da, hemen her **libc** "sistem çağrısı" sözdizimin gerçek çekirdek sistemi çağrılarındaki sözdizimiyle örtüşmesi anlamına gelir (ve tam tersi). Buna ek olarak, **GNU libc (glibc)** sürümden sürüme daha yavaş hale gelmekte ve daha çok bellek tüketmektedir. İlerde siz de kendi, değişik türlerde, **libc**'ye özel işlevlerinizi (sadece birer sistem çağrısı değil) tanımlayacaksınız (`printf()` ve şürekası)... Ve buna hazırsınız, değil mi? `:)`

Doğrudan sistem çağrıları yapmanın artı ve eksileri şu şekilde özetlenebilir:

#### Artılar

- olası en küçük boyut; son baytı sistem dışında bırakmak
- olası en yüksek hız; favori karşılaştırmalı değerlendirme (benchmark) dışı döngüleri bunun dışında bırakmak
- tam denetim: program/kütüphanenizi size özgü dile veya bellek gereksinimlerine veya herhangi bir şeye uydurabilirsiniz.
- libc çerçöplerinin yol açacağı bir kirlilik olmaz
- C çağrı uzlaşımlarının yol açacağı bir kirlilik olmaz (eğer kendi dilinizi veya ortamınızı tasarlıyorsanız)
- durağan kütüphaneler libc yükseltmelerinden ve çökmelerinden veya yorumlayıcıya `#!` yolu ile asılmanızdan sizi bağımsız kılar. (ve daha hızlıdır)
- biraz da eğlence içindir (sembolik makina dili dışında heyecanlanmaz mısınız?)

#### Eksiler

- Eğer bilgisayarınızda bir başka program da libc kullanıyorsa, libc kodunun yinelenmesi otomatik olarak belleğin, korunması yerine, boşa harcanmasına sebep olacaktır.
- Pekçok durağan ikilikte (static binary) gereksiz yere tanımlanan servisler bellek israfıdır. Fakat kendi libc yerdeğiştirmenizin bir paylaşımlı kütüphane olmasını sağlayabilirsiniz. (NBB: Bu yukarıdaki iddiasını yalanlamıyor mu? `;-)`)

- Herşeyi sembolik makina dili ile yazmak yerine, bir çeşit bayt kodu, sözcük kodu veya yapısal yorumlayıcıya sahip olmakla, boyut çok daha iyi korunur. (derleyicinin kendisi C veya sembolik makina dilinde yazılabilir). İkilik çeşitliliğini küçük tutmanın en iyi yolu, çoklu ikilikler yapmamaktır, yerine `#!` önekiyle başlayan yorumlanan işlem dosyaları kullanmaktır. Bu, OCaml'ın sözcük kodu kipinde çalıştığındaki durumdur (eniyilenmiş doğal kod kipine karşın) ve de libc kullanımıyla uyumludur. Bu aynı zamanda unix araçlarının yeniden gerçeklenimi olan Tom Christiansen'in Perl Güç Araçları (Perl PowerTools)'nın nasıl çalıştığının açıklamasıdır. Son olarak, bunları küçük tutmanın bir yolu da, tam olarak yolu kodlanmış harici bir dosyaya bağımlı olmamaktır, bu da kütüphane veya yorumlayıcı olsun, tek bir ikilik dosyaya sahip olmak ve buna sabit veya sembolik bağlar yapmaktır: aynı ikilik size en makul alanda, alt yordamların gereksiz kullanımı veya gereksiz ikilik başlıkları olmadan, ihtiyacınız olan herşeyi sunacaktır; kendine özel davranışı *argv[0]* değerine bakarak yönlendirecektir; bu durumda tanınan ismiyle çağrılmaz, bir kabuğa öntanımlı olabilir ve muhtemelen bir yorumlayıcı olarak da kullanışlı olmuş olur!

- Nadir linux sistem çağrıları yanında libc'nin sunduğu pekçok işlevsellikten faydalanamazsınız: malloc, thread, locale, password, yüksek–seviyeli ağ yönetimi, v.b. işlevsellikler, kılavuz sayfalarının 2. bölümünde değil, 3. bölümünde yer alır.

- Bu yüzden, libc'nin, `printf()`'den `malloc()` ve `gethostbyname`'e uzanan çok sayıda parçalasını yeniden gerçeklemek zorunda kalabilirsiniz. libc varken bu gereksizdir, hatta *oldukça* sıkıcı olabilir. Bazılarının libc'nin bazı kısımları için "hafif" (ligth) yerdeğiştirmeler yazdıklalarına dikkat ediniz – bunları inceleyiniz! (Redhat'in minilibc'si, Rick Hohensee'nin libsys[B59]'si, Felix von Leitner'in dietlibc[B60]'si, Christian Fowelin'in libASM[B61]'si, asmutils[B62] projesi de tamamen saf sembolik makina libc'si ile çalışmaktadır)

- Durağan kütüphaneler sizi, libc güncellemelerinden ve aynı zamanda, gzip–sıkıştırılmış dosyalarını düzgün şekilde ihtiyaç olduğunca açmanızı sağlayan, **`zlib`** paketi gibi libc eklentilerinden faydalanmaktan alıkoyar.

- libc tarafından eklenen çok az sayıdaki komutun sistem çağrılarının maliyetine kıyasla *küçük* bir hız yükü olabilir. Eğer hız gözönüne alınırsa, temel sorununuz, onların sarmalayıcı işlevlerini değil, kendi sistem çağrılarınızın kullanımıdır.

- Kendi çağrı uzlaşımları olan ve standart uzlaşım kullanımında kural dönüşüm yüküne (high convention–translation overhead) büyük önem veren L4Linux gibi Linux'un mikro çekirdek sürümleri çalıştırılırken, sistem çağrıları için standart sembolik makina dili API'leri kullanmak, libc API'leri kullanmaktan daha yavaştır (L4Linux, sistem çağrı API'leri ile yeniden derlenmiş şekilde gelir; elbette kendi kodunuzu onun API'leriyle tekrar derleyebilirsiniz).

- Genel hız eniyileme konularıyla ilgili olarak önceki konulara bakınız.

- Eğer sistem çağrıları size göre çok yavaşsa, kullanıcı adasında kalmak yerine çekirdek kaynak kodlarını (C dilindeki) elden geçirmeyi isteyebilirsiniz.

Eğer yukarıdaki artı ve eksileri zihninizde iyice ölçüp tarttıysanız ve hala doğrudan sistem çağrılarını kullanmak istiyorsanız, size bazı önerilerim olacak:

- Sistem çağrı işlevlerinizi taşınabilir bir şekilde, C dilinde (sembolik makina dilinin taşınamaz özelliğine karşın) `asm/unistd.h` ile sunulan makroları kullanarak tanımlayabilirsiniz.

- Sistem çağrısı işlevlerini değiştirmeyi deneyecekseniz, libc'den kaynak kodunu alıp inceleyin. (Ve daha iyisini yapabileceğinizi düşünüyorsanız, bunu yazarlara bildirin!)

- İstediğiniz her işi yapan bir sembolik makina kodu için *Özkaynaklar* (sayfa: 27)ına bakınız.

Temelde, eax içerisine __NR_sistemçağrı_ismi numarası (asm/unistd.h dosyasındadırlar) ile int 0x80 değeri ve parametreleri de sırasıyla (*altı* (sayfa: 22)ya kadar) ebx, ecx, edx, esi, edi, *ebp* (sayfa: 22) içine konur.

Sonuç eax içerisinde döndürülür, negatif sonuçlarda hata ile döner, bunun karşılığı da libc içerisinde errno'dur. Kullanıcı yığıtına dokunulmaz, dolayısiyle bir sistem çağrısı yaparken geçerli bir kullanıcı yığıtına ihtiyacınız yoktur.

**Bilgi**

ebp'ye 6 parametre aktarımı Linux 2.4 sürümünde mümkün olmuştur, daha önceki Linux sürümleri yazmaçlarda sadece 5 parametreye bakıyordu.

Linux Çekirdeğinin Dahili Yapısı (Linux Kernel Internals)[B66] belgesi ve özellikle i386 Mimarisinde Sistem Çağrıları Nasıl Gerçeklenir? (How System Calls Are Implemented on i386 Architecture?)[B67] bölümü çok daha sağlıklı bilgi verecektir.

Başlatırken bir sürece parametrelerin aktarılmasında olduğu gibi, genel prensip, yığıtın orjinal olarak argüman sayısını (*argc*) ve ardından **argv*'ler halinde argüman göstericilerini, bundan sonra da *environ* (ortam) için boş gösterici ile sonlandırılmış boş karakter sonlandırmalı isim=deger dizgelerini içereceğidir. Daha ayrıntılı bilgi için, *Özkaynaklar* (sayfa: 27) bölümünü okuyunuz, libc'nizdeki C başlatma (crt0.S veya crt1.S) kodlarını veya bunların Linux çekirdeğinde olanlarını (exec.c ve linux/fs içindeki binfmt_*.c) inceleyiniz.

### 5.1.4. Linux altında donanımsal G/Ç

Eğer Linux altında doğrudan port erişimi ile G/Ç işlemleri gerçekleştirmek istiyorsanız, bu ya işletim sisteminde bir değişiklik gerektirmeyen basit bir iştir ve bununla ilgili G/Ç portları ve programlama (IO-Port-Programming) küçük nasıl belgesini okumanız yeterli olur ya da bir çekirdek aygıt sürücüsü gereklidir ve çekirdek kaynak kodlarını elden geçirme, aygıt sürücüsü geliştirme, çekirdek modülleri, v.b. ile ilgili daha fazla bilgi edinmeniz gerekir. Bunlarla ilgili pek çok belge ve NASILlar LDP sayfalarında bulunmaktadır.

Belki de, grafik programlama yapmak istersiniz, o zaman GGI[B69] veya XFree86[B70] projelerinden birine katılın.

Bazıları daha iyisini bile yapabilir, yorumlanmış belli bir alana özgü bir dilde, GAL, küçük ve güçlü XFree86 sürücüleri yazabilirler, bazı değerlendirmelerden sonra da C ile yazılmış sürücülerin verimini arttırabilirler (sürücüler ne sadece asm'dir ne de sadece C!). Burada sorun, verimi arttırmak için kullanılacak bazı değerlendiricilerin özgür olmamasıdır. Bunların özgür sürümlerini gerçekleştirecek olan var mı?

Herneyse, tüm bu durumlarda, herşeyi sembolik makina kodu ile yazmak yerine GCC satıriçi sembolik makina dilini linux/asm/*.h dosyalarındaki makrolarla kullanmak daha iyi olacaktır.

### 5.1.5. Linux/i386'daki 16 bitlik sürücülere erişim

Böyle bir şey teorik olarak doğrudur (kanıt: DOSEMU[B71]'nun programlara seçici bir şekilde port atamalarını nasıl yaptığını inceleyiniz) ve ben de bir yerlerde birilerinin bunu yaptığı söylentilerini de duydum (bir PCI sürücüsü mü? bir VESA erişim aracı mı? ISA PnP mi? bilmiyorum). Eğer bunun hakkında çok net bilginiz varsa, o zaman çok daha memnun olacaksınız. Herneyse, daha ayrıntılı bilgi için bakılması gereken kaynaklar Linux çekirdeğinin kaynak kodları, DOSEMU kaynakları (ve DOSEMU deposundaki diğer programlar) ve de Linux altında pekçok düşük seviyeli programın kaynaklarıdır ... (belki CGI'da olabilir, eğer VESA desteği varsa).

Temel olarak ya 16 bitlik korumalı kipi veya vm86 kipini kullanmalısınız.

İlkinin yapılandırılması daha basittir, fakat sadece segman aritmetiği veya mutlak segman adreslemesi (özellikle 0. segmanı adreslerken) ile ilgili işlemler yapmayacak iyi davranışlı kodla çalışır, fakat şans eseri tüm segmanlar kullanılırsa, LDT ile ileri düzey ayarlama yapılabilir.

İkincisi ise harcıalem 16 bitlik ortamlarla daha bir uyumluluk sağlar, fakat idaresi daha karmaşıktır.

Her iki durumda da 16 bitlik koda geçmeden önce, şunları yapmalısınız:

- 16 bitlik kod içerisinde kullanılan herhangi bir mutlak adresi (ROM, video tamponları, DMA hedefleri ve bellek eşlemli G/Ç gibi) `/dev/mem`'den sürecinizin adres uzayına `mmap`'leyin.
- LDT ve/veya vm86 kipi gözlemleyici ayarlayın
- çekirdekten uygun G/Ç izinlerini kapın (üst bölümlere bakınız)

Tekrar, DOSEMU projesiyle sunulan belgeleri dikkatlice okuyunuz, özellikle Linux/i386 altında ELKS ve/veya `.COM` programlarını çalıştırmak için kullanılan küçük emülatörlerle ilgili kısımları.

## 5.2. DOS ve Windows

Pekçok DOS çoğaltıcıları (extenders) DOS servisleri için bazı servislerle beraber gelir. Bununla ilgili belgeleri okuyunuz, fakat genelde, `int 0x21` ve benzerine benzetim yaparlar (simulate) ve siz de sanki gerçek kipteymiş gibi çalışırsınız (Az gelişmişlik dışında birşeyleri olduğundan ve işlemleri 32 bitlik terimlerle çalışır hale getirdiklerinden şüpheliyim, daha çok gelen kesmeleri gerçek kip veya vm86 eylemcisine yansıtıyor gibiler.)

DPMI hakkındaki belgeler (ve fazlası) `ftp://x2ftp.oulu.fi/pub/msdos/programming/` adresinde bulunabilir (yine, asıl x2ftp sitesi kapanıyor (kapandı?), onun için yansıyı[B73] kullanın).

DJGPP kendi (sınırlı) **`glibc`** türev/altküme/yerdeğiştirmeleri v.b.leri ile gelmektedir.

Linux'tan DOS'a çapraz–derleme (cross–compile) yapmak mümkündür, metalab.unc.edu için olan yerel FTP yansınızın `devel/msdos/` dizinine bakınız; Aynı zamanda Utah üniversitesindeki Flux Projesi[B74]'nden MOSS DOS–extender (DOS–genişletici)'ye de bakınız.

Diğer belgeler ve SSS, DOS merkezlidir; biz DOS gelişimini tavsiye etmiyoruz.

**Windows ve Şürekası**

Bu belge Windows programlama hakkında değildir, bununla ilgili pekçok belgeyi heryerde bulabilirsiniz... Bilmeniz gereken, GNU programlarının Win32 altında çalışması için, cygwin32.dll[B75] kütüphanesini olduğudur, böylece sizler GCC, GAS ve tüm GNU araçları ile pekçok diğer Unix uygulamasını kullanabilmektesiniz.

## 5.3. Kendi işletim sisteminiz

Denetim duygusu pekçok işletim sistemi geliştiricisini sembolik makina diline çeken şeydir, bu da genelde sembolik makina dili kodları elden geçirmeye yol açmakta veya ondan kaynaklanmaktadır. Her ne kadar temelini oluturan bir sistemin tepesinde çalışabiliyor olsa da (Mac üzerindeki Linux veya Unix üzerindeki OpenGenera), kendi kendine gelişime izin veren bir sistem ancak işletim sistemi olarak isimlendirilebilir.

Böylece, kolay hata ayıklama amaçları için, ilk başlarda kendi işletim sisteminizi Linux üzerinde çalışır şekilde tasarlayabilirsiniz (yavaşlığına rağmen), daha sonra Flux OS aracı[B76]nı kullanarak (kendi işletim sisteminizde Linux ve BSD sürücülerinin kullanımını garanti eder), onu kendi başına çalışır hale getirebilirsiniz. İşletim sisteminiz kararlı olduğunda, artık gerçekten sevdiyseniz, kendi donanım sürücülerinizi yazmanın vaktidir.

Bu NASIL belgesi önyükleyici (bootloader) kodlarını, 32 bitlik kipe geçmeyi, kesmelerle işlem yapmayı, Intel'in temel güvenli kipini veya V86/R86 beyinölümlülüğünü (braindeadness), nesne biçiminizi tanımlamayı ve çağrı uzlaşımlarını kapsaMAmaktadır.

Tüm bunlar için güvenli bilgi bulabileceğiniz yer halihazırdaki işletim sisteminin veya önyükleyicinin kaynak kodlarıdır. Pekçok konu şu adreste mevcuttur: `http://www.tunes.org/Review/OSes.html`

# 6. Hızlı başlangıç

## 6.1. Giriş

Son olarak, eğer hala bu çılgınca fikri denemek ve sembolik makina kodu yazmak istiyorsanız (eğer bu kısma ulaştıysanız gerçekten de bir sembolik makina hayranısınızdır), başlangıç için gerekenleri bu kısımda bulacaksınız.

Daha önce de okuduğunuz gibi, Linux için değişik şekillerde yazabilirsiniz; size *doğrudan* sistem çağrılarını nasıl yapacağınızı göstereceğim, çünkü bu çekirdek servislerini çağırmanın en hızlı yoludur; kodumuz hiç bir kütüphaneye bağlı değildir, ELF yorumlayıcısını kullanmayınız, çünkü çekirdek ile doğrudan iletişim kurar.

Aynı kodu **nasm** ve **gas** için göstereceğim ve böylelikle Intel ve AT&T söz dizimini de göstermiş olacağım.

Aynı zamanda Unix bembolik makne dili ile programlamaya giriş kılavuzu[B78]nu okumak isteyebilirsiniz; UNIX benzeri işletim sistemleri için örnek kodlar da içermektedir.

### 6.1.1. İhtiyacınız olan araçlar

Herşeyden önce bir çeviriciye (derleyici) ihtiyacınız vardır – **nasm** veya **gas**

İkinci olarak, bir ilintileyiciye (linker) ihtiyacınız vardır – **ld**, çünkü çeviriciler sadece nesne kodunu üretmektedir. Hemen her dağıtım **gas** ve **ld**'yi binutils içerisinde sunmaktadır.

**nasm**'a gelince, Linux için paketleri ve belgeleri *nasm sayfasından* (sayfa: 13) indirip kurmanız gerekebilir, pekçok dağıtımın (Stampede, Debian, SuSe, Mandrake) **nasm**'ı kendi sürümleri içerisinde barındırdıklarını unutmayın, önce bir kontrol edin.

Eğer daha derine inecekseniz, işletim sisteminizin başlık dosyalarını ve mümkünse çekirdek kaynak paketini edinmelisiniz.

## 6.2. Merhaba Dünyalı :-)

### 6.2.1. Yerleşim

Linux, 32 bitliktir, korumalı kipte çalışır, düz bellek modeline sahiptir ve ikilikler için ELF biçimini kullanır.

Bir program bölümlere ayrılabilir: kodunuz için `.text` kısnı (salt–okunur), verileriniz için `.data` kısmı (oku–yaz), ilklendirilmemiş veriler için `.bss` kısmı (oku–yaz); aslında bir kaç tane daha, kullanıcı tanımlı bölüm yanında, standart bölüm olabilir, fakat onların kullanılacakları durumlar çok nadir olmaktadır ve bizim ilgi alanımız dışındalar. Bir program en azından `.text` kısmına sahip olmalıdır.

Şimdi ilk programımızı yazacağız.

### 6.2.2. NASM (hello.asm)

```
section .text                           ;bölüm bildirimi

                        ;giriş noktasını ELF ilintileyiciye veya yükleyiciye
    global _start       ;göndermeliyiz (export). Giriş noktasını uzlaşımsal
                        ;olarak _start ile belirtiriz. Öntanımlı durumu
                        ;değiştirmek için: ld -e foo kullanın.

_start:

;dizgemizi stdout'a yazar

        mov     edx,len     ;üçüncü argüman: ileti uzunluğu
        mov     ecx,msg     ;ikinci argüman: yazılacak iletinin göstericisi
```

```
        mov     ebx,1       ;ilk argüman: dosya tutucu (stdout)
        mov     eax,4       ;sistem çağrısı numarası (sys_write)
        int     0x80        ;çekirdeği çağır

;ve çık

        mov     ebx,0       ;ilk sistem çağrısı argümanı: çıkış kodu
        mov     eax,1       ;sistem çağrı numarası (sys_exit)
        int     0x80        ;çekirdeği çağır

section .data               ;bölüm bildirimi

        msg     db      "Hello, world!",0xa     ;sevgili dizgemiz
        len     equ     $ - msg                 ;sevgili dizgemizin boyu
```

### 6.2.3. GAS (hello.S)

```
.text                           # bölüm bildirimi

                                ;giriş noktasını ELF ilintileyiciye veya yükleyiciye
    global _start               ;göndermeliyiz (export). Giriş noktasını uzlaşımsal
                                ;olarak _start ile belirtiriz. Öntanımlı durumu
                                ;değiştirmek için: ld -e foo kullanın.

_start:

# dizgemizi stdout'a yazar

        movl    $len,%edx       # üçüncü argüman: ileti uzunluğu
        movl    $msg,%ecx       # ikinci argüman: yazılacak iletinin göstericisi
        movl    $1,%ebx         # ilk argüman: dosya tutucu (stdout)
        movl    $4,%eax         # sistem çağrı numarası
        int     $0x80           # çekirdeği çağır

# ve çık

        movl    $0,%ebx         # ilk sistem çağrısı argümanı: çıkış kodu
        movl    $1,%eax         # sistem çağrı numarası (sys_exit)
        int     $0x80           # çekirdeği çağır

.data                           # bölüm bildirimi

        msg:
                .ascii  "Hello, world!\n"       # sevgili dizgemiz
                len = . - msg                   # sevgili dizgemizin boyu
```

## 6.3. Çalıştırılabilir bir kod üretmek

### 6.3.1. Nesne kodu üretimi

Çalıştırılabilir bir kod elde etmenin ilk adımı nesne dosyasını kaynaktan derlemek (veya çevirmek)tir:

**Nasm** örneği için:

```
$ nasm -f elf hello.asm
```

**gas** örneği için

```
$ as -o hello.o hello.S
```

Bu `hello.o` nesne dosyasını oluşturur.

### 6.3.2. Çalıştırılabilir üretmek

İkinci adım ilintileyiciyi çağırarak nesne dosyasının kendisinden çalıştırılabilir bir dosya üretmektir:

```
$ ld -s -o hello hello.o
```

Sonuç olarak bu `hello` çalıştırılabilir dosyasını üretecektir.

Hey, çalıştırmayı deneyin... Çalıştı mı? İşte bu. Oldukça basit.

## 6.4. MIPS Örneği

Gerçek dünyada x86 sülalesi dışında da bir evren var. Aşağıda Spencer Parkin tarafından sunulan ve MIPS işlemciler için yazılmış bir örnek vardır. Buraya kadar gelmişken http://www.cuillin.demon.co.uk/nazz/trivia/hw/hw_assembler.html adresindeki Bir Grup Sembolik Makina Dili ile Yazılmış Merhaba Dünya Programına bakabilirsiniz.

```
# hello.S       by Spencer T. Parkin

# Bu benim ilk MIPS-RISC sembolik makina dili programım!
# Derlemek için:
# > gcc -o hello hello.S -non_shared

# Bu program PlayStation2 MIPS R5900 (EE Çekirdek)
# üzerinde hatasız ve uyarısız derlenir.
# EE Duygu Makinası (Emotion Engine) anlamına gelir

# -non_shared seçeneği gcc'ye
# yeniden tahsis edilebilir kod ile ilgilenmediğimizi söyler.
# Eğer isteseydik, PIC-ABI çağrım kurallarını
# ve diğer protokolleri kullanmalıydık

#include <asm/regdef.h>         // Anaşılabilir yazmaç adları için
#include <asm/unistd.h>         // Sistem servisleri için

        .rdata                  # yalnız okunabilir veri bölümüne başlangıç
        align           2       # belleğin yapım şeklinden dolayı böyle

hello:  .asciz  "Hello, world!\n"  # Null ile sonlanmış bir karakter dizisi
        .align  4               # belleğin yapım şeklinden dolayı
length: .word   .- hello        # length = IC - (hello-addr)

        .text                   # kod bölümü başlangıcı
        .globl  main            # gcc/ld bağlamasından dolayı
        .ent    main            # gdp hata ayıklama bilgisi

main:                           # gcc'ye bir -non_shared alanı sunmalıyız
                                # ya da aşağıdaki üç satır etkin olmalı
#       .set    noreorder       # yeniden komut sıralamasını kapat
#       .cpload t9              # PIC ABI zırvası
#       .set    reorder         # yeniden komut sıralaması aktif
```

```
        move    a0,$0           # dosya tanımlayıcısı standart çıktıyı göstersin
        la      a1,hello        # karakter dizisi adresini yükle
        lw      a2,length       # karakter dizisi boyunu yükle
        li      v0,__NR_write   # sistem yazma servislerini belirt
        syscall                 # çekirdeği çağır (karakter dizisini yaz)
        li      v0,0            # geri dönüş kodunu yükle
        j       ra              # çağırana dönüş
        .end    main            # dgb için hata ayıklama bilgisi

# hepsi bu kadar millet!
```

# 7. Özkaynaklar

## 7.1. Siteler

Linux/UNIX sembolik makina dili ile programlama malzemeleri için asıl özkaynak:

http://linuxassembly.org/resources.html

Mutlaka ziyaret edin, Değişik UNIX işletim sistemleri ve işlemcileriyle ilgili pekçok sembolik makina dili projeleri, araçlar, kılavuz sayfaları, belgeler, rehberler, v.b. edinin. Çok çabuk geliştiği için, ikinci bir kere daha tekrar etmeyeceğim.

Eğer sembolik makina diline yabancıysanız işte size bir kaç başlangıç noktası:

- Sembolik Makina Dili Sanatı (The Art Of Assembly)[B82]
- x86 Sembolik Makina Dili Sıkça Sorulan Soruları (Google'u kullanın)
- ÇekirdekSavaşları (CoreWars)[B83], sembolik makina dilini genel hatlarıyla eğlenceli öğrenmenin bir yolu

## 7.2. Haber grupları

- comp.lang.asm.x86[B84];
- alt.lang.asm[B85]

## 7.3. Listeler

Eğer Linux/UNIX sembolik makina diliyle ilgileniyorsanız (veya sorularınız varsa ya da sadece merak ediyorsanız), sizi özellikle Linux sembolik makina dili programlama listesine davet ediyorum.

Bu, Linux, *BSD, BeOS ve diğer UNIX/POSIX benzeri işletim sistemleri altında sembolik makina dili ile programlama tartışmalarının olduğu açık bir ortamdır; ayrıca sadece x86 ile sınırlı değildir (Alpha, Sparc, PPC ve diğer bilgisayar üstatları da davetlidir!).

İleti listesi adresi: <linux–assembly (at) vger.kernel.org>.

Kayıt olmak için iletinin gövde bölümüne aşağıdaki metni yazıp <majordomo (at) vger.kernel.org> adresine epostanızı gönderiniz.

```
subscribe assembly
```

Ayrıntılı bilgi ve liste arşivleri http://linuxassembly.org/list.html adresinde mevcuttur.

# 8. Sıkça Sorulan Sorular

Aşağıda Linux sembolik makina dili ile programlamada sıkça sorulan sorular (cevapları ile) verilmiştir. Bazı soruar ve cevapları *linux–assembly ileti listesi* (sayfa: 27)nden alınmıştır.



### 8.1. Linux altında nasıl grafik programlama yaparım?

Paul Furber <paulf (at) icom.co.za> 'den bir cevap:

```
Pekala, Linux'ta grafik işleri için pekçok araç vardır. Hangisini
kullanacağınız ne yapmak istediğinize bağlıdır. Tüm bilgilere sahip
bir web sayfası yoktur ama işte bir kaç püf nokta:

SVGALib: Bu, konsoldan SVGA erişimi için C kütüphanesidir.
Artıları: öğrenmesi kolay, iyi kodlama örnekleri, DOS'taki gfx
kütüphanesinden pek de farklı değil, DOS'taki tüm etkiler az bir
değişiklikle dönüştürülebilir.
Eksileri: program doğrudan donanıma eriştiği için, çalışmak için root
erişim yetkilerine gerek duyar, her çip (chipset) ile çalışmaz,
X-Windows altında çalışmaz.
http://ftp.is.co.za'da svgalib-1.4.x diye aratın.

Framebuffer: SVGA ile ilgili kendinizce yapılabilen grafikler
Artıları: hızlı, doğrusal olarak haritalanmış video erişimi,
eğer isterseniz ASM kullanılabilir :)
Eksileri: çekirdek içinde derlenmeli, çipsete özgün özellikler,
çalışması için X kapatılmalı, iyi linux sistem çağrıları ve çekirdek
bilgisine dayanır, hata ayıklaması zordur
Örnekler: asmutils (http://www.linuxassembly.org), yaprak örneği,
framebuffer kodu ve asm ile ilgili ipuçları için benim sayfam
(http://ma.verick.co.za/linux4k/)

Xlib: XFree86 için uygulama ve geliştirme kütüphaneleri.
Artıları: X uygulamanız üzerinde tam bir kontrol
Eksileri: Öğrenmesi zor, çalışması korkunç ve az da olsa X'in düşük
seviyede nasıl çalıştığı bilgisini gerektirir
Tavsiye edilmez, ama onu için bu kadar yanıp tutuşuyorsanız durmayın.
Muhtemelen tüm başlık ve kütüphane  dosyaları yüklenmiştir, dolayısiyle
ihtiyacınız olana sahipsiniz.

Düşük seviyeli API'ler: PTC, SDL, GGI ve Clanlib'i içerir.
Artıları: çok esnek, X veya konsolda çalışır, video donanımını soyutlar
onun için düzgün doğrusal bir yüzey çizebilirsiniz, pekçok güzel örnek kod,
OpenGL ve ses kütüphaneleri gibi diğer API'lere bağlantı kurabilir
Microsoft DirectX sürümleri özgürdür.
Artıları: Kendi yaptığınız kadar hızlı değildir, gelişim sürecince bazen
sürümler çok sık değişir.
Örnekler: PTC ve GGI mükemmel demolara sahiptir, SDL ise oyunlar için,
sdlQuake, Myth II, Civ CTP ve ayrıca Clanlib'de kullanılmıştır.

Yüksek seviyeli API'ler: OpenGL - başka var mı?
```

```
Artıları: temiz API, yüzlerce işlevsellik ve örnek, endüstriyel
standart; bundan dolayı mesela SGI'dan öğrenilebilir
Eksileri: donanım hızlandırılması normalde bir zorunluluktur,
bazı sürümler ve platformlar arasında acayiplikler
Örnekler: çokça - bağlantılar bölümü altındaki www.mesa3d.org kısma bakınız.

Bakmayı sürdürmek için svgalib örneklerini inceleyin ve aynı zamanda
SDL'yı yükleyin ve çalışır duruma getirin. Bundan sonrasında ise limit
gökyüzüdür.
```

### 8.2. Saf (pure) sembolik makina kodunu Linux altında nasıl derlerim?

Sembolik Makina Dili Hata Ayıklayıcısı'nın (Assembly Language Debugger[B89]), sembolik makina kodlarıyla çalışması için tasarlanmış eski bir sürümü vardır ve de Linux ve *BSD üzerinde çalışmabilmesi için yeterince taşınabilirdir. Halihazırda işlevseldir ve de doğru seçim olacaktır, bir bakın!

**gdb**'yi de deneyebilirsiniz ;). Kaynak kod hata ayıklayıcısı olmasına rağmen, saf sembolik makina kodlarını ayıklamak için de kullanılabilir, biraz hileyle **gdb**'ye istediğinizi yapmanızı söyleyebilirsiniz (maalesef **nasm**'ın `g` seçeneği **gdb** için yeterli bilgi üretmemektedir; sanırım bu **nasm**'ın bir açığı). Aşağıda Dmitry Bakhvalov <dl (at) gazeta.ru>'dan bir cevap var:

```
Kişisel olarak, gdb'yi asm uygulamarınının hatalarını bulmak için
kullanırım. şunu deneyin:

1) Derlemek için aşağıdaki kodu kullanın:
   $ nasm -f elf -g smth.asm
   $ ld -o smth smth.o

2) gdb'yi çalıştırın
   $ gdb smth

3) gdb içinde:
   (gdb) disassemble _start
   at _start+1'e bir kesme koyun
    (eğer at _start konursa çalışmaz, nedenini bilmiyorum)
   (gdb) b *0x8048075

   kodu takip edebilmek için aşağıdaki kodu kullanırım
   (gdb)define n
   >ni
   >printf "eax=%x ebx=%x ...etc...",$eax,$ebx,...etc...
   >disassemble $pc $pc+15
   >end

   daha sonra programı r parametresiyle çalıştırıp, n ile hata ayıklayın

   Umarım yardımcı olmuştur.
```

???'dan ek bir bilgi:

```
.gdbinit'imin içinde epeydir kullandığım bir makro var,
ve eminim hayatı daha kolay hale getiriyor.
Az farkla: "x /8i $pc"' kullanırım bu da belli sayıdaki çevrilmemiş
koda müsaade eder.  Daha sonra iyi şekilde boyutu seçilmiş xterm'imle
gdb çıktısı tazelenmiş olarak ve kaydırma gerektirmeden görünür.
```

Eğer kodunuza kesmeler koymak istiyorsanız, sadece `int 3` ifadesini kesme olarak kullanabilirsiniz. (**gdb** içerisine elle adresi gömmek yerine).

Eğer **gas** kullanıyorsanız, **gas** ve **gdp** ile ilgili belgelere[B90] başvurmalısınız..

### 8.3. Başka faydalı araçlar var mı?

Elbette, **strace** size yardım edebilir (FreeBSD'de **ktrace** ve **kdump**), bu sistem çağrı ve sinyallerini takip etmek için kullanılır. Ayrıntılar için kılavuz sayfasını (**man trace**) ve **strace --help** komutunu deneyiniz.

### 8.4. Linux'tan (BSD, BeOS, v.b.) BIOS işlevlerine nasıl erişebilirim?

Kısa cevap: Hiç bir şekilde. Bu korumalı bir kiptir, yerine işletim sistemi servislerini kullanın. Tekrar ediyorum, `int 0x10`, `int 0x13`, v.b.'yi kullanamazsınız. Ne şans ki, hemen her şey sistem çağrıları ve kütüphane işlevleri ile ifade edilebilmektedir. En kötü durumda, port erişimini deneyebilir ve bir çekirdek yamasıyla istenileni gerçekleştirmeyi deneyebilirsiniz veya LRMI kütüphanesini kullanarak BIOS işlevlerine erişmeyi deneyin.

### 8.5. Sembolik makina dilinde çekirdek modülleri yazmak mümkün mü?

Evet, aslında mümkün. Her ne kadar genelde iyi bir fikir olmamasına rağmen (bir şeyleri çok zor hızlandıracaktır), böylesine bir yönteme ihtiyaç olabilir. Bir modülün kendisini yazmak o kadar da zor değildir – bir modülün kendisinin öntanımlı evrensel işlevleri olmalıdır, aynı zamanda bazı harici işlevleri de çekirdekten çağırması gerekebilir. Ayrıntılar için çekirdek kaynak koduna (bir modül olarak derlenebilenlere) bakınız.

Bu arada, işte size en basitinden bir çekirdek modülü (kaynak APJ #8'den mammon_'un örneğine dayanmaktadır):

```
section .text

        global init_module
        global cleanup_module
        global kernel_version

        extern printk

init_module:
        push    dword str1
        call    printk
        pop     eax
        xor     eax,eax
        ret

cleanup_module:
        push    dword str2
        call    printk
        pop     eax
        ret

str1            db      "init_module done",0xa,0
str2            db      "cleanup_module done",0xa,0

kernel_version  db      "2.2.18",0
```

Bu örneğin yaptığı tek şey yaptıklarını rapor etmekdir. `kernel_version`'unu sizinkine uygun halde değiştirin ve modülü şöyle derleyin:

```
$ nasm -f elf -o module.m module.asm
$ ld -r -o module.o module.m
```

Artık onunla **insmod/rmmod/lsmod** (root yetkileri gerekir) kullanarak oynayabilirsiniz; çok eğlenceli, değil mi?

### 8.6. Belleği dinamik olarak nasıl tahsis edebilirim?

H–Peter Recktenwald <phpr (at) snafu.de> 'dan özlü bir cevap:

```
ebx := 0        (in fact, any value below .bss seems to do)
sys_brk
eax := current top (of .bss section)

ebx := [ current top < ebx < (esp - 16K) ]
sys_brk
eax := new top of .bss
```

Tiago Gasiba <ee97034 (at) fe.up.pt> 'dan daha gelişmiş bir cevap:

```
section .bss

var1    resb    1

section .text

;
;allocate memory
;

%define LIMIT   0x4000000          ; yaklaşık 100Megs

        mov     ebx,0              ; data bölümünün en alt kısmını elde et
        call    sys_brk

        cmp     eax,-1             ; tamam mı?
        je      erro1

        add     eax,LIMIT          ; +LIMIT bellek kısmını tahsis et
        mov     ebx,eax
        call    sys_brk

        cmp     eax,-1             ; tamam mı?
        je      erro1

        cmp     eax,var1+1         ; data bölümü büyüdü mü?
        je      erro1

;
;tahsis edilmiş alanı kullan
;
                                   ; şimdi eax data bölümünün alt kısmını
                                   ; barındırır
        mov     ebx,eax            ; alt kısmı kaydet
        mov     eax,var1           ; eax=data bölümün başlangıç kısmı
repeat:
        mov     word    [eax],1    ; 1'lerle doldur
        inc     eax
        cmp     ebx,eax            ; şu anki pozisyon = en alt?
        jne     repeat

;
```

```
;belleği serbest bırak
;

        mov     ebx,var1            ; belleği geri ver
        call    sys_brk             ; başlangıcını=var1 yaparak

        cmp     eax,-1              ; tamam mı?
        je      erro2
```

### 8.7. `select` sistem çağrılarını nasıl kullanacağımı anlayamıyorum!

Patrick Mochel <mochel (at) transmeta.com> 'den bir cevap

```
sys_open'ı çağırdığınız zaman, sürecinizle ilgili açık olan tüm
dosya tanıtıcılarının olduğu bir tablodan bir indis olarak, bir
dosya tanıtıcı döndürür. stdin, stdout ve stderr için sırasıyla
0, 1 ve 2'dir, çünkü bunlar süreciniz için her zaman açık durur.
Aynı zamanda ilk açtığınız dosya tanıtıcısının 3 olocağını ve
artacağını unutmayın.

İndislemeyi anlamak select'in ne yaptığını anlamanızı sağlar.
select'i çağırdığınız zaman, okumak için, yazmak için, istisnai
durumları belirlemek için belli bir dosya tanıtıcısını beklediğinizi
belirtiyorsunuzdur. Süreciniz 1024 açık dosya tanıtıcısına sahip olabilir,
dolayısiyle fd_set sadece bir bit maskesi gibi çalışarak hangi
dosya tanıtıcısının hangi işlem için geçerli olduğunu belirtir.
Bir şeyler ifade etti mi?

Her açtığınız fd birer indis olduğu için, her bir fd_set için on
veya off olmaya ihtiyacı vardır, sadece 1024 bitlik bir fd_set yapısına
ihtiyacınız vardır. Yapıyı belirtmek için 1024 / 32 = 32 long gereklidir.

Şimdi basit bir örnekle açıklayalım.
Farzedelimki dosya tanıtıcıyı okumaya çalışıyorsunuz (zamanaşımı olmadan).

- fd_set'e belleği tahsis et

.data

my_fds: times 32 dd 0

- okumak istediğiniz dosya tanıtıcıyı açın

- fd_set yapısındaki bitini ayarlayın

    Öncelikle, 32 dwords'ün hangisinde bitin olduğunu belirlemelisiniz.

    Daha sonra, bts'yi kullanarak bu dword içindeki biti ayarlayın,
    bts biti 32'ye bölümden kalana göre ayarlayacaktır.
    Bu da önce hangi dword ile çalışmaya başlamanızı belirlemenin sebebidir.

    mov edx, 0
    mov ebx, 32
    div ebx

    lea ebx, my_fds
    bts ebx[eax * 4], edx
```

```
- son adımı okumak istediğiniz dosya tanıtıcılar için yineleyin

- belli bir eylem beklediğiniz diğer iki fd_set'in herbiri için de
  tüm örneği tekrarlayın

Bundan, geriye denklemin diğer iki parçası kalır - n parametresi ve
zaman aşımı parametresi. Zaman aşımı parametresini okuyucuya örnek
olarak bırakıyorum (evet, tembelim), fakat kısaca n parametresiyle
ilgili konuşacağım.

Bu, seçtiğiniz dosya tanıtıcıları içerisindeki en büyük değere sahip
olanın değeri (herhangi bir fd_set'ten) artı birdir. Neden artı bir?
Çünkü, bu değerden maskeyi belirlemek kolaydır. Farzedin ki x dosya
göstericisi üzerinde bir veri var, fakat sizin ilgilendiğiniz en yüksek
olan (n-1). fd_set sadece bir bit maskesi olduğundan, çekirdeğin
select'en geri dönmesi veya dönememesi için verimli bir yola ihtiyacı
vardır. Dolayısiyle, bu, ilgilendiğiniz bitleri maske dışında bırakır,
halihazırda atanmış bitlerde herhangi bir değer var mı diye kontrol eder,
eğer varsa geri döner. Aslında, fantastik olduğunu söylemek düşündüğüm kadar
kolay değil. Çekirdeğin bu maskeyi nasıl belirlediğini görmek için,
çekirdek kaynak ağacındaki fs/select.c'ye bakınız.

Herneyse, bu numarayı bilmeye ihtiyacınız vardır, en kolay yol da açılmış
olan en son dosya tanıtıcının numarasını bir yerlere kaydetmektir.

Evet, bildiklerim bunlar. Yukarıdaki kodla ilgili uyarı (her zaman ki gibi)
şu: test edilmemiştir. Sanırım çalışır, eğer çalışmazsa beni haberdar edin.
Fakat, eğer evrensel bir nükleer felakete sebep olursa, o zaman aramayın. ;)
```

*Şimdilik hepsi bu kadar..*

# 9. Ekler

## 9.1. Tarihçe

Belgenin geçmişini merak ediyorsanız orjinal belgeye[B94] bakabilirsiniz.

## 9.2. Teşekkür

Fikirleri, yanıtları, yorumları ve moral destekleri olan herkese teşekkür ederim, ayrıca yazılış sırasıyla aşağıda yazılanlara da teşekkür ederim.

- Linux için Linus Torvalds'a <buried.alive (at) in.mail>
- a86'dan bcc'yi çıkardığı için Bruce Evans'a <bde (at) zeta.org.au>
- NASM için Simon Tatham <anakin (at) pobox.com> ve Julian Hall'a (<jules (at) earthcorp.com>)
- NASIL belgelerini sürdürdükleri için Greg Hankins <gregh (at) metalab.unc.edu> ve şimdilerde Tim Bynum'a <linux–howto (at) metalab.unc.edu>
- Belgenin SSS'i için Raymond Moon'a <raymoon (at) moonware.dgsys.com>
- küçük–NASIL belgesini Fransızca'ya (üzücü olan taraf orjinal yazarın Fransız olması ve İngilizce yazması) çevirdiği için Eric Dumas'a <dumas (at) linux.eu.org>
- bana yardım ettikleri için, olmasa bile NASIL belgesini üstlendikleri için Paul Anderson <paul (at) geeky1.ebtech.net> ve Rahim Azizarab'a(<rahim (at) megsinet.net>

- GCC çağrımıyla ilgili görüşleri için Marc Lehman'a <pcg (at) goof.com>
- bana argüman aktarım uzlaşımları konusunda yardım ettiği için Abhijit Menon–Sen'a <ams (at) wiw.org>

# GNU Free Documentation License

Version 1.2, November 2002



1. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

2. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world–wide, royalty–free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front–matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The

Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front–Cover Texts or Back–Cover Texts, in the notice that says that the Document is released under this License. A Front–Cover Text may be at most 5 words, and a Back–Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine–readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ascii without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard–conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine–generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

3. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

4. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front–Cover Texts on the front cover, and Back–Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine–readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer–network location from which the general network–using public has access to download using public–standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front–matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—-for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front–Cover Text, and a passage of up to 25 words as a Back–Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front–Cover Text and one of Back–Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

7. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

8. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

9. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

10. TERMINATION

    You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

11. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

## ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
  Copyright (C)  year  your name.
  Permission is granted to copy, distribute and/or modify this document
  under the terms of the GNU Free Documentation License, Version 1.2
  or any later version published by the Free Software Foundation;
  with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
  A copy of the license is included in the section entitled "GNU
  Free Documentation License".
```

If you have Invariant Sections, Front–Cover Texts and Back–Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

# 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.

(B1) http://linuxassembly.org

(B2) http://tunes.org

(B5) http://linuxassembly.org/howto.html

(B6) http://linuxdoc.org/docs.html

(B7) http://www.gnu.org/philosophy/

(B8) ../howto/gpl.pdf

(B9) http://www.fsf.org/

(B18) news:comp.compilers

(B19) news:comp.compilers

(B21) http://www.delorie.com/djgpp/

(B22) http://www.cygwin.com/

(B23) http://www.mingw.org/

(B29) http://goof.com/pcg/

(B32) http://para.inria.fr/

(B33) http://www.jwdt.com/~paysan/gforth.html

(B47) http://http://webster.cs.ucr.edu/AsmTools/HLA/

(B48) http://www.cs.cornell.edu/talc/

(B49) http://www.freepascal.org/

(B51) http://www.terse.com/

(B52) news:comp.compilers

(B54) ftp://ftp.forth.org/pub/Forth/Compilers/native/unix/this4th.tar.gz

(B55) ftp://ftp.tunes.org/pub/tunes/obsolete/dist/tunes.0.0.0/tunes.0.0.0.25.src.zip

(B57) http://www.tunes.org/

(B59) ftp://linux01.gwdg.de/pub/cLIeNUX/interim/libsys.tgz

(B60) http://www.fefe.de/dietlibc/

(B61) http://www.fowelin.de/christian/computer/libASM/

(B62) http://linuxassembly.org/asmutils.html

(B66) http://www.linuxdoc.org/LDP/lki/

(B67) http://www.linuxdoc.org/LDP/lki/lki-«2.html#ss2.11

(B69) http://www.ggi-«project.org/

(B70) http://www.xfree86.org/

(B71) http://www.dosemu.org/

(B73) ftp://ftp.lip6.fr/pub/pc/x2ftp/README.mirror_sites

(B74) http://www.cs.utah.edu/projects/flux/

(B75) http://www.cygwin.com/

(B76) http://www.cs.utah.edu/projects/flux/oskit/

(B78) http://linuxassembly.org/intro.html

(B82) http://webster.cs.ucr.edu/AoA/

(B83) http://www.koth.org/

(B84) news://comp.lang.asm.x86

(B85) news://alt.lang.asm

(B89) http://ald.sourceforge.net/

(B90) http://linuxassembly.org/resources.html#tutorials

(B94) http://tldp.org/HOWTO/Assembly-«HOWTO/history.html

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

6 Şubat 2007