GNU C Derleyicisi Kullanımı

Yazan:
M. Ali Vardar
<ali (at) linuxprogramlama.com>

Yazan:
Barış Şimşek
<simsek (at) acikkod.org>

2005

Özet

Bu belgede sadece GNU C Derleyicisinin kullanımı, o da başlangıç seviyesinde ele alınmıştır.

Konu Başlıkları

Geçmiş

1.1	Temmuz 2005	NBB

Özgün sürümü http://www.acikkod.org adresinde bulunan ve Barış Şimşek tarafından yazılmış "Kod Geliştirme–1: Program Kütüphaneleri" isimli makale belgeye eklendi.

1.0	2003	MAV

Belgenin özgün sürümü http://www.linuxprogramlama.com adresinde bulunabilir.

Yasal Açıklamalar

1. Basit GCC Kullanımı

GCC ismi "GNU Compiler Collection" sözcüklerinin baş harflerinden oluşmuştur. Gerçek ismini GNU C Compiler kelimelerinden almaktadır. Bu değişikliğin sebebi GCC paketinin eskiden sadece C derleyicisinden oluşmakta olduğudur. Diğer diller daha sonradan GCC ailesine eklenmiştir. Bu konuda bilgi için http://en.wikipedia.org/wiki/GNU_Compiler_Collection adresine bir ziyaret yapılabilir. http://gcc.gnu.org/ adresinden GCC paketinin son sürümlerini takip edebilirsiniz.

Kullandığınız GCC sürümünü öğrenmek için;

```
# gcc -v
```

komutunu kullanabilirsiniz. Benim sistemimde dönen sonuç:

```
# gcc -v
Reading specs from /usr/lib/gcc-lib/i386-slackware-linux/3.2.2/specs
Configured with: ../gcc-3.2.2/configure --prefix=/usr --enable-shared
--enable-threads=posix --enable-__cxa_atexit --disable-checking --with-gnu-ld
--verbose --target=i386-slackware-linux --host=i386-slackware-linux
Thread model: posix
gcc version 3.2.2
```

Derleyicinin uygulamaları doğru bir şekilde derlemesi için bir takım parametrelerin doğru olarak verilmesi gerekmektedir. En basit şekli ile uygulamamızı yazalım ve ilk derleme işlemimize geçelim.

```
#include <stdio.h>
int main()
{
  printf("Örnek uygulama");
}
```

Yukarıdaki en basit uygulamamızdan başlamış olduk. Şimdi bu uygulamamızı derlemeye geçelim. Uygulamayı herhangi bir metin düzenleyici ile yazdıktan sonra `ilk.c` olarak kaydededelim ve konsolda komut istemine aşağıdaki satırları yazalım.

```
# gcc ilk.c
```

Bu satırdan sonra uygulamada hata yok ise veya bir uyarı mesajı vermezse hemen komut istemine düşülür. Ancak kaynak kodun olduğu dizine bakıldığı zaman `a.out` isimli bir dosyanın oluştuğu görülecektir. Uygulamamızı bir isim vermeden derledik. Bu gibi durumlarda **gcc** öntanımlı olarak `a.out` dosya adını kullanacaktır. Şimdide uygulamamızı kendi ismiyle derleyelim. Bu amaçla `-o` seçeneği kullanılır. `-o` seçeneği parametre olarak çıkış dosya adınını alır:

```
# gcc ilk.c -o ilk
```

Komut satırından sonra ilgili dizin içerisinde artık `ilk` isimli bir çalıştırılabilir dosya oluşacaktır. Bu aşamadan sonra artık basit bir uygulamanın derlenmesi işlemi anlaşılmış oldu. Peki ya uygulama içerisinde dışarıdan bir takım kitaplıklar eklenmek istenirse nasıl derlenmelidir? Bu aşamada örnek basit uygulamamız üzerinde bir iki ufak değişiklik yapalım;

```
#include <ncurses.h>
int main()
{
  initscr();
  getch();
  endwin();
  printf("bitti\n");
}
```

GNU C kütüphanesinde **getch()** işlevi yoktur. Ancak ncurses kütüphanesi kullanılması durumunda bu komut mevcuttur. Dikkat edilecek olursa `stdio.h` uygulamaya eklenmedi. Ön bilgi olarak, `ncurses.h` kullanılması durumunda `stdio.h` kullanılmasına gerek yoktur. Bu uygulamayı aynı şekilde aşağıdaki ilk öğrendiğimiz standart derleme yöntemi ile derleyelim.

```
# gcc ilk.c -o ilk
/tmp/cc4k34cd.o(.text+0x11): 'main' işlevinde:
: 'initscr'ye tanımsız başvuru
/tmp/cc4k34cd.o(.text+0x1a): main' işlevinde:
: 'stdscr'ye tanımsız başvuru
/tmp/cc4k34cd.o(.text+0x1f): main' işlevinde:
: 'wgetch'ye tanımsız başvuru
/tmp/cc4k34cd.o(.text+0x27): main' işlevinde:
: 'endwin'ye tanımsız başvuru
collect2: ld çıkış durumu 1 ile döndü
```

gcc bize bir çok tanımlanmamış nesne olduğunu bildiriyor. Aslında bunlar yazdığımız kodla ilgili hatalar değil, ilintileyici (**ld**) bazı kütüphaneleri bulamamış. Bu işlevlerin **ncurses** kütüphanesinde bulunduğunu biliyoruz. Bizim yapmamız gereken **ncurses** kütüphanesini **gcc**'ye tanıtmak, yani ilgili seçeneği (`-l`) komut satırına eklemek olacaktır.

```
# gcc ilk.c -o ilk -lncurses
```

Bu komuttan sonra, uygulama hata vermeden derleme işlemini gerçekleştirecektir. Önemli hatırlatma: uygulamaya eklenecek olan kütüphaneler eğer sistemde standart olan yerlerde ise bu kullanım yeterlidir. Ama bu standart yerler nerelerdir? Bu tanım dağıtımlara göre farklılıklar gösterebilir ancak tüm kütüphanelere ait başlık dosyaları genel olarak `/usr/include` içindedir. Eğer bu başlık dosyaları farklı bir yerde ise, bulundukları yerin belirtilmesi için farklı bir seçenek (`-I`) kullanılmalıdır:

```
# gcc ilk.c -o ilk -I/usr/local/include -lncurses
```

Bu satır da aynı şekilde uygulamamızı sorunsuz olarak derleyecektir. Burada uygulamamız içine eklenen başlık dosyalarını araması için standart dizin dışında `/usr/local/include` dizininide eklemiş olduk. Bu aşamadan itibaren **ncurses** kütüphanesini uygulamamız içerisine ekleyebiliyoruz ve istediğimiz **ncurses** komutlarını kullanabiliriz. Bunun yanında **mysql** kütüphanesini de uygulamaya katmak istersek uygulamamızı aşağıdaki şekle sokabiliriz:

```
#include <ncurses.h>
#include <mysql.h>
int main()
{
  initscr();
  getch();
  endwin();
  printf("bitti\n");
}
```

En son derleme satırımız ile derleyelim. Aynı zamanda `-lmysql` ile mysql kütüphanesini kullanacağımızı belirtelim:

```
# gcc ilk.c -o ilk -I/usr/local/include -lncurses -lmysql
ilk.c:2:19: mysql.h: Böyle bir dosya ya da dizin yok
```

Neden `mysql.h` dosyasını bulamadı? Halbuki `-I/usr/local/include` yazdık. Sistemde ufak bir arama yapalım bakalım doğru mu yazdık?

```
# locate mysql.h
```

```
/usr/share/apps/quanta/doc/php/ref.mysql.html
/usr/include/mysql/mysql.h
```

İşte aradığımız satır. `mysql.h` dosyası `/usr/include/mysql` dizini içerisinde bulunmaktaymış, öyleyse derleme seçeneklerinde bir değişiklik yapalım:

```
# gcc ilk.c -o ilk -I/usr/local/include -I/usr/include/mysql -lncurses -lmysql
```

Ve sonuç başarılı olacaktır. Bu şekilde istendiği kadar kütüphaneyi uygulamamız içerisine ekleyebiliriz. Eğer aranan kütüphanenin nesne dosyaları farklı bir dizin içerisinde ise `-L` seçeneği ile bu kütüphanenin yolunu tanımlamalıyız. Örnek olarak;

```
# gcc ilk.c -o ilk -I/usr/local/include -I/usr/include/mysql -L/usr/local/lib
¬ -lncurses -lmysql
```

Aynı şekilde matematik kütüphanesini uygulamamızda kullanacaksak `-lm` eklemeliyiz. Şifreleme algoritması kullanacaksak `-lcrypt`, postgresql için `-lpq`, POSIX evreleri kullanımı için `-lpthread`, glib kütüphanesini kullanmak için `-lglib`, vga kütüphanesini kullanmak için `-lvga` seçeneklerini örnek olarak verebiliriz. Bu kullanım örnekleri sisteminize kurduğunuz kütüphanelerin miktarına göre değişebilir.

Şimdi de **gcc** ile kullanabileceğimiz diğer seçeneklere gelelim. `-m` seçeneği işlemci seçimli derleme işlemini gerçekleştirmek için kullanılır. 386 komut seti için `-m386`, 486 komut seti için `-m486`, pentium komut seti için `-mpentium` parametresi verilmelidir. Örnek olarak;

```
# gcc ilk.c -o ilk -I/usr/local/include -I/usr/include/mysql -lncurses -lmysql
¬ -mpentium
```

komutu sonrası uygulama pentium işlemci için derlenmiş olacaktır. 486 seçimli derleme işleminden sonra özellikle büyük uygulamalarda kodda bir miktar büyüme olabilir, ancak bu hız ile ters orantılı olarak artmaktadır.

Diğer önemli bir kullanım şekli uygulamanın kullandığı kütüphaneleri doğrudan uygulama içerisine eklemektir (`-static` seçeneği). Bu şekilde uygulamanın çalıştırılabilir dosyası içerisine kütüphaneler de eklenecektir. Bu işlemi örnek uygulamamız içerisinde deneyelim;

```
# gcc ilk.c -o ilk -I/usr/local/include -I/usr/include/mysql -lncurses -lmysql
¬ -mpentium -static
```

Her iki farklı derleme şeklinden sonra uygulamanın büyüklüğüne dikkat ediniz, büyüklüğünde epey bir fark olduğu görülecektir.

Uygulama içerisinde verilen hata iletileri haricinde bir takım uyarı iletileri de bulunmaktadır. Örneğin, bir değişkenin tanımlanmış olmasına rağmen kullanılmaması; uygulama içerisinde bir değişken tanımlayalım. Derleme seçeneklerine de `-Wall` seçeneğini ekleyelim ve derleyelim.

```
# gcc ilk.c -o ilk -I/usr/local/include -I/usr/include/mysql -lcurses -Wall
ilk.c: 'main' işlevinde:
ilk.c:5: uyarı: değişken 'a' kullanılmamış
ilk.c:10: uyarı: denetim void olmayan işlevin sonunu aşıyor
```

Böylece karşımıza uyarılar çıkmaya başlayacaktır. Uygulama içerisinde eniyileme yapılmasına önem verilmesi durumunda bu gibi gereksiz değişkenler temizlenmelidir. `-w` seçeneği ile tüm uyarıların ekrana çıkmasına engel olunur.

gcc aynı zamanda kendi içerisinde yazılmış olan kodlar üzerinde de eniyileme yapmaktadır. Bu amaçla `-O` seçeneği kullanılmaktadır. Örnek uygulamamızı aşağıdaki şekilde derleyelim:

```
# gcc ilk.c -o ilk -I/usr/include -I/usr/include/mysql -lcurses -O1
```

Bu seçenek `-O0`, `-O1`, `-O2`, `-O3`, `-Os` değerlerini alabilir. `-O0` hiç eniyileme yaptırmazken, `-O3` en yüksek eniyileme ile uygulama kodunu düzenler. `-Os` ise `-O2` eniyilemelerine ek olarak kod boyunu kısaltacak eniyilemeleri yapar.

2. Kütüphaneler ve GCC

Yazılım kütüphaneleri, daha sonra yeni geliştirilecek yazılımlara dahil edilmek üzere saklanan derlenmiş hazır kod ve veri dosyalarıdır. Kütüphane dosyaları, modüler yazılım geliştirmeye olanak sağlarlar

2.1. Statik Kütüphaneler

Statik kütüphaneler, yeniden derlenmeye ihtiyaç duymaksızın yazılıma derleme aşamasında yani çalıştırılmadan önce eklenir. `lib` dizinlerindeki "`.a`" uzantılı dosyalar statik kütüphanelerdir. Derleme zamanını azaltmak için yakın geçmişe kadar statik kütüphaneler kullanılıyordu. Ancak günümüzün hızlı bilgisayarları için derleme zamanı sorun olmaktan çıktı. Bu nedenle statik kütüphaneler artık tercih edilmemektedir. Ancak yine de kütüphane kodlarını açmak istemeyenler için kullanışlıdır.

Statik kütüphane oluşturmak için **ar**(1) komutu kullanılır.

```
# ar rc libtest.a test.o util.o
```

Bu komut `libtest.a` kütüphanesini oluşturur, `test.o` ve `util.o` nesne dosyalarını bu kütüphaneye ekler. `c` seçeneği ile eğer kütüphane yoksa oluşturulması belirtilmiş olunur. Eğer varsa nesne dosylarını bu kütüphaneye ekler. `r` seçeneği ile eski nesne dosyalarının yenileri ile değiştirilmesi talep edilir.

Yeni oluşturulan veya değiştirilen bir kütüphane dosyasını yeniden indekslemek gerekir. Bu işlem derleme sırasında derleyicinin sembolleri denetlemesini hızlandırmak içindir. İndeks oluşturmak için **ranlib**(1) kullanılır.

```
# ranlib libtest.a
```

Derlenecek bir yazılıma daha önce oluşturduğumuz bir kütüphaneyi dahil etmek için **gcc**'ye `-l` seçeneği ile kütüphane ismi parametre olarak verilmelidir. Bu işlem için tercih edilmese de doğrudan **ld**(1) kullanılabilir.

```
# gcc prog.o -L/home/simsek/libtest -ltest
```

`-l` ile kütüphane ismini verirken dosya isminin başındaki `lib`'in kaldırıldığına dikkat edilmeli. `-L` seçeneği ile **gcc**'ye kütüphaneleri ararken hangi dizinleri kullanacağı bildirilir.

Aşağıda bir kütüphane oluşturma ve kullanma senaryosu mevcut:

```
# cat main.c

#include <stdio.h>

/* işlevler external bildirilmeli */
extern void test();
extern void util();

int main() {
  printf("main içinde\n");
  /* Kütüphane içinden işlev çağıralım */
  test();
  util();
  return 0;
}
```

```
# cat test.c

int test() {
  printf("Test içerisinde\n");
  return 0;
}

# cat util.c

int util() {
  printf("Util içerisinde\n");
  return 0;
}

# gcc -c test.c
# gcc -c util.c
# ar rc libtest.a test.o util.o
# ranlib libtest.a
# gcc -c main.c
# gcc main.o -L /home/simsek -ltest -o testprog
```

Son satırda kütüphane işlevlerini kullanan yazılım derlenip **testprog** isimli çalışabilir dosya derleyici tarafından oluşturuldu. **testprog** çalıştırıldığında aşağıdaki çıktıyı verecektir:

```
# ./testprog
main içinde
Test içerisinde
Util içerisinde
```

2.2. Paylaşımlı (Shared) Kütüphaneler

Paylaşımlı kütüphaneler, yazılım çalıştırıldığında sürece dahil edilirler. Yazılımın içerisine gerçekten kod eklenmez. Paylaşımlı kütüphaneden bir işlev cağrılan yerlere, kütüphaneye bir referans verilir. Kütüphaneyi kullanan bütün yazılımlar bu şekilde aynı işlev kütüphanesini ortak kullanırlar.

Paylaşımlı her kütüphane “so ismi” (soname) denilen bir özel isme sahiptir. “so ismi” kütüphanenin gerçek dosya ismine sembolik bağdır. Ayrıca derleyicinin bir kütüphaneyi talep ettiğinde kullandığı bir de ilintileyici ismi vardır. Bu, sürüm numarası olmayan bir so ismi olarak düşünülebilir.

```
simsek@mail:/usr/lib$ ls libz*
libz.a libz.so.1@ libz.so.1.2.1.1*
```

Listede görülen `libz.a` statik kütüphaneye aittir. `libz.so.1` ise `libz.so.1.2.1.1` paylaşımlı kütüphanesine sembolik bağ olarak kütüphanenin so ismidir. Bu kütüphane için ilintileyici ismi `libz.so`'dur.

Yazılımlar içerisinde kütüphanelerin so isimleri tutulur. Bu sayede kütüphanede bir değişiklik yapıldığında yeni kütüphane kullanıma alınmış olunur.

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.

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

6 Şubat 2007