Linux Çekirdeğine Nasıl Katkı Verilir?

Ebru Akagündüz [1], Tülin İzer [2]

[1] Çanakkale Onsekiz Mart Üniversitesi
[2] Galatasaray Üniversitesi
ebru.akagunduz@gmail.com, tulinizer@gmail.com

Özet: Linux özgür bir işletim sistemidir. 20 yılı aşan bir süredir geliştirilmektedir. Diğer tüm özgür yazılım projeleri gibi Linux da kendi ekibi dışındaki geliştiricilerden katkı alır. Çekirdek son kullanıcıya yakın olmadığından katkı vermesi belki bir miktar daha zor olabilir. Çekirdeğe katkı verme sırasında okumak için çok belge var ancak bunlar ilk bakışta çok açık olmayabilir. Bu belgede Linux Kernel gibi büyük bir organizasyona katkı verirken izlenilmesi gereken yollar, bilinmesi ve uyulması gereken temel şeyler açıklanmıştır.

1. Katkı Verme Sürecinde Kullanılacak Temel Araçlar

Temel olarak kullandığımız iki araç git ve bir editör. Editör seçimi herhangi bir editör olabilir, biz vi editörünü kullandık.vimrc'deki herhangi bir ayarı değiştirmeden editörü kullanabiliriz. Linux çekirdeği C ile kodlanmıştır ve her satırdaki girinti 4 boşluktan oluşur. Eğer vi editörü kullanıyorsak boşlukları sekme (tab) kullanarak bırakılmaması gerekir.

Git'te yapılması gereken ayarlar:

.gitconfig dosyasında ad ve e-posta adresi aşağıdaki gibi belirtilmelidir. Ayrıca yama dosyaları git kullanılarak gönderileceği için aşağıdaki gibi ayarlamaları yapmalıyız. Yama dosyalarının içeriğini e-posta içerisine kopyalayıp göndermek geliştiriciler tarafından kabul edilmiyor çünkü koddaki boşluklar kayabilir. Bu yüzden yama dosyaları tarayıcıya ya da bir istemciye kopyala-yapıştır şeklinde gönderilmemelidir.

```
[user]
	email = ebru.akagunduz@gmail.com
	name = Ebru Akagunduz
[sendemail]
	smtpencryption = tls
	smtpserver = smtp.gmail.com
	smtpuser = ebru.akagunduz@gmail.com
	smtpserverport = 587
	pass = PASS
	chainreplyto = false
[core]
	editor = vim
```

Katkı vermek için temel olarak önerilen sanal makineye Linux kurup, orada çalışmak. Ancak sanal makine üzerinde çalışmak biraz yavaş olabilir. Bu yüzden biz kendi ana makinemizde çalıştık.

2- Çekirdek Dizini İçerisindekiler

Linux çekirdeğindeki staging sürücüsüne yama göndermek için öncelikle bu sürücünün bakıcısı (mainteiner) olan Greg Kroah-Hartman'ın staging deposunu klonlamalıyız.

```
$git clone -b staging-next git://git.kernel.org/pub/scm/linux/kernel/git/gregkh/staging.git
```

Ardından, make komutu ile çekirdeği derlemeliyiz. Çok çekirdekli işlemcili bir bilgisayara sahipsek, make komutuna jX opsiyonu vererek çekirdeği çok işlemci üzerinde derleyebiliriz. Ardından, derlediğimiz çekirdeği yüklemeliyiz. Bunun için ya GRUB'ı manuel olarak güncellemeliyiz ya da **instalkernel scriptimiz** varsa onu çalıştırmalıyız. Ubuntu'da instalkernel scripti yüklü olarak geliyor. Bu durumda çekirdeği yüklemek için şu komut çalıştırılmalı:

```
$sudo make modules_install install
```

Linux, hem eski hem yeni çekirdek versiyonlarımızı saklıyor, bu sayede yeni çekireğimizde bir sorunla karşılaşırsak yeni çekirdeği boot edebiliriz. GRUB hangi çekirdeği boot edeceğimizi seçmemize olanak tanıyor.

3- Kullandığımız Git İşlemleri

Git dağıtık bir versiyon kontrol sistemidir. Git'in dağıtık olması demek kod deposunun tek bir merkezden yönetilmediği anlamına geliyor. Yani diğer geliştiricilerden bağımsız olarak kendi versiyonumuz üzerinde çalışabiliriz. Bir kod deposunu kopyalayarak aslında klonlama (clone) işlemi gerçekleştiriyoruz ve orijinal deponun sahip olduğu tüm fonksiyonelliğe sahibiz. Ayrıca dağıtık olması sayesinde, çevrim dışı iken de kodlarımızı içeri atabiliyoruz (commit) . Çevrim içi olduğumuzda uzak kod deposuna (remote) yaptığımız değişiklikleri gönderebiliriz (push). Uzak kod deposunda değişiklikleri kendi kod depomuza çekebiliriz (pull).

Daha sonra dosyalarımızı indekslemek için git add . komutunu kullanabiliriz. Bu kod ile dosyalarımızdaki değişiklikler algılanır ve kod deposuna eklenmeye hazır gelir. Dosyalarımızı kod deposuna eklemek için git commit -m "İlk commit" yazabiliriz. Böylelikle dosyalarımız kod deposuna eklenmiş olur.

Git ile ilgili bilmemiz gereken 3 ana terim repository, branch ve fork.

Respository, github üzerinde barındırdığımız proje yani depomuz.

Branch 'ı (dal), deponun alt klasörleri şeklinde düşünülebilir. Her dal tamamen farklı yama setlerini içerebilir. Linux kernel geliştiricileri genellikle her yama seti için ayrı bir dal oluşturur. Örneğin bir dal hata ayıklama, diğer bir dal geliştirmekle olduğunuz yeni bir özellik içeriyor olabilir. 'git branch' komutunu çalıştırarak hangi dalda çalışmakta olduğumuzu ve oluşturduğumuz diğer dalları görebiliriz.

```
tulin@ubuntu:~/git/kernels/staging$ git branch
 * staging-next
```

'ilk-yama' adında yeni bir dal oluşturmak ve bu dala geçmek için **checkout** komutunu kullanabiliriz.

```
$git checkout -b ilk-yama
```

Fork etmek demek ise, başkasının yarattığı bir depoyu, kendi depomuz olarak kopyalamak demek.

Oluşturduğumuz kod deposunu uzağa taşımak için öncelikle kendi depomuza uzak bir deponun adresini git remote add origin uzak_depo_adresi kodunu yazarak ekleyebiliriz. Yaptığımız değişiklikleri uzak depoya göndermek için ise git push origin master kodunu kullanabiliriz. Uzak depoda yapılan değişiklikleri kendi depomuza çekmek istersek git pull komutunu kullanabiliriz.

Linux çekirdeğine yama göndermek için Greg Kroah-Hartman'ın staging deposunu kendi bilgisayarımıza klonlamamız gerekli. clone komutunu kullandığımızda, Greg Kroah-Hartman'in deposunu uzak depo olarak eklemiş olduk. 'git branch' komutuna -a opsiyonunu vererek uzak depolarımızın dallarını da görebiliriz.

```
 tulin@ubuntu:~/git/kernels/staging$ git branch -a
* ilk-yama
  staging-next
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
  remotes/origin/staging-linus
  remotes/origin/staging-next
```

Oluşturduğumuz ilk uzak depoya varsayılan olarak "origin" deniyor. Aynı anda farklı isimler kullanarak birden fazla uzak depo ile çalışabiliriz. Uzak staging depomuzda 3 branch olduğunu görüyoruz. master, staging-linus ve staging-next. staging-linus branchi geçerli hata ayıklama yamalarını, staging-next branchi ise sıradaki çekirdek sürümü için yamaları içeriyor. Kod temizlemekten oluşan yamalar staging-next branchinde olduğundan dolayı, yamalarımızı bu dalı baz alarak oluşturmalıyız.

4. Çekirdeği Güncellemek

Linux çekirdeği katkı alma konusunda diğer projelerden daha yoğun, sürekli depodaki kodlar güncelleniyor. Bir kaç gün uğraş verip çözmek istediğimiz bir hata zaten başkası tarafından çözülmüş olabilir. Buna engel olmak için yerel depodaki çekirdek kodlarını mümkün olduğunca güncel tutmak gereklidir.

```
$ git remote add staging git://url
```

```
$ git fetch staging
$ git checkout -b staging-fixes-rebase
$ git rebase staging/staging-next
```

remote add ile izlediğimiz depolar arasına geliştiricinin deposunu eklemiş oluruz. Yerelde yeni dal açarak geliştiricinin **pull request'**leri kabul ettiği dalı depomuza eklemiş oluruz.

Burada "merge" yerine "rebase" kullanılması çok fazla kişinin bir arada çalışıyor olması. "merge" işleminin sorunsuz gerçekleşmesi için herkesin her şeyi mükemmel yapıyor olması gerekir ve genelde az kişinin çalıştığı projelerde kullanılabilir. Üstelik katkı verenler uzak depoya yazamadıklarından yaptıkları değişiklikler yerelde kalır, uzak depo ve yerel arasındaki farklardan dolayı "merge" işlemi gerçekleştirilemez.

4.1 Çekirdeğin Derlenmesi ve Yapılandırması

Derlemede gerekli paketler: gcc, libncurses5-dev, make

Çekirdek kodlarının bulunduğu dizinde bir yapılandırma dosyası (.config adında) olması gerekir. Bu dosyayı **/boot/** dizini altından sağlayabileceğiniz gibi yeniden de oluşturabilirsiniz. **"make menuconfig"**, yapılandırma dosyasına hangi sürücülerin eklenip eklenmeyeceğini seçmek için bir arayüz sağlar. menuconfig yerine **gconfig** veya **xconfig** gibi başka çeşitleri de kullanılabilir. Sonraki adımda "sudo make" ile derleme işlemini yapmalıyız. Bu işlemden sonra gerekli ayar dosyaları oluşmuş olacak. Sistemi yeniden başlattığımızda artık kendi derlediğimiz çekirdek ile sistem çalışacak.

Aynı zamanda Linux çekirdeği üzerinde çalışmak için önerilen en çok önerilen 2 kitap:
[1] http://lwn.net/Kernel/LDD3/
[2] http://it-ebooks.info/book/972/

5. Sürücü Kodlarında Değişiklik Yapmak

Bir sürücü üzerinde **printk("...")** ile bir şeyler yazdırmak gibi temel oynamalar yapılabilir. Burada bilinmesi gereken bir kaç bash komutu: **dmesg** ile printk("..")'da yazdırdığımız şeylerin çıktısını görebiliriz. **lsmod** ile yüklü olan modülleri görebiliriz. Sürücüyü derledikten sonra **.ko** olarak ikilik (binary) bir dosya oluşur. **insmod** ile modül yükleyebilir, **rmmod** ile kaldırabiliriz. Yüklü olan modülleri aynı zamanda **"/lib/modules/"** dizini altında görebiliriz.

6- Yama oluşturmak

Linux kernel geliştiricileri kod stili konusunda oldukça seçici ve katılar. Bu nedenle checkpatch.pl adında bir script geliştirmişler. Bu script yamamızın kernel kodlama stiline uygun olup olmadığını kontrol ediyor. Ancak, bir dosyadaki tüm checkpatch hatalarını bir yamada düzeltmemiz istenmiyor. Bunun yerine bir yamada bir mantıksal değişiklik yapılmalı.

7- Genel yama oluşturma kuralları

Yamamızın çekirdeğe kabul edilebilmesi için uymamız gereken bazı kurallar var. Mesela, yamamızın konusu kısa olmalı (50 karakterden daha az). Okuyana basitçe yamanın ne ile alakalı olduğunu anlatmalı. Açıklama kısmında, yamadaki değişikliğin neden yapıldığı anlatılmalı. Neden sorusu geliştiriciler için nasıl sorusundan daha önemli. Yani, burada değişikliği yaptığımız sürücünün bakıcısını (maintainer) yamamızı neden kabul etmesi gerektiği konusunda ikna etmeliyiz. Yamanın açıklaması "Signed-of-by" ile bitmeli. Birbirine bağlı değişiklikleri içeren bir kaç yama hazırlıyorsak, bunu bir yama serisi (patchset) şeklinde yapabiliriz. Her yama bir mantıksal değişiklik içermeli.

Gönderdiğimiz bir yamanın kabul edilmemesi durumunda, yamamızı alınan geri bildirime uygun bir şekilde güncellememiz ve tekrar göndermemiz gerekir. Bunun için 'git rebase -i' komutunu kullanabiliriz. Bu komut sayesinde, yama serimizi tekrar sıralama, birleştirme, bölme, düzenleme işlemleri gerçekleştirebiliriz. İşimize çok yarayabilecek başka bir komut ise 'git reflog'. Bu komut ile kaydetmediğimiz eski versiyonlara dönebiliriz.

Daha detaylı bilgi için: http://kernelnewbies.org/PatchPhilosophy, http://lxr.linux.no/linux/Documentation/CodingStyle.

8. Katkı Verilebilecek Alanlar

Katkı vermek için olan kodlar drivers/staging/ dizini altındadır. Geliştiricilerin kodlama biçimi düzeltme için yazdıkları bir Perl betiği var. Onunla şu gibi hatalar, uyarılar düzeltilebilir:

```
rivers/staging/rtl8187se/r8180_wx.c:519: ERROR: trailing statements should be on next line
rivers/staging/rtl8187se/r8180_wx.c:540: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:541: ERROR: need consistent spacing around '*' (ctx:WxV)
rivers/staging/rtl8187se/r8180_wx.c:544: ERROR: "(foo*)" should be "(foo *)"
rivers/staging/rtl8187se/r8180_wx.c:583: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:587: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:716: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:716: WARNING: printk() should include KERN_ facility level
rivers/staging/rtl8187se/r8180_wx.c:817: WARNING: Prefer netdev_info(netdev, ... then dev_info(dev, ... then pr_info(...  to printk(KERN_INFO ...
rivers/staging/rtl8187se/r8180_wx.c:821: WARNING: Prefer netdev_info(netdev, ... then dev_info(dev, ... then pr_info(...  to printk(KERN_INFO ...
rivers/staging/rtl8187se/r8180_wx.c:826: WARNING: Prefer netdev_info(netdev, ... then dev_info(dev, ... then pr_info(...  to printk(KERN_INFO ...
rivers/staging/rtl8187se/r8180_wx.c:855: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:875: WARNING: space prohibited before semicolon
rivers/staging/rtl8187se/r8180_wx.c:894: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:913: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:951: WARNING: space prohibited before semicolon
rivers/staging/rtl8187se/r8180_wx.c:1022: WARNING: printk() should include KERN_ facility level
rivers/staging/rtl8187se/r8180_wx.c:1037: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1038: ERROR: trailing whitespace
rivers/staging/rtl8187se/r8180_wx.c:1060: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1070: WARNING: printk() should include KERN_ facility level
rivers/staging/rtl8187se/r8180_wx.c:1071: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1071: ERROR: that open brace { should be on the previous line
rivers/staging/rtl8187se/r8180_wx.c:1072: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1079: WARNING: printk() should include KERN_ facility level
rivers/staging/rtl8187se/r8180_wx.c:1081: WARNING: printk() should include KERN_ facility level
rivers/staging/rtl8187se/r8180_wx.c:1087: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1088: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1121: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1122: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1152: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1183: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1353: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1357: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1361: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1364: WARNING: line over 80 characters
rivers/staging/rtl8187se/r8180_wx.c:1375: ERROR: "foo* bar" should be "foo *bar"
rivers/staging/rtl8187se/r8180_wx.c:1376: ERROR: "foo* bar" should be "foo *bar"
```

80 karakteri geçme hatası, **printk** yerine **netdev_info("..")** gibi başka mesaj döndürme fonksiyonlarının kullanılması, noktalı virgülden önce boşluk bırakılmaması, fazla parantez kullanımı gibi uyarılar düzeltebilir. Bunun dışında Todo dosyaları, Fixme etiketleri, Sparse, Coccinelle kullanarak kodları test edip katkı verebiliriz.

Todo dosyalarında istenenler genelde açık olarak belirtilmemiş oluyor. Eğer bu dosyalardaki işler üzerinde çalışmak istiyorsak, o sürücünün bakıcısından yardım almak gerekebilir. Geliştiriciler herhangi bir sıkıntı olmadan sorduğumuz her soruya cevap veriyorlar, konunun kendisini açıkça anlatıyorlar.

Fixme etiketleri de genelde çok açık olmayabiliyor. Onlar için de yardım almak gerekebilir. Fixme ve Todo dosyalarındaki işleri yapıp yamayı bir listeye e-posta gönderiyorsak cc'ye o sürücünün bakıcılarını eklemeliyiz. Bakıcı listesini **./scripts/get_maintainer.pl** dosyasını kullanarak alabiliriz.

Sparse, anlamsal ayrıştırıcıdır (semantic parser). Bu araç ile fonksiyon tiplerindeki yanlışlık, fazla olan kodları kaldırma, değişken tip dönüşümleri, fonksiyon prototipleri ekleme gibi uyarıları/hataları çözebiliriz.

Sparse kullanımı "$make C=2 drivers/staging/wlan-ng" şeklindedir.

Çekirdek kodlarında yaptığımız her değişiklikten sonra mutlaka sürücüyü derleyerek test etmeliyiz.

9. Commit Yapmak

Yaptığımız değişiklikleri commitlemeden önce bazen git diff ile kontrol etmek daha sağlıklı olabilir. **git commit -s** ile commit işlemini gerçekleştirmeliyiz. -s parametresi .gitconfig'de yaptığımız ayarlardan ismimizi ve e-posta adresimizi alır. **Signed-off-by** şeklinde commit mesajına otomatik ekler. **Singed-off-by** ifadesinin her commit içerisinde bulunması gerekir. Ayrıca commit mesajının ilk satırı düzelttiğimiz hatanın/uyarının ekrandaki tam çıktısı olmalıdır.

10- Yama göndermek

Bir yama göndermek için, yamayı bir mail olarak oluşturmak gerekir. Mail listelerine yamamızı dosya eki olarak gönderemeyiz.

Mutt ile yama göndermek:
$git format-patch -o /tmp/ HEAD^

Bu komut bir başlangıç commit ID si ve opsiyonel olarak uç commit ID si alır. Bu sayede başlangıç commitinden sonrası için yama oluştururuz. Oluşturduğumuz ilk commiti belirtmek için HEAD^ ya da HEAD~ kullanabiliriz. -o yamayı nerede oluşturacağımızı belirler. Bu komutu çalıştırdığımızda /tmp/ dosyası içerisinde yamalarımızı bulabiliriz.

Daha sonra, **mutt -H /tmp/0001-<yamanın-adı>** komutu ile yamamızı mail taslağı haline getirebiliriz.

Git send-email ile yama göndermek:
'git send-email' komutunu kullanarak da yama gönderebiliriz. Komuta, git format-patch komutu ile oluşturulan dosya verilebilir ya da git format-patch e verdiğimiz aynı commit ID sini vererek de işlem yapabiliriz. Örneğin;
$git send-email --annotate HEAD^.

Yamamızı gönderdikten sonra mail listelerinden geri bildirim aldığımızda bizden yamamızı güncelleyip tekrar göndermemiz istenebilir. Bu durumda yamamızı versiyon olarak belirtmeliyiz. Bunu yapmak gayet basit, 'git format-patch' komutuna --subject-prefix opsiyonu vermeliyiz. Örneğin;
$git format-patch --subject-prefix="PATCHv2"