

# distro Dokümanı

version

distro Linux

Ocak 07, 2024



# Contents

<b>Dağıtım</b>	<b>1</b>
Dağıtım Hazırlama	1
Dağıtım Nedir?	1
Dağıtım Nasıl Hazırlanmalı?	1
Ortam Hazırlama	2
Dağıtım İçin Ortamın Hazırlanması	2
Derleme Türleri	3
Derleme türleri	3
gcc libstdc++ Ekleme	3
Dynamic derleme	4
Static derleme	5
Static ve Dynamic derlemenin kıyaslanması	6
Kütüphane oluşturma	7
Kütüphaneyi Sisteme Dahil Etme ve Kullanma	7
Kütüphane Dosyasının Konumunu İsteğe Göre Belirleme(rpath)	8
Kütüphaneyi Uygulama İçine Gömme(Static Derleme)	8
Chroot Kullanımı	10
Chroot Nedir?	10
Bağımlılık Scripti	11
Basit Sistem Oluşturma	12
Sistem Dizinini Oluşturulması	12
ls Komutu	12
rmdir Komutu	12
mkdir Komutu	13
bash Komutu	13
chroot Sistemde Çalışma	13
Temel Paketleri Derleme	14
Temel Paketler	14
glibc Nedir?	15
glibc Derleme	15
glibc Yükleme	15
glibc Test Etme	15
Program Derleme	15
Program Yükleme	15
Programı Test Etme	15
Hata Çözümü	16
libreadline	17
libreadline Derleme	17
Program Derleme	17
Program Derleme	17
Program Test Etme	17

ncurses	18
ncurses Derleme	18
kmod	19
kmod Derleme	19
Kmod'u derleme için hazırlayalım:	19
kmod Araçlarını Oluşturma	20
kmod Test Edilmesi	20
util-linux	21
util-linux Derleme	21
eudev	22
eudev Derleme	22
busybox Nedir?	23
e2fsprogs Paketi	24
Grub Nedir?	25
grub Derleme	25
zlib Nedir?	26
<b>zlib Derleme</b>	26
Paket Sistemi Tasarlama	27
Paket Sitemi	27
bps Paket Sistemi	27
Paket Oluşturma	28
<b>bpsbuild</b> Dosyası	28
<b>bpspaketle</b> Dosyası	28
<b>bpsbuild</b> Dosyamızın Son Hali	29
<b>bpspaketle</b> Dosyamızın Son Hali	30
Paket Yapma	31
Paket Liste İndexi Güncelleme	32
<b>index.lst</b> Dosyasını Oluşturma	32
<b>index.lst</b> Dosyasını Güncelleme	32
İndirme Uygulaması	33
Paket Kurma	34
bpskur Scripti	34
bpskur Scriptini Kullanma	34
Paket Kaldırma	35
bpskaldır scripti	35
bpskaldır Kullanma	35
initrd Hazırlama	36
initrd	36
<b>Temel Dosyalar Ve Açılış Süreci</b>	36
<b>Bir linux sisteminin açılış süreci şu şekilde olmaktadır.</b>	36
<b>initrd Dosya İçeriği</b>	37
<b>initrd Scripti</b>	38
<b>S1- \$boot/bin/busybox</b>	38

<b>S2-S8 \$boot/bin/kmod</b>	39
<b>S9- \$boot/lib/modules/\$(uname -r)/moduller</b>	39
<b>S10-S13- \$boot/bin/udevadm</b>	39
<b>S14- distro/initrd/bin/init</b>	40
<b>init Dosyası</b>	40
<b>S15- distro/iso/initrd.img</b>	41
<b>S16- distro/iso/vmlinuz</b>	41
<b>S17- distro/iso/grub/grub.cfg</b>	41
Sistem Hazırlama	42
Dağıtım Hazırlama	42
<b>filesystem.squashfs Hazırlama</b>	42
İso Dosyasının Oluşturulması	43
Sistem Kurulumu	44
İki Bölüm Kurulum	44
Disk Hazırlanmalı	44
e2fsprogs Paketi	44
Dosya sistemini kopyalama	44
Bootloader kurulumu	45
Grub Kuralım	45
Grub yapılandırması	45
OpenRc Disk İşlemi	46
Fstab dosyası	46
Tek Bölüm Kurulum	47
Disk Hazırlanmalı(legacy)	47
Dosya sistemini kopyalama	47
Bootloader kurulumu	48
Grub Kuralım	48
Grub yapılandırması	48
OpenRc Disk İşlemi	49
Fstab dosyası	49
Uefi Sistem Kurulumu	50
Uefi - Legacy tespiti	50
Disk Hazırlanmalı	50
e2fsprogs Paketi	50
Dosya sistemini kopyalama	51
Bootloader kurulumu	51
Grub Kuralım	52
Grub yapılandırması	52
OpenRc Disk İşlemi	52
Fstab dosyası	52
Yardımcı Konular	53
Qemu Kullanımı	53
Qemu Nedir?	53

Sisteme Kurulum	53
Sistem Hızlandırılması	53
Boot Menu Açma	53
Uefi kurulum için:	54
qemu Host Erişimi:	54
Live Sistem Oluşturma	55
SquashFS Nedir?	55
SquashFS Oluşturma	55
Cdrom Erişimi	55
Cdrom Bağlama	55
squashfs Dosyasını Bulma	55
squashfs Bağlama	55
squashfs Sistemine Geçiş	55
busybox Nedir?	56
kmod Nedir?	56
Modul Yazma	57
hello.c dosyamız	57
Makefile dosyamız	57
modülün derlenmesi ve eklenip kaldırılması	57
Not:	58
sfdisk Nedir	58
Disk bölümlerini görüntüleme:	58
Disk bölümleri oluşturma:	58
Disk bölümlerini silme:	58
Bazı Örnekler	58
Örnek1:	58
Örnek2:	58
Örnek3:	59
Örnek4:	59
İmza Doğrulama	59
İmza Oluşturma	59
Belge İmzalama	59
İmzalı Belge Doğrulama	59
bash ile Doğrulama	59
c++ ile Doğrulama	61
c++ libgpgme ile Doğrulama	63
Kernel Modul Derleme	66
Kaynak Dosya İndirme	66
Kernel Ayarları	66
Kernel Derleme	66
Modul Derleme	66
Modül Yükleme	67
Servis Yönetimi	67

OpenRC	67
Kurulum	67
Çalıştırılması	68
Basit kullanım	68
Servis dosyası	68





# Dağıtım

## Dağıtım Hazırlama

### Dağıtım Nedir?

Linux kullanmaya başlayan kişilerin en çok karşılaştığı terimlerden birisi **dağıtım** kelimesidir. Dağıtım şirketi, grup veya ekiplerden oluşan kişiler tarafından paketler derlenerek veya hazırlanmış bir linux sisteminin çeşitli düzenlemeler yapılarak bir isim altında oluşturulan **Linux Sistemi**'ne verilen addır. Açık kaynak felsefesinde dağıtımlar ve uygulamalar belirli bir lisansla lisanslanarak yayınlanmaktadır. Genellikle lisansların bazı farklılıkları olsada "Al, kullan, değiştir ve kimseden izin almadan dağıt" şeklindedir. Lisanslamadaki bu felsefeden dolayı çok fazla dağıtım oluşmuş ve oluşmaya devam etmektedir.

Linux dağıtımları genelde **kernel** ve uygulamalardan oluşur. Kernel ve uygulamaların kodları github, gitlab vb. ortamlarda paylaşıldığı için sıfırdan bir dağıtımda oluşturmak mümkündür. Bunu oluştururken yasal olmayan hiçbir işlem yapmamış oluruz. Çünkü genel felsefe **"Al, kullan, değiştir ve kimseden izin almadan dağıt"** şeklinde olduğunu hatırlayalım.

Bu doküman basit seviyede bir dağıtım oluşturmak ve kurulabilir bir medya dosyası(iso dosya) nasıl hazırlanmak için bir rehber olacaktır. Bu dokümanı hazırlanmasında ve anlatılanları tecrübe ederek öğrenmeme katkısı olan **Turkman Linux** dağıtım ekibiden @sulincix(Ali Rıza KESKİN) ve Celaledin AKARSU'ya teşekkür ederim.

### Dağıtım Nasıl Hazırlanmalı?

Bir dağıtım hazırlamak için orta seviye linux komutları ve kavramları bilmeliyiz. Bu bağlamda bu dokümanı okurken yabancı olduğunuz terimleri araştırmanızı tavsiye ederim. Bir dağıtım için bilinmesi gereken konuları maddeler halinde şöyle sıralayabiliriz.

1. Dağıtım Ortamının Hazırlanması
2. Derleme ve Bağımlılık
3. chroot Nedir?
4. Temel Paketleri Derleme
5. Paket Sistemi Tasarlama
6. initrd Hazırlama
7. Sistemin Hazırlanması
8. İsonun Kurulması

Burada sıralanan maddeler konu başlıkları olarak anlatılacaktır.

## Ortam Hazırlama

### Dağıtım İçin Ortamın Hazırlanması

Dağıtım hazırlarken sistemin derlenmesi ve gerekli ayarlamaların yapılabilmesi için bir linux dağıtımı gerekmektedir. Tecrübeli olduğunuz bir dağıtımı seçmenizi tavsiye ederim. Fakat seçilecek dağıtım Gentoo olması daha hızlı ve sorunsuz sürece devam etmenizi sağlayacaktır. Bu dağıtımı hazırlarken Debian dağıtımı kullanıldı. Bazı paketler için, özellikle bağımlılık sorunları yaşanan paketler için ise Gentoo kullanıldı.

Bir dağıtım hazırlamak için çeşitli paketler lazımdır. Bu paketler;

- debootstrap : Dağıtım hazırlarken kullanılacak chroot uygulaması bu paket ile gelmektedir. chroot ayrı bir konu başlığıyla anlatılacaktır.
- make : Paket derlemek için uygulama
- squashfs-tools : Hazırladığımız sistemi sıkıştırılmış dosya halinde sistem görüntüsü oluşturmamızı sağlayan paket.
- gcc : c kodlarımızı derleyeceğimiz derleme aracı.
- wget : tarball vb. dosyaları indirmek için kullanılacak uygulama.
- unzip : Sıkıştırmış zip dosyalarını açmak için uygulama
- xz-utils : Yüksek sıkıştırma yapan sıkıştırma uygulaması
- tar : tar uzantılı dosya sıkıştırma ve açma için kullanılan uygulama.
- zstd : Yüksek sıkıştırma yapan sıkıştırma uygulaması
- grub-mkrescue : Hazırladığımız iso dizinini iso yapmak için kullanılan uygulama
- qemu-system-x86 : iso dosyalarını test etmek ve kullanmak için sanal makina emülatörü uygulaması.

```
sudo apt update  
sudo apt install debootstrap make squashfs-tools gcc wget unzip xz-utils tar zstd -y
```

Paket kurulumu yapıldıktan sonra kurulum için bir yeri(hedefi) belirlemeliyiz. Bu dokümanda sistem için kurulum dizini \$HOME/rootfs olarak kullanacağız.

## Derleme Türleri

### Derleme türleri

Paketler derlenirken farklı derleme araçları ve derleyiciler kullanılır. Bu bölümde kaynak kod derleme ile ilgili bilgiler verilecektir. Anlatım için C programlama dili ve gcc tercih edilecektir.

Bu bölümün amacı sizlere C kaynak kodlarının nasıl derlendiği ve nasıl çalıştığını anlatmaktır. C programlama dili ile ilgili yeterli bilginiz yoksa bu bölüme geçmeden önce bilgilerinizi gözden geçirmeyi öneririz.

Örneğin elimizde aşağıdaki gibi bir C dosyası bulunsun. Bu dosya derlenip bilgisayarın anlayabileceği hale getirilmek için **gcc** kullanılarak derlenmelidir.

```
//main.c dosyası
#include <stdio.h>
int main(){
    printf("Hello World\n");
}
```

Bu dosyayı derleyip çalıştırılabilir dosya elde edelim.

```
$ gcc -c main.c -o main.o
$ gcc -o main main.o
# .o oluşturmada doğrudan da derleyebilirsiniz.
$ gcc -o main main.c
```

Yukarıdaki örnekte kaynak kodu ilk önce **.o** uzantılı object dosyasına çevirdik. Daha sonra bu dosyadan çalıştırılabilir dosya ürettik.

Derlemeler **static** ve **dynamic** olarak 2 şekilde yapılabilir. Static olarak yapılan derleme herhangi bir bağımlılık olmaksızın çalışabilirken Dynamic olarak yapılmış derlemeler sistemdeki libc ve diğer gereken bağımlılıklara ihtiyaç duyar. static derleme boyut olarak daha büyüktür ve gerekli olan kütüphanelerin static hallerinin de bulunması gerekir.

### gcc libstdc++ Ekleme

gcc ile derlenen bazı dosyalarda **libstdc++** görülür. Bu kütüphaneyi static olarak dahil etmek için **-static-libstdc++** eklenmelidir.

```
$ gcc -o main main.c -static-libstdc++ #statik kütüphane dahil edildi
```

### Dynamic derleme

Dynamic olarak derlenen bir dosya düşük boyutludur ve bağımlılıkları bulunur. Derleme yapılırken ek parametre kullanılmaz.

```
$ gcc -o main main.c
```

Dynamic derlenmiş bir dosyanın bağımlılıklarını **ldd** komutu kullanarak öğrenebiliriz. Eğer ldd komutu hata mesajı ile geri dönüş veriyorsa static olarak derlenmiş demektir.

```
$ ldd /bin/bash
linux-vdso.so.1 (0x00007ffc8f136000)
libtinfo.so.6 => /lib/x86_64-linux-gnu/libtinfo.so.6 (0x00007ff10adcd000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007ff10adc7000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ff10ac02000)
/lib64/ld-linux-x86-64.so.2 (0x00007ff10af6c000)
```

Burada **libc.so.6** ve **ld-linux-x86\_64.so.2** dosyaları tamamında ortaktır ve **glibc** tarafından sağlanır. Dynamic derlenmiş bir dosyanın derlenmesi veya çalıştırılabilmesi için tüm bağımlılıklarının sistemde bulunması gereklidir.

## Derleme Türleri

### Static derleme

Static olarak derlenmiş bir dosya bağımlılığa sahip değildir. Initramfs gibi yerlerde kullanmak için uygundur. Bir kodu static olarak derlemek için **-static** parametresi kullanılır. Bu parametre ihtiyaç duyulan kütüphanelerin derlenmiş olan dosyaya gömülmesini sağlar.

```
$ gcc -o main main.c -static
```

Bir dosyanın static olup olmadığını anlamak için **ldd** komutunun hata mesajı vermesine bakılabilir.

```
$ ldd main  
not a dynamic executable
```

### Static ve Dynamic derlemenin kıyaslanması

Static olarak derlenmiş bir dosya sistemden bağımsız olarak çalışabilir. Örneğin elimizde aşağıdaki gibi bir kod olsun.

```
//main.c dosyası
#include <stdio.h>
int main(){
    printf("Hello world\n");
}
```

Bu kodu static ve dynamic olarak 2 farklı şekilde derleyip chroot içine atıp çalıştırmayı deneyelim.

```
$ mkdir chroot
$ gcc -o chroot/main main.c
$ chroot chroot /main
chroot: failed to run command '/main': No such file or directory
$ gcc -o chroot/main main.c -static
$ chroot chroot /main
Hello world
```

Gördüğümüz gibi dynamic olarak derlenmiş dosya libc bulamadığı için çalışmadı. Fakat static olarak derlenmiş dosyamız çalıştı. Bununla birlikte dosya boyutlarını aşağıdaki gibi kıyaslayabiliriz.

```
$ gcc -o main.dynamic main.c
$ gcc -o main.static main.c -static
$ du main*
 4    main.c
20    main.dynamic
768   main.static
```

Gördüğümüz gibi dynamic olarak derlenmiş dosya boyut olarak çok daha küçüktür. Bu yüzden sistem içerisinde genellikle dynamic derlemeler tercih edilirken, initramfs gibi yerlerde static derleme tercih edilir.

### Kütüphane oluşturma

Kütüphane dosyaları **so** uzantılıdır ve ihtiyaç duyulan diğer yazılımlar tarafından kullanılır. Kütüphane oluşturmak için öncelikle aşağıdaki gibi bir C kodumuz olsun.

```
//deneme.c dosyası
#include <stdio.h>
void yazdir(){
    printf("Merhaba Dünya\n");
}
```

Bu dosyayı doğrudan derlersek **main** fonksiyonu bulunmadığı için aşağıdaki gibi bir hata ile karşılaşırız.

```
$ gcc deneme.c -o libdeneme.so
...: in function `__start':
(.text+0x17): undefined reference to main'
collect2: error: ld returned 1 exit status
```

Kütüphane derlemek için aşağıdaki iki komutu arka arkaya kullanmalıyız.

İlk satır **denem.o** dosyası oluşturacaktır. İkinci satırımızda **-shared** parametresi kullanarak **main** bulunmayan kütüphane dosyamız derlendi ve **deneme.so** dosyası oluşturulmuş olur.

```
$ gcc -c -Wall -Werror -fpic deneme.c
$ gcc -shared -o libdeneme.so deneme.o
```

Kütüphane aşağıdaki gibi kullanılabilir.

```
extern void yazdir();
int main(){
    yazdir();
}
```

Şimdi bu kütüphaneyi başka bir kodu derlemek için kullanalım. Bunun için **-L** parametresi ile kütüphanenin bulunduğu yeri göstermeliyiz. **-l** parametresi ile de kütüphaneyi bağlamalıyız.

```
$ gcc -L/ders/kutuphane -o main main.c -ldeneme
```

### Kütüphaneyi Sisteme Dahil Etme ve Kullanma

Yukarıdaki örnekte /ders/kutuphane/libdeneme.so dosyasını kullandık.

Artık derlediğimiz main adındaki ikili dosyayı çalıştırabiliriz. Çalıştırdığımızda aşağıdaki gibi hatayla karşılaşırız. libdeneme.so dosyasını bulamadığını söylüyor.

```
$ ./main
./main: error while loading shared libraries: libdeneme.so: cannot open shared object file: No such file or directory
```

Bu hatayı daha net anlamak için ve **main** ikili dosyamızın hangi dosyalara ihtiyacı olduğunu görmek için aşağıdaki komutu çalıştırırız. **main** ikili dosyasının çalışması için 4 tane dosyaya ihtiyacı var. Bunlardan birisi bizim oluşturduğumuz **libdeneme.so** dosyası. Fakat dosyayı bulamadığını söylüyor. Aslında kütüphaneyi **/usr/lib/libdeneme.so** konumunda olup olmadığını göre bu mesajı veriyor.

```
$ ldd ./main
linux-vdso.so.1 (0x00007ffffab5e8000)
libdeneme.so => not found
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ff73002c000)
/lib64/ld-linux-x86-64.so.2 (0x00007ff73020d000)
```

**libdeneme.so** dosyamızı **/usr/lib/** konumuna kopyalayalım. Erişim izni verelim. **ldd ./main** komutunu çalıştırdığımızda artık dosyanın karşısında **not found** mesajı yok. Artık çalışacaktır.

```
$ sudo cp /ders/kutuphane/libdeneme.so /usr/lib
$ sudo chmod 0755 /usr/lib/libdeneme.so
$ ldd ./main
linux-vdso.so.1 (0x00007ffdf93fc000)
libdeneme.so => /lib/libdeneme.so (0x00007fa5c281d000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fa5c2658000)
/lib64/ld-linux-x86-64.so.2 (0x00007fa5c283e000)
$ ./main
Merhaba Dünya
```

### Kütüphane Dosyasının Konumunu İsteğe Göre Belirleme(rpath)

Bazen kütüphane dosyalarının **/usr/lib** konumunda değilde bizim belirleyeceğimiz konumda olmasını isteyebiliriz.

- Örneğin **/opt/main/** konumunda olmasını istersek aşağıdaki gibi yapmalıyız.
- Hatasız bir sonuç almak için öncelikle **/usr/lib/libdeneme.so** konumundaki dosyamızı silelim.
- Daha sonra **/opt/main/** konumunda olacak şekilde main ikili dosyamızı derleyelim.
- Eğer **/opt/main** klasörü yoksa oluşturmalıyız. Ben olmadığımı varsayıyorum ve oluşturuyorum.
- **libdeneme.so** dosyamızıda **/usr/lib/libdeneme.so** konumuna kopyalayıp izinlerini ayarlayalım.
- Son işlem olarak test edelim.

Bunun için;

```
$ sudo rm /usr/lib/libdeneme.so
$ gcc -L/ders/kutuphane -Wl,-rpath=/opt/main -Wall -o main main.c -ldeneme
$ sudo mkdir /opt/main
$ sudo cp /ders/kutuphane/libdeneme.so /opt/main/
$ sudo chmod 0755 /opt/main/libdeneme.so
$ ./main
Merhaba Dünya
```

### Kütüphaneyi Uygulama İçine Gömmek(Static Derleme)

Bazı durumlarda ise kütüphane dosyalarını proje içine gömmek isteyebiliriz. **main** uygulamamız bağımlılığı olmayan bir uygulama yapabiliriz. Bunun için;

```
$ gcc -c -Wall -Werror -fpic deneme.c
$ gcc -c main.c
```



## Derleme Türleri

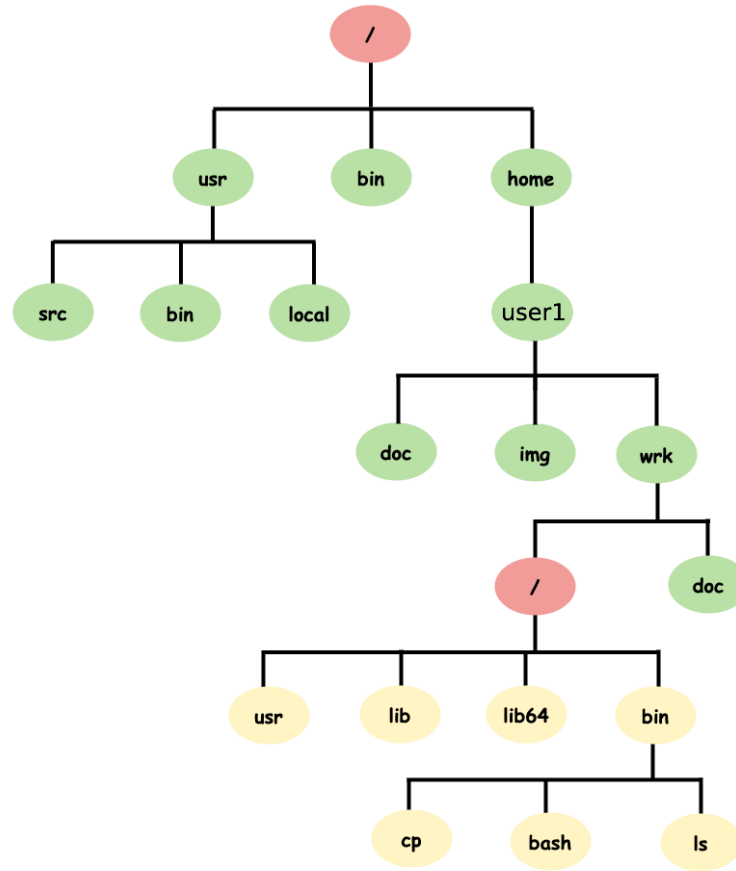
```
$ gcc main.o ./deneme.o -o main -static  
$ ldd ./main  
özdevimli bir çalıştırılabilir değil  
$ ./main  
Merhaba Dünya
```

## Chroot Kullanımı

### Chroot Nedir?

chroot komutu çalışan sistem üzerinde belirli bir klasöre root yetkisi verip sadece o klasörü sanki linux sistemi gibi çalıştıran bir komuttur. Sağladığı avantajlar çok fazladır. Bunlar;

- Sistem tasarlama
- Sistem üzerinde yeni dağıtımlara müdahale etme ve sorun çözme
- Kullanıcı kendine özel geliştirme ortamı oluşturabilir.
- Yazılım bağımlılıkları sorunlarına çözüm olabilir.
- Kullanıcıya sadece kendisine verilen alanda sınırsız yetki verme vb.



Yukarıdaki resimde user1 altında wrk dizini altına yeni bir sistem kurulmuş gibi yapılandırmayı gerçekleştirmiş.

**/home/user1/wrk** dizinindeki sistem üzerinde sisteme erişmek için;

```
sudo chroot /home/user1/wrk #sisteme erişim yapıldı.
```

**/home/user1/wrk** dizinindeki sistem üzerinde sistemi silmek için;

```
sudo rm -rf /home/user1/wrk #sistem silindi
```

## Chroot Kullanımı

Yeni sistem tasarlamak ve erişmek için temel komutları ve komut yorumlayıcısının olması gerekmektedir. Bunun için bize gerekli olan komutları bu yapının içine koymamız gerekmektedir. Örneğin ls komutu için doğrudan çalışıp çalışmadığını ldd komutu ile kontrol edelim.

```
etapadmin@etahta: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
etapadmin@etahta:~$ ldd /bin/ls  
linux-vdso.so.1 (0x00007ffc68471000)  
libgtk3-nocsd.so.0 => /usr/lib/x86_64-linux-gnu/libgtk3-nocsd.so.0 (0x00007ff3fa9db000)  
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007ff3fa9ad000)  
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ff3fa7cc000)  
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007ff3fa7c7000)  
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007ff3fa7c2000)  
libpcre2-8.so.0 => /usr/lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007ff3fa726000)  
/lib64/ld-linux-x86-64.so.2 (0x00007ff3faa2a000)  
etapadmin@etahta:~$
```

Görüldüğü gibi ls komutunun çalışması için bağımlı olduğu kütüphane dosyaları bulunmaktadır. Bağımlı olduğu dosyaları yeni oluşturduğumuz sistem dizinine aynı dizin yapısında kopyalamamız gerekmektedir. Bu dosyalar eksiksiz olursa ls komutu çalışacaktır. Fakat bu işlemi tek tek yapmamız çok zahmetli bir işlemdir. Bu işi yapacak script dosyası aşağıda verilmiştir.

## Bağımlılık Scripti

lddscript.sh

```
#!/bin/bash  
  
if [ $# != 2 ]  
then  
    echo "usage $0 PATH_TO_BINARY target_folder"  
    exit 1  
fi  
path_to_binary="$1"  
target_folder="$2"  
  
# if we cannot find the the binary we have to abort  
if [ ! -f "${path_to_binary}" ]  
then  
    echo "The file '${path_to_binary}' was not found. Aborting!"  
    exit 1  
fi  
  
echo "---> copy binary itself" # copy the binary itself  
cp --parents -v "${path_to_binary}" "${target_folder}"  
  
echo "---> copy libraries" # copy the library dependencies  
ldd "${path_to_binary}" | awk -F'[> ]' '{print $(NF-1)}' | while read -r lib  
do  
    [ -f "$lib" ] && cp -v --parents "$lib" "${target_folder}"  
done
```

## Chroot Kullanımı

### Basit Sistem Oluşturma

Bu örnekte kullanıcının(etapadmin) ev dizinine(/home/etapadmin) test dizini oluşturuldu ve işlemler yapıldı. ls, rmdir, mkdir ve bash komutlarından oluşan sistem hazırlama.

### Sistem Dizin Oluşturulması

```
mkdir /home/etapadmin/test/ #ev dizinine test dizini oluşturuldu.
```

/home/etapadmin/ dizinine **Bağımlılık Scripti** kodunu **lddscripts.sh** oluşturalım.

### ls Komutu

```
bash lddscripts.sh /bin/ls /home/etapadmin/test/ #komutu ile ls komutunu ve bağımlılığını kopyalandı.
```

```
etapadmin@etahta: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
etapadmin@etahta:~$ bash lddscripts.sh /bin/ls /home/etapadmin/test/  
---> copy binary itself  
/bin -> /home/etapadmin/test/bin  
'/bin/ls' -> '/home/etapadmin/test/bin/ls'  
---> copy libraries  
/usr -> /home/etapadmin/test/usr  
/usr/lib -> /home/etapadmin/test/usr/lib  
/usr/lib/x86_64-linux-gnu -> /home/etapadmin/test/usr/lib/x86_64-linux-gnu  
'/usr/lib/x86_64-linux-gnu/libgtk3-nocsd.so.0' -> '/home/etapadmin/test/usr/lib/x86_64-linux-gnu/libgtk3-nocsd.so.0'  
/lib -> /home/etapadmin/test/lib  
/lib/x86_64-linux-gnu -> /home/etapadmin/test/lib/x86_64-linux-gnu  
'/lib/x86_64-linux-gnu/libselinux.so.1' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libselinux.so.1'  
'/lib/x86_64-linux-gnu/libc.so.6' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libc.so.6'  
'/lib/x86_64-linux-gnu/libdl.so.2' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libdl.so.2'  
'/lib/x86_64-linux-gnu/libpthread.so.0' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libpthread.so.0'  
'/usr/lib/x86_64-linux-gnu/libpcres2-8.so.0' -> '/home/etapadmin/test/usr/lib/x86_64-linux-gnu/libpcres2-8.so.0'  
/lib64 -> /home/etapadmin/test/lib64  
'/lib64/ld-linux-x86-64.so.2' -> '/home/etapadmin/test/lib64/ld-linux-x86-64.so.2'  
etapadmin@etahta:~$
```

Bu işlemi diğer komutlar içinde sırasıyla yapmamız gerekmektedir.

### rmdir Komutu

```
bash lddscripts.sh /bin/rmdir /home/etapadmin/test/ #komutu ile rmdir komutunu ve bağımlılığını kopyalandı.
```

```
etapadmin@etahta: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
etapadmin@etahta:~$ bash lddscripts.sh /bin/rmdir /home/etapadmin/test/  
---> copy binary itself  
'/bin/rmdir' -> '/home/etapadmin/test/bin/rmdir'  
---> copy libraries  
'/usr/lib/x86_64-linux-gnu/libgtk3-nocsd.so.0' -> '/home/etapadmin/test/usr/lib/x86_64-linux-gnu/libgtk3-nocsd.so.0'  
'/lib/x86_64-linux-gnu/libc.so.6' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libc.so.6'  
'/lib/x86_64-linux-gnu/libdl.so.2' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libdl.so.2'  
'/lib/x86_64-linux-gnu/libpthread.so.0' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libpthread.so.0'  
'/lib64/ld-linux-x86-64.so.2' -> '/home/etapadmin/test/lib64/ld-linux-x86-64.so.2'  
etapadmin@etahta:~$
```

## Chroot Kullanımı

### mkdir Komutu

`bash lddscripts.sh /bin/mkdir /home/etapadmin/test/` #komutu ile mkdir komutunu ve bağımlılığını kopyalandı.

```
etapadmin@etahta: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
etapadmin@etahta:~$ bash lddscripts.sh /bin/mkdir /home/etapadmin/test/  
---> copy binary itself  
'/bin/mkdir' -> '/home/etapadmin/test/bin/mkdir'  
---> copy libraries  
'/usr/lib/x86_64-linux-gnu/libgtk3-nocsd.so.0' -> '/home/etapadmin/test/usr/lib/x86_64-linux-gnu/libgtk3-nocsd.so.0'  
'/lib/x86_64-linux-gnu/libselinux.so.1' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libselinux.so.1'  
'/lib/x86_64-linux-gnu/libc.so.6' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libc.so.6'  
'/lib/x86_64-linux-gnu/libdl.so.2' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libdl.so.2'  
'/lib/x86_64-linux-gnu/libpthread.so.0' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libpthread.so.0'  
'/usr/lib/x86_64-linux-gnu/libpcre2-8.so.0' -> '/home/etapadmin/test/usr/lib/x86_64-linux-gnu/libpcre2-8.so.0'  
'/lib64/ld-linux-x86-64.so.2' -> '/home/etapadmin/test/lib64/ld-linux-x86-64.so.2'  
etapadmin@etahta:~$
```

### bash Komutu

`bash lddscripts.sh /bin/bash /home/etapadmin/test/` #komutu ile bash komutunu ve bağımlılığını kopyalandı.

```
etapadmin@etahta: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
etapadmin@etahta:~$ bash lddscripts.sh /bin/bash /home/etapadmin/test/  
---> copy binary itself  
'/bin/bash' -> '/home/etapadmin/test/bin/bash'  
---> copy libraries  
'/usr/lib/x86_64-linux-gnu/libgtk3-nocsd.so.0' -> '/home/etapadmin/test/usr/lib/x86_64-linux-gnu/libgtk3-nocsd.so.0'  
'/lib/x86_64-linux-gnu/libtinfo.so.6' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libtinfo.so.6'  
'/lib/x86_64-linux-gnu/libc.so.6' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libc.so.6'  
'/lib/x86_64-linux-gnu/libdl.so.2' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libdl.so.2'  
'/lib/x86_64-linux-gnu/libpthread.so.0' -> '/home/etapadmin/test/lib/x86_64-linux-gnu/libpthread.so.0'  
'/lib64/ld-linux-x86-64.so.2' -> '/home/etapadmin/test/lib64/ld-linux-x86-64.so.2'  
etapadmin@etahta:~$
```

### chroot Sistemde Çalışma

`sudo chroot /home/etapadmin/test` komutunu kullanmalıyız.

```
etapadmin@etahta: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
etapadmin@etahta:~$ sudo chroot /home/etapadmin/test  
[sudo] password for etapadmin:  
bash-5.2# ls  
bin lib lib64 usr  
bash-5.2# mkdir abc  
bash-5.2# ls  
abc bin lib lib64 usr  
bash-5.2# rmdir abc  
bash-5.2# ls  
bin lib lib64 usr  
bash-5.2# pwd  
/  
bash-5.2# ldd  
bash: ldd: command not found  
bash-5.2# exit  
exit  
etapadmin@etahta:~$
```

- **abc** dizini oluşturuldu.
- **abc** dizini silindi.
- **pwd** komutuyla konum öğrenildi.
- **ldd** komutu sistemimizde olmadığından hata verdi.
- Çıkış için ise **\*exit\*** komutu kullanılarak sistemden çıkıldı.

Kaynak:

<https://stackoverflow.com/questions/64838052/how-to-delete-n-characters-appended-to-ldd-list>

## Temel Paketleri Derleme

### Temel Paketler

Dağıtım temel seviyede kullanıcıya tty ortamı sunan bir yapıdan oluşacak. Ayrıca kendini kurup grubu yükleyecek bir yapıda olmasını planlamaktayız. Bu yapıda bir dağıtım için aşağıdaki paketlere ihtiyacımız olacak.

Bunlar;

#### 1. **glibc**

- **readline**
  - bash
- **ncurses**
  - bash
- **zlib**
  - kmod
- **xz-utils**
  - kmod
- util-linux
- eudev
- busybox
- e2fsprogs
- grub

Paket listemizde **glibc** tüm paketlerin ihtiyaç duyacağı kütüphaneleri sağlayan pakettir.

Örneğin listede **bash** uygulamasının çalışabilmesi için **readline** ve **ncurses** kütüphaneleri gerekli. **readline** ve **ncurses** kütüphanelerinin çalışabilmesi içinde **glibc** kütüphanesi gerekli. Bash paketinin bağımlı olduğu kütüphaneler geriye doğru takip ederek listelenir.

Sonuç olarak bash paketini derlerken paketin;

- name="bash"
- version="x.x.x"
- depends="glibc,readline,ncurses" şeklinde temel bilgilerini belirterek paketler yapacağız.

Bu sayede paketimizin çalışabilmesi için temel bilgileri belirlemiş oluyoruz. Bu bilgileri paketi derlerken belirteceğiz. Bu temel bilgiler paketin dağıtımına kurulması, kaldırılması, bağımlılık ve bağımlılık çakışmalarının tespitinde kullanılacak.

Listede bulunan tüm paketlerin hepsinde burada anlatılan bağımlılık tespiti hatasız yapılmalıdır. Burada tüm paketlerin derlenmesinde izlenmesi gereken işlem adımlarını **bash** ve **kmod** paketleri özelinde anlatılmaya çalışıldı.

**glibc** dağıtımda sistemdeki bütün uygulamaların çalışmasını sağlayan en temel C kütüphanesidir. GNU C Library(glibc)'den farklı diğer C standart kütüphaneler şunlardır: Bionic libc, dietlibc, EGLIBC, klibc, musl, Newlib ve uClibc. **glibc** yerine alternatif olarak çeşitli avantajlarından dolayı kullanılabilir. **glibc** en çok tercih edilen ve uygulama (özgür olmayan) uyumluluğu bulunduğu için bu dokümanda glibc üzerinden anlatım yapılacaktır.

Listede bulunan paketler sırasıyla nasıl derleneceği ayrı başlıklar altında anlatılacaktır.

## Temel Paketleri Derleme

### glibc Nedir?

glibc (GNU C Kütüphane) Linux sistemlerinde kullanılan bir C kütüphanesidir. Bu kütüphane, C programlama dilinin temel işlevlerini sağlar ve Linux çekirdeğiyle etkileşimde bulunur.

glibc, birçok standart C işlevini içerir ve bu işlevler, bellek yönetimi, dosya işlemleri, dize işlemleri, ağ işlemleri ve daha fazlası gibi çeşitli görevleri yerine getirmek için kullanılabilir. Bu kütüphane, Linux sistemlerinde yazılım geliştirme sürecini kolaylaştırır ve programcılara güçlü bir araç seti sunar.

glibc, Linux sistemlerinde C programlama dilini kullanarak yazılım geliştirmek için önemli bir araçtır. Bu kütüphane, Linux'ta çalışan birçok programın temelini oluşturur ve geliştiricilere güçlü bir platform sunar.

### glibc Derleme

```
cd $HOME/ # Ev dizinine geçiyorum.
wget https://ftp.gnu.org/gnu/libc/glibc-2.38.tar.gz # glibc kaynak kodunu indiriyoruz.
tar -xvf glibc-2.38.tar.gz # glibc kaynak kodunu açıyoruz.
mkdir build-glibc && cd build-glibc # glibc derlemek için bir derleme dizini oluşturuyoruz.
../glibc-2.38/configure --prefix=/ --disable-werror # Derleme ayarları yapılıyor
make # glibc derleyelim.
```

### glibc Yükleme

**\$HOME/rootfs** kalsörünü oluşturudan aşağıdaki gibi yükleme yapmalıyız.

```
make install DESTDIR=$HOME/rootfs # Ev Dizinindeki rootfs dizinine glibc yükleyelim.
```

### glibc Test Etme

glibc kütüphanemizi **\$HOME/rootfs** komununa yükledik. Şimdi bu kütüphanenin çalışıp çalışmadığını test edelim.

Aşağıdaki c kodumuzu derleyelim ve **\$HOME/rootfs** konumuna kopyalayalım.

```
#include<stdio.h>
void main()
{
puts("Merhaba Dünya");
}
```

### Program Derleme

```
gcc -o merhaba merhaba.c
```

### Program Yükleme

Derlenen çalışabilir merhaba dosyamızı **glibc** kütüphanemizin olduğu dizine yükleyelim.

```
cp merhaba $HOME/rootfs/merhaba # derlenen merhaba ikili dosyası $HOME/rootfs/ konumuna kopyalandı.
```

### Programı Test Etme

**glibc** kütüphanemizin olduğu dizin dağıtımızın ana dizini oluyor. **\$HOME/rootfs/** konumuna **chroot** ile erişelim.

Aşağıdaki gibi çalıştırdığımızda bir hata alacağız.

## Temel Paketleri Derleme

```
sudo chroot $HOME/rootfs/ /merhaba
chroot: failed to run command '/merhaba': No such file or directory
```

### Hata Çözümü

```
# üstteki hatanın çözümü sembolik bağ oluşturmak.
cd $HOME/rootfs/
ln -s lib lib64
```

#merhaba dosyamızı tekrar chroot ile çalıştıralım. Aşağıda görüldüğü gibi hatasız çalışacaktır.

```
sudo chroot rootfs /merhaba
Merhaba Dünya
```

**Merhaba Dünya** mesajını gördüğümüzde glibc kütüphanemizin ve merhaba çalışabilir dosyamızın çalıştığını anlıyoruz. Bu aşamadan sonra **Temel Paketler** listemizde bulunan paketleri kodlarından derleyerek **\$HOME/rootfs/** dağıtım dizinimize yüklemeliyiz. Derlemede **glibc** kütüphanesinin derlemesine benzer bir yol izlenecektir. **glibc** temel kütüphane olması ve ilk derlediğimiz paket olduğu için detaylıca anlatılmıştır.

**glibc** kütüphanemizi derlerken yukarıda yapılan işlem adımlarını ve hata çözümlemesini bir script dosyasında yapabiliriz. Bu dokümanda altta paylaşılan script dosyası yöntemi tercih edildi.

```
# kaynak kod indirme ve derleme için hazırlama
version="2.38"
name="glibc"
mkdir -p $HOME/distro
cd $HOME/distro
rm -rf ${name}-${version}
rm -rf build-${name}-${version}
wget https://ftp.gnu.org/gnu/libc/${name}-${version}.tar.gz
tar -xvf ${name}-${version}.tar.gz
mkdir build-${name}-${version}
cd build-${name}-${version}
../${name}-${version}/configure --prefix=/ --disable-werror

# derleme
make

# derlenen paketin yüklenmesi ve ayarlamaların yapılması
make install DESTDIR=$HOME/rootfs
cd $HOME/rootfs/
ln -s lib lib64
```

Diğer paketlerimizde de **glibc** için paylaşılan script dosyası gibi dosyalar hazırlayıp derlenecektir.



## Temel Paketleri Derleme

### libreadline

libreadline, Linux işletim sistemi için geliştirilmiş bir kütüphanedir. Bu kütüphane, kullanıcıların komut satırında girdi almasını ve düzenlemesini sağlar. Bir programcı olarak, libreadline'i kullanarak kullanıcı girdilerini okuyabilir, düzenleyebilir ve işleyebilirsiniz.

#### libreadline Derleme

```
# kaynak kod indirme ve derleme için hazırlama
version="8.1"
name="readline"
mkdir -p $HOME/distro
cd $HOME/distro
rm -rf ${name}-${version}
rm -rf build-${name}-${version}
wget https://ftp.gnu.org/pub/gnu/readline/${name}-${version}.tar.gz
tar -xvf ${name}-${version}.tar.gz
mkdir build-${name}-${version}
cd build-${name}-${version}
../${name}-${version}/configure --prefix=/ --enable-shared --enable-multibyte

# derleme
make

# derlenen paketin yüklenmesi ve ayarlamaların yapılması
make install DESTDIR=$HOME/rootfs
```

#### Program Derleme

Altta görülen **readline** kütüphanesini kullanarak terminalde kullanıcıdan mesaj alan ve mesajı ekrana yazan programı hazırladık.

```
# merhaba.c doyası
#include<stdio.h>
#include<readline/readline.h>
void main()
{
char* msg=readline("Adınızı Yaz:");
puts(msg);
}
```

#### Program Derleme

```
gcc -o merhaba merhaba.c -lreadline
cp merhaba $HOME/rootfs/merhaba
```

#### Program Test Etme

```
sudo chroot $HOME/rootfs /merhaba
```

Program hatasız çalışıyorsa **readline** kütüphanemiz hatasız derlenmiş olacaktır.

## Temel Paketleri Derleme

### ncurses

ncurses, Linux işletim sistemi için bir programlama kütüphanesidir. Bu kütüphane, terminal tabanlı kullanıcı arayüzleri oluşturmak için kullanılır. ncurses, terminal ekranını kontrol etmek, metin tabanlı menüler oluşturmak, renkleri ve stil özelliklerini ayarlamak gibi işlemlere sahiptir.

ncurses, kullanıcıya metin tabanlı bir arayüz sağlar ve terminal penceresinde çeşitli işlemler gerçekleştirmek için kullanılabilir. Örneğin, bir metin düzenleyici, dosya tarayıcısı veya metin tabanlı bir oyun gibi uygulamalar ncurses kullanarak geliştirilebilir.

### ncurses Derleme

```
# kaynak kod indirme ve derleme için hazırlama
version="6.4"
name="ncurses"
mkdir -p $HOME/distro
cd $HOME/distro
rm -rf ${name}-${version}
rm -rf build-${name}-${version}
wget https://ftp.gnu.org/pub/gnu/ncurses/${name}-${version}.tar.gz
tar -xvf ${name}-${version}.tar.gz
mkdir build-${name}-${version}
cd build-${name}-${version}
../${name}-${version}/configure --prefix=/ --with-shared --disable-tic-depends --with-versioned-syms --enable-widec
# derleme
make

# derlenen paketin yüklenmesi ve ayarlamaların yapılması
make install DESTDIR=$HOME/rootfs
cd $HOME/rootfs/lib
ln -s libncursesw.so.6 libtinfo.so.6
ln -s libncursesw.so.6 libtinfo.so.6
ln -s libncursesw.so.6 libncurses.so.6
```

## Temel Paketleri Derleme

### kmod

Linux çekirdeği ile donanım arasındaki haberleşmeyi sağlayan kod parçalarıdır. Bu kod parçalarını kernele eklediğimizde kerneli tekrardan derlememiz gerekmektedir. Her kod ekleme ve her kod çıkartma işleminden sonra kernel derlemek ciddi bir iş yükü ve karmaşa oluşturacaktır.

Bu sorunların çözümü için modul vardır. Moduller kernele istediğimiz kod parçalarını ekleme ya da çıkartma yapabilmemizi sağlar. Bu işlemleri yaparken kernel derleme işlemi yapmamıza gerek yoktur.

### kmod Derleme

```
# kaynak kod indirme ve derleme için hazırlama
version="31"
name="kmod"
mkdir -p $HOME/distro
cd $HOME/distro
rm -rf ${name}-${version}
rm -rf build-${name}-${version}
wget https://mirrors.edge.kernel.org/pub/linux/utils/kernel/kmod/${name}-${version}.tar.xz
tar -xvf ${name}-${version}.tar.xz
mkdir build-${name}-${version}
cd build-${name}-${version}
../${name}-${version}/configure --prefix=/ \
    --libdir=/lib/ \
    --bindir=/sbin
# remove xsltproc dependency
rm -f man/Makefile
echo -e "all:\ninstall:" > man/Makefile

# derleme
make

# derlenen paketin yüklenmesi ve ayarlamaların yapılması
make install DESTDIR=$HOME/rootfs
cd $HOME/rootfs/sbin
for target in depmod insmod modinfo modprobe rmmod; do
    ln -sfv kmod $target
done
cd $HOME/rootfs/bin
ln -sfv ../sbin/kmod lsmod
```

Kmod'u derleme için hazırlayalım:

```
./configure --prefix=/usr \
            --sysconfdir=/etc \
            --with-openssl \
            --with-xz \
            --with-zstd \
            --with-zlib
```

İsteğe bağlı bağımlılıklar: - ZLIB kütüphanesi - LZMA kütüphanesi -ZSTD kütüphanesi - OPENSSL kütüphanesi (modinfo'da imza yönetimi) --with-openssl Bu seçenek Kmod'un PKCS7 imzalarını işlemesini sağlar. çekirdek modülleri. --with-xz, --with-zlib, Ve --with-zstd Bu seçenekler Kmod'un sıkıştırılmış çekirdeği işlemesini sağlar modüller.

## Temel Paketleri Derleme

Bu dokümanda aşağıdaki şekilde yapılandırılacak;

```
./configure --prefix=/ \
    --libdir=/lib/ \
    --bindir=/sbin

# remove xsltproc dependency
rm -f man/Makefile
echo -e "all:\ninstall:" > man/Makefile
```

### kmod Araçlarını Oluşturma

```
for target in depmod insmod modinfo modprobe rmmod; do
    ln -sfv sbin/kmod sbin/$target
done

ln -sfv sbin/kmod bin/lsmmod
```

veya kernele modul yükleme kaldırma için kmod aracı kullanılmaktadır. kmod aracı;

```
ln -s sbin/kmod sbin/depmod
ln -s sbin/kmod sbin/insmod
ln -s sbin/kmod sbin/lsmmod
ln -s sbin/kmod sbin/modinfo
ln -s sbin/kmod sbin/modprobe
ln -s sbin/kmod sbin/rmmod
```

şeklinde sembolik bağlarla yeni araçlar oluşturulmuştur.

- **lsmod** : yüklü modulleri listeler
- **insmod**: tek bir modul yükler
- **rmmod**: tek bir modul siler
- **modinfo**: modul hakkında bilgi alınır
- **modprobe**: insmod komutunun aynısı fakat daha işlevseldir. module ait bağımlı olduğu modülleride yüklemektedir. modprobe modülü /lib/modules/ dizini altında aramaktadır.
- **depmod**: /lib/modules dizinindeki modüllerin listesini günceller. Fakat başka bir dizinde ise basedir=konum şeklinde belirtmek gerekir. konum dizininde /lib/modules/\*\* şeklinde kalsörler olmalıdır.

### kmod Test Edilmesi

Bir modül eklendiğinde veya çıkartıldığında modülle ilgili mesajları dmesg logları ile görebiliriz.

## Temel Paketleri Derleme

### util-linux

util-linux, Linux işletim sistemi için bir dizi temel araç ve yardımcı programları içeren bir pakettir. Bu araçlar, Linux'un çeşitli yönlerini yönetmek ve kontrol etmek için kullanılır.

util-linux paketi, birçok farklı işlevi yerine getiren bir dizi komut satırı aracını içerir. Örneğin, disk bölümlerini oluşturmak ve yönetmek için kullanılan **fdisk**, disklerdeki dosya sistemlerini kontrol etmek için kullanılan **fsck**, sistem saatini ayarlamak , sistem performansını izlemek ve yönetmek için kullanılan araçları da içerir. Örneğin, **top** komutu, sistemdeki işlemci kullanımını izlemek için kullanılırken, **free** komutu, sistem belleği kullanımını gösterir. Tarih ve saat gösterimi için kullanılan **date** gibi araçlar bu paketin bir parçasıdır.

### util-linux Derleme

```
#https://www.linuxfromscratch.org/lfs/view/development/chapter07/util-linux.html
version="2.39"
name="util-linux"
mkdir -p $HOME/distro
cd $HOME/distro
rm -rf ${name}-${version}
rm -rf build-${name}-${version}
wget https://mirrors.edge.kernel.org/pub/linux/utils/util-linux/v2.39/${name}-${version}.tar.xz
tar -xvf ${name}-${version}.tar.xz
#cd $HOME/distro/${name}-${version}
#sed -i 's/(link_all_deplibs)=no/\1=unknown/'
mkdir build-${name}-${version}
cd build-${name}-${version}

../${name}-${version}/configure --prefix=/ \
    --libdir=/lib \
    --bindir=/bin \
    --enable-shared \
    --disable-su \
    --disable-runuser \
    --disable-chfn-chsh \
    --disable-login \
    --disable-sulogin \
    --disable-makeinstall-chown \
    --disable-makeinstall-setuid \
    --disable-pylibmount \
    --disable-raw \
    --without-systemd \
    --without-libuser \
    --without-utempter \
    --without-econf \
    --enable-libmount \
    --enable-libblkid

make
make install DESTDIR=$HOME/rootfs
mkdir -p $HOME/rootfs/lib
cp .libs/* -rf $HOME/rootfs/lib/
mkdir -p $HOME/rootfs/bin
cp $HOME/rootfs/lib/cfdisk $HOME/rootfs/bin/
```

### eudev

eudev, Linux işletim sistemlerinde donanım aygıtlarının tanınması ve yönetimi için kullanılan bir sistemdir. "eudev" terimi, "evdev" (evolutionary device) ve "udev" (userspace device) kelimelerinin birleşiminden oluşur.

eudev, Linux çekirdeği tarafından sağlanan "udev" hizmetinin bir alternatifidir. Udev, donanım aygıtlarının dinamik olarak tanınmasını ve yönetilmesini sağlar. Eudev ise, udev'in daha hafif ve basitleştirilmiş bir sürümüdür.

Özetlemek gerekirse, eudev Linux işletim sistemlerinde donanım aygıtlarının tanınması ve yönetimi için kullanılan bir sistemdir. Donanım aygıtlarının otomatik olarak algılanması ve ilgili sürücülerin yüklenmesi gibi işlemleri gerçekleştirir. Bu sayede, kullanıcılar donanım aygıtlarını kolayca kullanabilir ve yönetebilir.

### eudev Derleme

```
#https://www.linuxfromscratch.org/lfs/view/9.1/chapter06/eudev.html
version="3.2.14"
name="eudev"
mkdir -p $HOME/distro
cd $HOME/distro
rm -rf ${name}-${version}
rm -rf build-${name}-${version}
wget https://github.com/eudev-project/eudev/releases/download/v3.2.14/${name}-${version}.tar.gz
tar -xvf ${name}-${version}.tar.gz
mkdir build-${name}-${version}
cd build-${name}-${version}

../${name}-${version}/configure --prefix=/ \
    --bindir=/sbin \
    --sbindir=/sbin \
    --libdir=/lib \
    --disable-manpages \
    --disable-static \
    --disable-selinux \
    --enable-blkid \
    --enable-modules \
    --enable-kmod
make
make install DESTDIR=$HOME/rootfs
```

### busybox Nedir?

Busybox tek bir dosya halinde bulunan birçok araç setine sahip olan bir programdır. Bu araçlar initramfs sisteminde ve sistem genelinde sıkça kullanılır. Busybox aşağıdaki gibi kullanılır. Örneğin, dosya listelemek için ls komutunu kullanmak isterseniz:

```
$ busybox ls
```

Busyboxtaki tüm araçları sisteme sembolik bağ atmak için aşağıdaki gibi bir yol izlenebilir. Bu işlem var olan dosyaları sildiği için tehlikeli olabilir. Sistemin tasarımına uygun olarak yapılmalıdır.

```
$ busybox --install -s /bin # -s parametresi sembolik bağ olarak kurmaya yarar.
```

Busybox **static** olarak derlenmediği sürece bir libc kütüphanesine ihtiyaç duyar. initramfs içerisinde kullanılacaksa içerisine libc dahil edilmelidir. Bir dosyanın static olarak derlenip derlenmediğini öğrenmek için aşağıdaki komut kullanılır.

```
$ ldd /bin/busybox # static derlenmişse hata mesajı verir. Derlenmemişse bağımlılıklarını listeler.
```

Busybox derlemek için öncelikle **make defconfig** kullanılarak veya önceden oluşturduğumuz yapılandırma dosyasını atarak yapılandırma işlemi yapılır. Ardından eğer static derleme yaparsak yapılandırma dosyasına müdahale edilir. Son olarak **make** komutu kullanarak derleme işlemi yapılır.

```
$ make defconfig
$ sed -i "s|.*CONFIG_STATIC_LIBGCC .*|CONFIG_STATIC_LIBGCC=y|" .config
$ sed -i "s|.*CONFIG_STATIC .*|CONFIG_STATIC=y|" .config
$ make
```

Derleme bittiğinde kaynak kodun bulunduğu dizinde busybox dosyamız oluşmuş olur.

Static olarak derlemiş olduğumuz busybox'u kullanarak minimal kök dizin oluşturabiliriz. Burada static yapı kullanılmayacaktır. Sistemdeki /bin/busybox kullanılacaktır. Eğer yoksa busybox sisteme yüklenmelidir.

### e2fsprogs Paketi

e2fsprogs paket sistemde mkfs.ext4, e2fsck, tune2fs vb sistem araçlarının yüklenmesini sağlar. Eğer sistemde bu sistem uygulamaları yoksa bu paketin yüklenmesi veya derlenmesi gerekmektedir.

Eğer /boot bölümünü ayırmayacaksanız grub yüklenirken **unknown filesystem** hatası almanız durumunda aşağıdaki yöntemi kullanabilirsiniz.

```
$ e2fsck -f /dev/sda2  
$ tune2fs -O ^metadata_csum /dev/sda2
```



### Grub Nedir?

Grub (Grand Unified Bootloader), Linux işletim sistemlerinde kullanılan bir önyükleme yükleyicisidir. Bilgisayarınızı başlatırken, işletim sisteminin yüklenmesini sağlar. Grub, bilgisayarınızın BIOS veya UEFI tarafından başlatılmasından sonra devreye girer ve işletim sisteminin yüklenmesi için gerekli olan işlemleri gerçekleştirir.

Grub, önyükleme konfigürasyon dosyası olan grub.cfg veya grub.conf dosyasını kullanır. Bu dosya, hangi işletim sistemlerinin yüklü olduğunu, hangi sürücü ve bölümde olduklarını ve hangi işletim sisteminin önyükleneceğini belirten bilgileri içerir.

Aşağıda, Grub ile ilgili bir örnek konfigürasyon dosyası gösterilmektedir:

```
default=0
timeout=5
menuentry "Linux" {
    set root=(hd0,1)
    linux /vmlinuz root=/dev/sda1
    initrd /initrd.img
}
menuentry "Windows" {
    set root=(hd0,2)
    chainloader +1
}
```

Bu örnekte, Grub, öntanımlı olarak Linux işletim sistemini başlatacaktır. Eğer Windows'u başlatmak isterseniz, Grub menüsünden "Windows" seçeneğini seçebilirsiniz.

### grub Derleme

grub paketini derlemek için aşağıdaki scripti kullanabilirsiniz.

```
version="2.06"
name="grub"

mkdir -p $HOME/distro
cd $HOME/distro
rm -rf ${name}-${version}
rm -rf build-${name}-${version}

wget ftp://ftp.gnu.org/gnu/grub/${name}-${version}.tar.gz
tar -xvf ${name}-${version}.tar.gz
mkdir build-${name}-${version}
cd build-${name}-${version}

../${name}-${version}/configure --prefix= \
    --sysconfdir=/etc \
    --libdir=/lib/ \
    --disable-werror

make
make install DESTDIR=$HOME/rootfs
cd $HOME/rootfs
```

### zlib Nedir?

zlib, sıkıştırma ve açma işlemleri için kullanılan bir kütüphanedir. Linux sistemlerinde sıkıştırma ve açma işlemlerini gerçekleştirmek için sıklıkla kullanılır. zlib, verileri sıkıştırarak daha az yer kaplamasını sağlar ve aynı zamanda sıkıştırılmış verileri orijinal haline geri dönüştürmek için kullanılır.

zlib, genellikle dosya sıkıştırma, ağ iletişimi ve veritabanı yönetimi gibi alanlarda kullanılır. Örneğin, bir dosyayı sıkıştırmak ve daha az depolama alanı kullanmak istediğinizde zlib'i kullanabilirsiniz. Ayrıca, ağ üzerinden veri iletişimi yaparken veri boyutunu azaltmak için de zlib kullanabilirsiniz.

Linux sistemlerinde zlib kütüphanesi genellikle C dilinde kullanılır. Aşağıda basit bir örnek verilmiştir:

```
#include <stdio.h>
#include <zlib.h>

int main() {
    char source[] = "Bu bir örnek metindir.";
    char compressed[1024];
    char decompressed[1024];
    uLong sourceLen = strlen(source);
    uLong compressedLen = sizeof(compressed);
    uLong decompressedLen = sizeof(decompressed);

    compress((Bytef *)compressed, &compressedLen, (const Bytef *)source, sourceLen);
    uncompress((Bytef *)decompressed, &decompressedLen, (const Bytef *)compressed, compressedLen);

    printf("Orijinal Metin: %s\n", source);
    printf("Sıkıştırılmış Metin: %s\n", compressed);
    printf("Açılmış Metin: %s\n", decompressed);

    return 0;
}
```

Bu örnekte, compress fonksiyonu ile source metni sıkıştırılır ve compressed dizisine kaydedilir. Ardından, uncompress fonksiyonu ile compressed dizisi açılır ve decompressed dizisine kaydedilir. Sonuç olarak, orijinal metin, sıkıştırılmış metin ve açılmış metin ekrana yazdırılır.

zlib, Linux sistemlerinde sıkıştırma ve açma işlemleri için güvenilir ve yaygın olarak kullanılan bir kütüphanedir.

### zlib Derleme

```
cd $HOME
wget https://zlib.net/current/zlib.tar.gz

tar -xvf zlib.tar.gz
cd zlib-1.3
#https://www.linuxfromscratch.org/~thomas/multilib/chapter08/zlib.html

./configure --prefix=/

make
make install DESTDIR=$HOME/rootfs
```

## Paket Sistemi Tasarlama

### Paket Sitemi

Dağıtımlarda uygulamalar paketler halinde hazırlanır. Bu paketleri dağıtımda kullanabilmek için temel işlemler şunlardır;

1. Paket Oluşturma
2. Paket Liste İndexi Güncelleme
3. Paket Kurma
4. Paket Kaldırma
5. Paket Yükseltme gibi işlemleri yapan uygulamaların tamamı paket sistemi olarak adlandırılır.

Paket sistemide, uygulama paketi haline getirilip sisteme kurulur. Genelde paket sistemi dağıtımın temel bir parçası olması sebebiyle üzerinde yüklü gelir.

Bazı dağıtımların kullandığı paket sistemeleri şunlardır.

- apt: Debian dağıtımının kullandığı paket sistemi.
- emerge :Gentoo dağıtımının kullandığı paket sistemi.
- ymp : Turkman Linux dağıtımının kullandığı paket sistemi.

### bps Paket Sistemi

Bu dokümanda hazırlanan dağıtımın paket sistemi için ise bps(basit/basic/base paket sistemi) olarak ifade edeceğimiz paket sistemi adını kullandık. Bps paket sistemindeki beş temel işlemin nasıl yapılacağı ayrı başlıklar altında anlatılacaktır. Paket sistemi delemeli bir dil yerine bash script ile yapılacaktır. Bu dokümanı takip eden orta seviye bilgiye sahip olan linux kullanıcısı yapılan işlemleri anlayacaktır.

### Paket Oluşturma

bps paket sisteminin temel parçalarından en önemlisi paket oluşturma uygulamasıdır. Dokümanda temel paketlerin nasıl derlendiği **Temel Paketler** başlığı altında anlatılmıştı. Bir paket üzerinden(readline) örneklendirerek paketimizi oluşturacak scriptimizi yazalım.

Dokümanda readline paketi nasıl derleneceği aşağıdaki script ile ifade edilmişti.

```
# kaynak kod indirme ve derleme için hazırlama
version="8.1"
name="readline"
mkdir -p $HOME/distro
cd $HOME/distro
rm -rf ${name}-${version}
rm -rf build-${name}-${version}
wget https://ftp.gnu.org/pub/gnu/readline/${name}-${version}.tar.gz
tar -xvf ${name}-${version}.tar.gz
mkdir build-${name}-${version}
cd build-${name}-${version}
../${name}-${version}/configure --prefix=/ --enable-shared --enable-multibyte

# derleme
make

# derlenen paketin yüklenmesi ve ayarlamaların yapılması
make install DESTDIR=$HOME/rootfs
```

Bu script readline kodunu internetten indirip derliyor ve kurulumu yapıyor. Aslında bu scriptle **paketleme**, **paket kurma** işlemini bir arada yapıyor. Bu işlem mantıklı gibi olsada paket sayısı arttıkça ve rutin yapılan işlemleri tekrar tekrar yapmak gibi işlem fazlalığına sebep olmaktadır.

Bu sebeplerden dolayı **readline** paketleme scriptini yeniden düzenleyelim. Yeni düzenlenen halini **bpspaketle** ve **bpsbuild** adlı script dosyaları olarak düzenleyeceğiz. Genel yapısı aşağıdaki gibi olacaktır.

#### **bpsbuild** Dosyası

```
setup() {}
build() {}
package() {}
```

#### **bpspaketle** Dosyası

```
#genel değişkenler tanımlanır
initsetup() {}

#bpsbuild dosya fonksiyonları birleştiriliyor
source bpsbuild # bu komutla setup build package fonksiyonları bpsbuild doyasından alınıp birleştiriliyor

packageindex() {}
packagecompress() {}
```

## Paket Sistemi Tasarlama

Aslında yukarıdaki **bpspaketle** ve **bpsbuild** adlı script dosyaları tek bir script dosyası olarak **bpspaketle** dosyası. İki dosyayı birleştiren **source bpsbuild** komutudur. **bpspaketle** dosyası aşağıdaki gibi düşünebiliriz.

```
#genel değişkenler tanımlanır
initsetup() {}

setup() {} #bpsbuild dosyasından gelen fonksiyon
build() {} #bpsbuild dosyasından gelen fonksiyon
package() {} #bpsbuild dosyasından gelen fonksiyon

packageindex() {}
packagecompress() {}
```

Bu şekilde ayrılmasının temel sebebi **bpspaketle** scriptinde hep aynı işlemler yapılırken **bpsbuild** scriptindekiler her pakete göre değişmektedir. Böylece paket yapmak için ilgili pakete özel **bpsbuild** dosyası düzenlememiz yeterli olacaktır. **bpspaketle** dosyamızda **bpsbuild** scriptini kendisiyle birleştirip paketleme yapacaktır.

### **bpsbuild** Dosyamızın Son Hali

```
#!/usr/bin/env bash
version="8.1"
name="readline"
depends="glibc"
description="readline kütüphanesi"
source="https://ftp.gnu.org/pub/gnu/readline/${name}-${version}.tar.gz"
groups="sys.apps"
setup()
{
    ../${name}-${version}/configure --prefix=/ --enable-shared --enable-multibyte
}
build()
{
    make
}
package()
{
    make install DESTDIR=$DESTDIR
}
```

### **bpspaketle** Dosyamızın Son Hali

```
#!/usr/bin/env bash
set -e
paket=$1
dizin=$(pwd)
if [ ! -d "${paket}" ]; then echo "Bir paket değil!"; exit; fi
if [ ! -f "${paket}/bpsbuild" ]; then echo "Paket dosyası bulunamadı!"; exit; fi
echo "Paket : $paket"
source ${paket}/bpsbuild
DESTDIR=/tmp/bps/build/rootfs-${name}-${version}
SOURCEDIR=/tmp/bps/build/${name}-${version}
BUILDDIR=/tmp/bps/build/build-${name}-${version}

# paketin indirilmesi ve /tmp/bps/build konumunda derlenmesi için gerekli izinler hazırlanır.
initsetup()
{
    mkdir -p /tmp/bps
    mkdir -p /tmp/bps/build
    cd /tmp/bps/build
    rm -rf ./.*
    rm -rf build-${name}-${version}*
    rm -rf ${name}-${version}*
    rm -rf rootfs-${name}-${version}*

    if [ -n "${source}" ]
    then
        wget ${source}
        downloadfile=$(ls|head -1)
        filetype=$(file -b --extension $downloadfile|cut -d'/' -f1)
        echo "*****dosya sıkıştırma türü*****:${filetype}"
        if [ ${filetype} == "bz2" ]; then tar -xvf ${downloadfile}; fi
        if [ ${filetype} == "tar" ]; then tar -xvf ${downloadfile}; fi
        if [ ${filetype} == "xz" ]; then tar -xvf ${downloadfile}; fi
        if [ "${filetype}" == "gz" ]; then echo "*****dosya gz ile sıkıştırılmış***"; tar -xvf ${downloadfile}; fi
        if [ "${filetype}" == "???" ]; then echo "*****dosya zip ile sıkıştırılmış***"; unzip ${downloadfile}; fi
        #*****
        director=$(find ./.* -maxdepth 0 -type d)
        if [ "${director}" != "./${name}-${version}" ]; then mv $director ${name}-${version}; fi
    fi
    mkdir -p build-${name}-${version}
    mkdir -p rootfs-${name}-${version}
    cp ${dizin}/${paket}/bpsbuild /tmp/bps/build
    cd build-${name}-${version}
}

#paketlenecek dosyaların listesini tutan file.index dosyası oluşturulur
packageindex()
{
    rm -rf file.index
    cd /tmp/bps/build/rootfs-${name}-${version}
    find . -type f | while IFS= read file_name; do if [ -f ${file_name} ]; then echo ${file_name:1}>>../file.index; fi done
    find . -type l | while IFS= read file_name; do if [ -L ${file_name} ]; then echo ${file_name:1}>>../file.index; fi done
}

# paket dosyası oluşturulur;
# kurulacak data rootfs.tar.xz, file.index ve bpsbuild dosyaları tek bir dosya olarak tar.gz dosyası olarak hazırlanıyor.
# tar.gz dosyası olarak hazırlanan dosya bps ismiyle değiştirilip paketimiz hazırlanır.

packagecompress()
{
    cd /tmp/bps/build/rootfs-${name}-${version}
    tar -cf ../rootfs.tar ./.*
    cd /tmp/bps/build/
    xz -9 rootfs.tar
    tar -cvzf paket-${name}-${version}.tar.gz rootfs.tar.xz file.index bpsbuild
    cp paket-${name}-${version}.tar.gz ${dizin}/${paket}/${name}-${version}.bps
}

# fonksiyonlar aşağıdaki sırayla çalışacaktır.
echo "***** initsetup *****"; initsetup #bu dosya içindeki fonksiyon
echo "***** setup *****"; setup #bpsbuild dosyasından gelen fonksiyon
echo "***** build *****"; build #bpsbuild dosyasından gelen fonksiyon
echo "***** package *****"; package #bpsbuild dosyasından gelen fonksiyon
echo "***** packageindex *****"; packageindex #bu dosya içindeki fonksiyon
echo "***** packagecompress *****"; packagecompress #bu dosya içindeki fonksiyon
```

Burada **readline** paketini örnek olarak **bpspaketle** dosyasının ve **bpsbuild** dosyasının nasıl hazırlandığı anlatıldı. Diğer paketler için sadece hazırlanacak pakete uygun şekilde **bpsbuild** dosyası hazırlayacağız. **bpspaketle** dosyamızda değişiklik yapmayacağız. Artık **bpspaketle** dosyası paketimizi oluşturan script **bpsbuild** ise hazırlanacak paketin bilgilerini bulunduran script dosyasıdır.

## Paket Sistemi Tasarlama

### Paket Yapma

Bu bilgilere göre readline paketi nasıl oluşturulur onu görelim. Paketlerimizi oluşturacağımız bir dizin oluşturarak aşağıdaki işlemleri yapalım. Burada yine **readline** paketi anlatılacaktır.

```
mkdir readline
cd readline
#readline için hazırlanan bpsbuild dosyası bu konuma oluşturulur ve içeriği readline için oluşturduğumuz bpsbuild içeriği olarak ayarlanır.
cd ..
./bpspaketle readline # bpspaketle dosyamızın bu konumda olduğu varsayılmıştır ve parametre olarak readline dizini verilmiştir.
```

Komut çalışınca readline/readline-8.1.bps dosyası oluşacaktır. Artık sisteme kurulum için ikili dosya, kütüphaneleri ve dizinleri barındıran paketimiz oluşturuldu. Bu paketi sistemimize nasıl kurarız? konusu **Paket Kurma** başlığı altında anlatılacaktır.

### Paket Liste İndexi Güncelleme

Dağıtımlarda uygulamalar paketler halinde hazırlanır. Bu paketleri isimleri, versiyonları ve bağımlılık gibi temel bilgileri barındıran liste halinde tutan bir dosya oluşturulur. Bu dosyaya **index.lst** isim verebiliriz. Bu dokümanda bu listeyi tutan **index.lst** dosyası kullanılmıştır. Paket sisteminde güncelleme aslında **index.lst** dosyanın en güncellen halinin sisteme yüklenmesi olayıdır.

bps paketleme sisteminde **bpsupdate** scripti hazırlanmıştır. Bu script **index.lst** dosyasının paketlerimizin en güncel halini sistemimize yükleyecektir. Bu dağıtımda paketlerimizi github.com üzerinde oluşturulan bir repostory üzerinden çekilmektedir. Paket listemiz ise yapılan her yeni paketi yükleme sırasında güncellenmektedir.

Paket güncelleme için iki script kullanmaktayız. Bunlar;

#### **index.lst** Dosyasını Oluşturma

```
#index alma scripti
#!/bin/sh
set -ex
>index.lst
find ./ * -type f -name *bps |
    while IFS= read file_name; do
        tar -xf ${file_name} bpsbuild
        version=$(cat bpsbuild|grep version=)
        name=$(cat bpsbuild|grep name=)
        depends=$(cat bpsbuild|grep depends=)
        echo "$name:$version:$depends">>index.lst
    done
rm -rf bpsbuild
mkdir /output -p
cp -rf index.lst /output
```

Bu script bps paket dosyalarımızın olduğu dizinde tüm paketleri açarak içerisinde **bpsbuild** dosyalarını çıkartarak paketle ilgili bilgileri alıp **index.lst** dosyası oluşturmaktadır. İstersek paketler local ortamdada index oluşturabiliriz. Bu dokümanda github üzerinde oluşturacak şekilde anlatılmıştır. Paket indeksi oluşturan **index.lst** dosyası aşağıdaki gibi olacaktır. Listede name, version ve depends(bağımlı olduğu paketler) bilgileri bulunmaktadır. Bilgilerin arasında : karakteri kullanılmıştır.

```
name="glibc":version="2.38":depends=""
name="gmp":version="6.3.0":depends="glibc,readline,ncurses"
name="grub":version="2.06":depends="glibc,readline,ncurses"
name="kmod":version="31":depends="glibc,zlib"
```

#### **index.lst** Dosyasını Güncelleme

##### **bpsupdate** dosya içeriği

```
#!/bin/sh
./indirgentoo /tmp/index.lst https://bayramkarahan.github.io/distro-binary-package/index.lst
```

**index.lst** dosyamızı github üzerinden indiren scriptimiz tek bir satırdan oluşmaktadır. **indirgentoo** dosyamız gento üzerinde curl kütüphanesi kullanan bir c kodundan oluşmaktadır.



## Paket Sistemi Tasarlama

### İndirme Uygulaması

İndirme dosyamız **indirgentoo** dur. Adının böyle olmasının sebebi bağımlılık sorunlarını en aza indirmek için static olarak **gentoo** ortamında derlendiği için böyle adlandırıldı. **indirgentoo** kodları aşağıdadır görülmektedir.

```
#include <stdio.h>
#include <curl/curl.h>
struct FtpFile { const char *filename; FILE *stream;};
static size_t my_fwrite(void *buffer, size_t size, size_t nmemb, void *stream){
    struct FtpFile *out = (struct FtpFile *)stream;
    if(!out->stream) {
        out->stream = fopen(out->filename, "wb");
        if(!out->stream) return -1; /* failure, cannot open file to write */
    }
    return fwrite(buffer, size, nmemb, out->stream);
}
int main(int argc, char **argv) {
    const char *outname; argv++; outname = *argv;
    const char *fileaddress; argv++; fileaddress=*argv;
    printf("ad1:%s", outname);
    printf("adres:%s", fileaddress);
    CURL *curl;
    CURLcode res;
    struct FtpFile ftpfile = {
        outname, /* name to store the file as if successful */
        NULL
    };

    curl_global_init(CURL_GLOBAL_DEFAULT);
    curl = curl_easy_init();
    if(curl) {
        curl_easy_setopt(curl, CURLOPT_URL, fileaddress);
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, my_fwrite); /* Define our callback to get called when there's data to be written */
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &ftpfile); /* Set a pointer to our struct to pass to the callback */
        curl_easy_setopt(curl, CURLOPT_USE_SSL, CURLUSESSL_ALL); /* We activate SSL and we require it for both control and data */
        curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L);
        curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L); /* Switch on full protocol/debug output */
        res = curl_easy_perform(curl);
        curl_easy_cleanup(curl); /* always cleanup */
        if(CURLE_OK != res) { /* we failed */ fprintf(stderr, "curl told us %d\n", res); }
    }

    if(ftpfile.stream)
        fclose(ftpfile.stream); /* close the local file */
    curl_global_cleanup();
    return 0;
}
```

**indirgentoo** çalışabilir dosyamız iki parametre almaktadır. ilk parametre indirilecek dosyanın nereye ve hangi adla kaydedileceği belirtiliyor. İkinci parametre ise hangi adresten incekse ilgili adres bilgisidir.

```
./indirgentoo /tmp/index.lst https://bayramkaran.github.io/distro-binary-package/index.lst
```

Bu komut <https://bayramkaran.github.io/distro-binary-package/index.lst> adresindeki dosyayı index.lst dosyasını //tmp/index.lst konumuna indirecektir.

### Paket Kurma

Hazırlanan dağıtımda paketlerin kurulması için sırasıyla aşağıdaki işlem adımları yapılmalıdır.

1. Paketin indirilmesi
2. İndirilen paketin /tmp/bps/kur/ konumunda açılması
3. Açılan paket dosyalarının / konumuna yüklenmesi(kopyalanması)
  - Paketin bağımlı olduğu paketler varmı kontrol edilir
  - Yüklü olmayan bağımlılıklar yüklenir
4. Yüklenen paket bilgileri(name, version ve bağımlılık) yüklü paketlerin index bilgilerini tutan paket sistemi dizininindeki index dosyasına eklenir.
5. Açılan paket içindeki yüklenen dosyaların nereye yüklendiğini tutan file.index dosyası paket sistemi dizinine yüklenir

Bu işlemler daha detaylandırılabilir. Bu işlemlerin detaylı olması paket sisteminin kullanılabilirliğini ve yetenekleri olarak ifade edebiliriz. İşlem adımlarını kolaylıkla sıralarken bunları yapacak script yazmak ciddi planlamalar yapılarak tasarlanması gerekmektedir.

Burada basit seviyede kurulum yapan script kullanılmıştır. Detaylandırıldıkça doküman güncellenecektir. Kurulum scripti aşağıda görülmektedir.

#### bpskur Scripti

```
#!/bin/sh
#set -e
paket=$1
paketname="name=\"${paket}\""
ROOTFS=$2
#echo "$paket"
indexpaket=$(cat /tmp/index.lst|grep $paketname)
name=""
version=""
depends=""
if [ -n "$indexpaket" ]
then
    namex=$(echo $indexpaket|cut -d":" -f1)
    versionx=$(echo $indexpaket|cut -d":" -f2)
    dependx=$(echo $indexpaket|cut -d":" -f3)
    name=${namex:6:-1}
    version=${versionx:9:-1}
    depends=${dependx:9:-1}
else
    echo "*****Paket Bulunamadı*****"; exit
fi

# paketi indirme
mkdir -p /tmp/bps
mkdir -p /tmp/bps/kur
rm -rf /tmp/bps/kur/*
./indirgentoo /tmp/bps/kur/${name}-${version}.tar.gz https://github.com/bayramkarahan/distro-binary-package/raw/master/${name}/${name}-${version}.bps
mkdir -p /var/lib/bps
cd /tmp/bps/kur/

# paketi açma
tar -xf ${name}-${version}.tar.gz
mkdir -p rootfs
tar -xf rootfs.tar.xz -C rootfs

# paketi kurma
cp -prfv rootfs/* $ROOTFS/

#name version depends /var/bps/index.lst eklenmesi
echo "name=\"${name}\" : \"version=\"${version}\" : \"depends=\"${depends}\"" >> var/bps/index.lst
#paket içinde gelen paket dosyalarının dosya ve izin yapısını tutan file index dosyanının /var/bps/ konumuna kopyalanması
cp file.lst /var/bps/${name}-${version}.lst
```

#### bpskur Scriptini Kullanma

Script iki parametre almaktadır. İlk parametre paket adı. İkinci parametremiz ise nereye kuracağını belirten hedef olmalıdır. Bu scripti kullanarak readline paketi aşağıdaki gibi kurulabilir.

```
./bpskur readline /
```

### Paket Kaldırma

Sistemde kurulu paketleri kaldırmak için işlem adımları şunlardır.

1. Paketin kullandığı bağımlılıkları başka paketler kullanıyor mu kontrol edilir. Eğer kullanılmıyorsa kaldırılır.
2. Paketin file.lst dosyası içerisindeki dosyalar, dizinler kaldırılır.
3. Kaldırılan dosyalardan sonra /var/bps/paket-version.lst dosyasından paket bilgisi kaldırılır.
4. /var/bps/index.lst dosyasından ilgili paket satırı kaldırılmalıdır.

Paketi kaldırmak için ise aşağıdaki örnek kullanılabilir.

bpskaldir scripti

```
#!/bin/sh
#set -e
paket=$1
paketname="name=\"${paket}\""
indexpaket=$(cat /var/bps/index.lst|grep $paketname)
name=""
version=""
depends=""
if [ -n "${indexpaket}" ]; then
    namex=$(echo $indexpaket|cut -d":" -f1)
    versionx=$(echo $indexpaket|cut -d":" -f2)
    dependsx=$(echo $indexpaket|cut -d":" -f3)
    name=${namex:6:-1}
    version=${versionx:9:-1}
    depends=${dependsx:9:-1}
else
    echo "*****Paket Bulunamadı*****"; exit
fi
# Bağımlılıkları başka paketler kullanıyor mu kontrol edilir
echo "bağımlılık kontrolü yapılacak"

# Paketin file.lst dosyası içerisindeki dosyalar, dizinler kaldırılır.
cat /var/bps/${paket}-${version}.lst | while read dosya ; do if [[ -f "$dosya" ]]; then rm -f "$dosya"; fi done
cat /var/bps/${paket}-${version}.lst | while read izin ; do if [[ -d "$dizin" ]]; then rmdir "$dizin" || true; fi done

# /var/bps/paket-version.lst dosyasından paket bilgisi kaldırılır.
rm -f /var/bps/${paket}-${version}.lst
# /var/bps/index.lst dosyasından ilgili paket satırı kaldırılır.
sed '/^name=\"${paket}\"/d' /var/bps/index.lst
```

Bağımlılıkları başka paketler kullanıyor mu kontrol edilir. Script içinde bu işlem yapılmamıştır. Daha sonra güncellenecektir. Bu örnekte paket listesini satır satır okuduk. Önce dosya olanları sildik. Daha sonra tekrar okuyup boş kalan dizinleri sildik. Son olarak paket listesi dosyamızı sildik. Bu işlem sonunda paket silinmiş oldu.

bpskaldir Kullanma

**bpskaldir** scripti aşağıdaki gibi kullanılır.

```
./bpskaldir readline
```

## initrd Hazırlama

### initrd

initrd (initial RAM disk), Linux işletim sistemlerinde kullanılan bir geçici dosya sistemidir. Bu dosya sistemi, işletim sistemi açılırken kullanılan bir köprü görevi görür ve gerçek kök dosya sistemine geçiş yapmadan önce gerekli olan modülleri ve dosyaları içerir. Ayrıca, sistem başlatıldığında kök dosya sistemine erişim sağlamadan önce gerekli olan dosyaları yüklemek için de kullanılabilir.

### Temel Dosyalar Ve Açılış Süreci

Linux sisteminin açılabilmesi için aşağıdaki 3 dosya yeterlidir.

```
distro/iso/boot/initrd.img
distro/iso/boot/vmlinuz
distro/iso/boot/grub/grub.cfg
```

Bu dosyaları yukarıdaki gibi dizin konumlarına koyduktan sonra;

```
grub-mkrescue iso/ -o distro.iso #iso dosyamız oluşturulur.
```

Bu komut çalışınca **distro.iso** dosyası elde ederiz. Artık iso dosyamız boot edebilen hazırlanmış bir dosyadır. Burada bazı sorulara cevap vermemiz gerekmektedir.

**distro/iso/boot/initrd.img** dosyası sistemin açılış sürecinde ön işlemleri yaparak gerçek sisteme geçiş sürecini yöneten bir dosyadır. Yazın devamında nasıl hazırlanacağı anlatılacaktır.

**distro/iso/boot/vmlinuz** dosyamız kernelimiz oluyor. Ben kullandığım debian sisteminin mevcut kernelini kullandım. İstenirse kernel derlenebilir.

**distro/iso/boot/grub/grub.cfg** dosyamız ise initrd.img ve vmlinuz dosyalarının grub yazılımını nerede bulacağını gösteren yapılandırma dosyasıdır.

### Bir linux sisteminin açılış süreci şu şekilde olmaktadır.

1. Bilgisayara Güç Verilmesi
2. Bios İşlemleri Yapılıyor(POST)
3. LILO/GRUB Yazılımı Yükleniyor(grub.cfg dosyası okunuyor ve vmlinuz ve initrd.img devreye giriyor)
4. vmlinuz initrd.img sistemini belleğe yüklüyor
5. initrd.img içindeki init dosyasındaki işlem sürecine göre sistem işlemlere devam ediyor\*\*
6. initrd.img içindeki init dosyası temel işlemleri ve modülleri yükledikten sonra disk üzerindeki sisteme(/sbin/init) exec switch\_root komutuyla süreci devrederek görevini tamamlamış olur

Yazının devamında sistem için gerekli olan 3 temel dosyanın(initrd.img, vmlinuz, grub.cfg) hazırlanması ve iso yapıma süreci anlatılacaktır.

### initrd Dosya İçeriği

**initrd.img** dosyasını hazırlarken gerekli olacak dosyalarımızın dizin yapısı ve konumu aşağıdaki gibi olmalıdır. Anlatım buna göre yapılacaktır. Örneğin S1 ifadesi satır 1 anlamında anlatımı kolaylaştırmak için yazılmıştır. Aşağıdaki yapıyı oluşturmak için yapılması gerekenleri adım adım anlatılacaktır.

```
S1- $boot/bin/busybox                #dosya
S2- $boot/sbin/kmod                  #dosya
S3- $boot/sbin/debmod                #dosya
S4- $boot/sbin/insmod                 #dosya
S5- $boot/bin/lsmmod                 #dosya
S6- $boot/sbin/modprobe               #dosya
S7- $boot/sbin/rmmod                  #dosya
S8- $boot/sbin/modinfo                #dosya
S9- $boot/lib/modules/${uname -r}/modu #dizin
S10- $boot/bin/udevadm                 #dosya
S11- $boot/bin/udev                   #dosya
S12- $boot/etc/udev/rules.d            #dizin
S13- $boot/lib/udev/rules.d            #dizin
S14- $boot/initrd/bin/init             #dosya
S15- distro/iso/initrd.img             #dosya
S16- distro/iso/vmlinuz                #dosya
S17- distro/iso/grub/grub.cfg          #dosya
```

S1-S17 arasındaki dosya ve dizin yapısını hazırladığımız **initrd** adındaki script hazırlayacak ve iso haline getirecektir.

## initrd Hazırlama

S1-S17 arasındaki adımları yapacak **initrd** scripti aşağıdaki gibi hazırlandı.

### initrd Scripti

```
#!/bin/bash
boot=$HOME/distro/initrd
rm -rf $boot

mkdir -p $HOME/distro
mkdir -p $boot
mkdir -p $boot/bin
#*****hazırlanmış olan bps paketlerimiz yükleniyor*****
./bpsupdate
./bpskur glibc $boot/          # Dağıtımımızın temel kütüphanesini oluşturan paket yükleniyor
./bpskur busybox $boot/        # S1- distro/initrd/bin/busybox paketi yükleniyor
./bpskur kmod $boot/           # S2-S8 distro/initrd/bin/kmod aşamalarını kmod paketi yüklenince oluşur

#*****modül yükleme*****S9- distro/initrd/lib/modules/$(uname -r)/moduller hazırlanıyor
mkdir -p $boot/lib/modules/
mkdir -p $boot/lib/modules/$(uname -r)
mkdir -p $boot/lib/modules/$(uname -r)/moduller
cp /lib/modules/$(uname -r)/kernel/* -prvf $boot/lib/modules/$(uname -r)/moduller/ #sistemden kopyalandı..
/sbin/depmod --all --basedir=$boot #modül indeksi oluşturluyor

./bpskur eudev $boot/          # S10-S13 eudev paketi yüklenerek oluşturur
./bpskur base-file $boot/      # S14- $boot/initrd/bin/init oluşturma
./bpskur util-linux $boot/
./bpskur grub $boot/
./bpskur e2fsprogs $boot/

#*****initrd.img oluşturuluyor*****# S15- distro/iso/initrd.img
cd $boot
find | cpio -H newc -o >../initrd.img
#*****iso *****
mkdir -p $HOME/distro/iso
mkdir -p $HOME/distro/iso/boot
mkdir -p $HOME/distro/iso/boot/grub
mkdir -p $HOME/distro/iso/live || true

#iso dizinine vmlinuz ve initrd.img dosyamız kopyalanıyor
cp /boot/vmlinuz-$(uname -r) $HOME/distro/iso/boot/vmlinuz #sistemde kullandığım kerneli kopyladım istenirde kernel derlenebilir.
mv $HOME/distro/initrd.img $HOME/distro/iso/boot/initrd.img #oluşturduğumuz **initrd.img** dosyamızı taşıyoruz.

#grub menüsü oluşturuluyor..
cat > $HOME/distro/iso/boot/grub/grub.cfg << EOF
linux /boot/vmlinuz net.ifnames=0 biosdevname=0
initrd /boot/initrd.img
boot boot=live
EOF
```

### S1- \$boot/bin/busybox

busybox küçük boyutlu dağıtım ve initrd hazırlamada kullanılan, birçok uygulamayı içinde barındıran dosyamızdır. **Temel Paketler** başlığı altında nasıl derleneceği anlatıldı. Derleme ve paket oluşturma aşamalarında **busybox** paketinizi oluşturduğunuzu varsayıyoruz. Burada sisteme nasıl ekleneceği anlatılacaktır.

```
./bpskur busybox $boot/
```

## S2-S8 \$boot/bin/kmod

kmod yazısında kmod anlatılmıştır. Burada sisteme nasıl ekleneceği anlatılacaktır. kmod paketi aşağıdaki komut satırıyla kurulmaktadır.

```
./bpskur kmod $boot/
```

Kurulum tamamlandığında paket içerisindeki dosya ve sembolik link dosyaları aşağıdaki gibi **\$boot** konumuna yüklenecektir.

```
$boot/sbin/kmod
ln -s $boot/sbin/kmod $boot/sbin/depmod      #kmod sembolik link yapılarak depmod hazırlandı.
ln -s $boot/sbin/kmod $boot/sbin/insmod      #kmod sembolik link yapılarak insmod hazırlandı.
ln -s $boot/sbin/kmod $boot/bin/lsmmod       #kmod sembolik link yapılarak lsmod hazırlandı.
ln -s $boot/sbin/kmod $boot/sbin/modinfo     #kmod sembolik link yapılarak modinfo hazırlandı.
ln -s $boot/sbin/kmod $boot/sbin/modprobe    #kmod sembolik link yapılarak modprobe hazırlandı.
ln -s $boot/sbin/kmod $boot/sbin/rmmod       #kmod sembolik link yapılarak rmmod hazırlandı.
```

## S9- \$boot/lib/modules/\$(uname -r)/moduller

Bu bölümde modüller hazırlanacak. Burada dikkat etmemiz gereken önemli bir nokta kullandığımız kernel versiyonu neyse **\$boot/lib/modules/modules** altında oluşacak dizinimiz aynı olmalıdır. Bundan dolayı **\$boot/lib/modules/\$(uname -r)** şeklinde dizin oluşturulmuştur. Aşağıda kullandığımız son satırdaki **/sbin/depmod --all --basedir=initrd, \$boot/lib/modules/\$(uname -r)/moduller** altındaki modüllerimizin indeksini oluşturuyor.

```
mkdir -p $boot/lib/modules/
mkdir -p $boot/lib/modules/$(uname -r)
mkdir -p $boot/lib/modules/$(uname -r)/moduller

cp /lib/modules/$(uname -r)/kernel/* -prvf $boot/lib/modules/$(uname -r)/moduller/ #modüller sistemden kopyalandı..
/sbin/depmod --all --basedir=$boot #modüllerin indeks dosyası oluşturuluyor
```

## S10-S13- \$boot/bin/udevadm

**udevadm**, Linux işletim sistemlerinde kullanılan bir araçtır. Bu araç, udev (Linux çekirdeği tarafından sağlanan bir hizmet) ile etkileşim kurmamızı sağlar. **udevadm** sistemdeki aygıtların yönetimini kolaylaştırmak için kullanılır. **udev** ise udevadm'in bir bileşenidir ve donanım olaylarını işlemek için kullanılır.

```
./bpskur eudev $boot/ # paket kuruluyor
```

Paket kurulunca aşağıdaki gibi bir dizin yapısı ve dosyalar dağıtım dizinimize(\$boot) yüklenecektir.



**S14- distro/initrd/bin/init**

kernel ilk olarak initrd.img dosyasını ram'e yükleyecek ve ardından **init** dosyasının arayacaktır. Bu dosya bir script dosyası veya binary bir dosya olabilir. **init** ve sistem için gereken temel dosyaları **base-file** paketi olarak hazırladık. **base-file** paketi aşağıdaki komutla kurulur.

```
./bpskur base-files $boot/          # paket kuruluyor
```

*base-file\** paketi içindeki **init** script dosyası aşağıdaki gibi hazırlandı.

**init Dosyası**

```
#!/bin/busybox ash
/bin/busybox mkdir -p /bin
/bin/busybox --install -s /bin
#*****
export PATH=/sbin:/bin:/usr/bin:/usr/sbin:

[ -d /dev ] || mkdir -m 0755 /dev
[ -d /root ] || mkdir -m 0700 /root
[ -d /sys ] || mkdir /sys
[ -d /proc ] || mkdir /proc
mkdir -p /tmp /run
touch /dev/null

# devtmpfs does not get automounted for initramfs
mount -t devtmpfs devtmpfs /dev
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mount -t tmpfs tmpfs /tmp
#*****init üzerinden dosya script çalıştırmak için****
for x in $(cat /proc/cmdline); do
    case $x in
        init=*)
            init=${x#init=}
            echo " bu bir test :${x#init=}"
            ${x#init=}
            ;;
    esac
done

echo "initrd başlatıldı"
/bin/busybox ash
```

Oluşturulan **initrd.img** dosyası çalışacak tty açacak(konsol elde etmiş olacağız. Aslında bu işlemi yapan şey busybox ikili dosyası.



### S15- distro/iso/initrd.img

initrd.img dosyası kernel(vmlinuz) ile birlikte kullanılan belleğe ilk yüklenen dosyadır. Bu dosyanın görevi sistemin kurulu olduğu diski tanımak için gereken modülleri yüklemek ve sistemi başlatmaktır.

Bu dosya /boot/initrd.img-xxx konumunda yer alır. **\$HOME/distro/initrd.img** konumuna dosyamızı aşağıdaki gibi oluşturulur.

```
cd $boot
find | cpio -H newc -o >../initrd.img
```

**initrd.img** iso dosyası hazırlamak için **\$HOME/distro/iso/boot/initrd.img** konumuna taşındı.

```
mv $HOME/distro/initrd.img iso/boot/initrd.img # Oluşturulan **initrd.img** dosyası taşınır.
```

### S16- distro/iso/vmlinuz

vmlinuz linuxta **kernel** diye ifade edilen dosyadır. Burada kernel derlemek yerine debianda çalışan kernel dosyamı kullandım. Kernel derlediğinizde **vmlinuz** dosyası elde edeceksiniz. Kernel derleme ayrı başlık altında anlatılmaktadır.

```
cp /boot/vmlinuz-$(uname -r) iso/boot/vmlinuz #sistemde kullandığım kerneli kopyaladım istenirde kernel derlenebilir.
```

### S17- distro/iso/grub/grub.cfg

grub menu dosyası oluşturuluyor.

```
cat > iso/boot/grub/grub.cfg << EOF
linux /boot/vmlinuz
initrd /boot/initrd.img
boot
EOF
```

Yukarıdaki script **iso/boot/grub/grub.cfg** dosyasının içeriği olacak şekilde ayarlanır.

## Sistem Hazırlama

### Dağıtım Hazırlama

**initrd** hazırlama aşamaları **initrd** konu başlığında detaylıca anlatıldı. Sistem hazırlanırken küçük farklılıklar olsada **initrd** hazırlamaya benzer aşamalar yapılacaktır. Sistemimin yani oluşacak **iso** dosyasının yapısı aşağıdaki gibi olacaktır. Aşağıda sadece **filesystem.squashfs** dosyasının hazırlanması kaldı.

```
$HOME/distro/iso/boot/grub/grub.cfg  
$HOME/distro/iso/boot/initrd.img  
$HOME/distro/iso/boot/vmlinuz  
$HOME/distro/live/filesystem.squashfs
```

### filesystem.squashfs Hazırlama

**filesystem.squashfs** dosyası **/initrd.img** dosyasına benzer yapıda hazırlanacak. En büyük farklılık **init** çalışabilir dosya içeriğinde yapılmalı. Yapı **/initrd.img** dizin yapısı gibi hazırlandıktan sonra **filesystem.squashfs** oluşturulmalı ve **\$HOME/distro/live/filesystem.squashfs** konuma kopyalanmalıdır. Aşağıdaki komutlarla **filesystem.squashfs** hazırlanıyor ve **\$HOME/distro/live/** konumuna taşınıyor.

```
cd $HOME/distro/  
mksquashfs $rootfs $HOME/distro/filesystem.squashfs -comp xz -wildcards  
mv $HOME/distro/filesystem.squashfs $HOME/distro/iso/live/filesystem.squashfs
```

### İso Dosyasının Oluşturulması

```
grub-mkrescue iso/ -o distro.iso #iso doyamız oluşturulur.
```

Artık sistemi açabilen ve tty açıp bize sunan bir yapı oluşturduk. Çalıştırmak için qemu kullanılabilir.

**qemu-system-x86\_64 -cdrom distro.iso -m 1G** komutuyla çalıştırıp test edebiliriz.

## Sistem Kurulumu

### İki Bölüm Kurulum

Bu bölümde **Ext4** dosya sistemine grub kullanarak kurulum anlatılacaktır. Anlatım boyunca **/dev/sda** diski üzerinden örneklemeye yapılmıştır. Siz kendi diskinize göre düzenleyebilirsiniz. Diskler üzerinde işlem yapabilmek için evdev veya udevd servisi çalışıyor olmalı. Disk ve isoya erişim için aşağıdaki modüllerin yüklü olduğundan emin olun.

- loop
- squashfs
- ext4 modülleri **modprobe** komutuyla yüklenmeli.

#### Disk Hazırlanmalı

Öncelikle **cfdisk** veya **fdisk** komutları ile diski bölümlendirelim. Ben bu anlatımda **cfdisk** kullanacağım.

0. cfdisk komutuyla disk bölümlendirilmeli. .. code-block:: shell

```
$ cfdisk /dev/sda
```

1. gpt seçilmeli
2. 512 MB type vfat alan(sda1)
3. geri kalanı type linux system(sda2)
4. write
5. quit
6. Bu işlem sonucunda sadece sda1 sda2 olur
7. mkfs.vfat ve mkfs.ext4 ile diskler biçimlendirilir.

```
$ mkfs.vfat /dev/sda1  
$ mkfs.ext4 /dev/sda2
```

#### e2fsprogs Paketi

e2fsprogs paket sistemde mkfs.ext4, e2fsck, tune2fs vb sistem araçlarının yüklenmesini sağlar. Eğer sistemde bu sistem uygulamaları yoksa bu paketin yüklenmesi veya derlenmesi gerekmektedir.

Eğer /boot bölümünü ayırmayacaksanız grub yüklenirken **unknown filesystem** hatası almanız durumunda aşağıdaki yöntemi kullanabilirsiniz.

```
$ e2fsck -f /dev/sda2  
$ tune2fs -O ^metadata_csum /dev/sda2
```

#### Dosya sistemini kopyalama

Kurulum medyası **/cdrom** dizinine bağlanır. Kurulacak sistemin imajını bir dizine bağlayalım.

```
$ mkdir -p cdrom  
$ mkdir -p source  
$ mount -t iso9660 -o loop /dev/sr0 /cdrom/  
$ mount -t squashfs -o loop /cdrom/live/filesystem.squashfs /source
```

## Sistem Kurulumu

Şimdi de disk bölümümüzü bağlayalım.

```
$ mkdir -p target
$ mkdir -p /target/boot
$ mount /dev/sda2 /target
$ mount -t vfat /dev/sda1 /target/boot
```

Ardından dosyaları kopyalayalım.

```
# -p dosya izinlerini korur
# -r alt dizinlerle beraber kopyalar
# -f soru sormayı kapatır
# -v detaylı çıktıları gösterir
$ cp -prfv /source/* /target
```

Daha sonra diski senkronize edelim.

```
$ sync
```

## Bootloader kurulumu

grub kurulumu yapmak için grub paketinin kurulu olduğundan emin olun.

```
$ mkdir -p /target/dev
$ mkdir -p /target/sys
$ mkdir -p /target/proc
$ mkdir -p /target/run
$ mkdir -p /target/tmp
$ mount --bind /dev /target/dev
$ mount --bind /sys /target/sys
$ mount --bind /proc /target/proc
$ mount --bind /run /target/run
$ mount --bind /tmp /target/tmp

# Bunun yerine aşağıdaki gibi de girilebilir.
for dir in /dev /sys /proc /run /tmp ; do
mount --bind /$dir /target/$dir
done
$ chroot /target
```

## Grub Kuralım

```
# kurulu sistemden bağımsız çalışması için --removable kullanılır.
$ grub-install --removable --boot-directory=/boot --efi-directory=/boot /dev/sda
```

## Grub yapılandırması

1. /boot bölümünde initrd.img-<çekirdek-sürümü> dosyamızın olduğundan emin olalım.
2. /boot bölümünde vmlinuz-<çekirdek-sürümü> kernel dosyamızın olduğundan emin olalım.
3. /boot/grub/grub.cfg konumunda dostamızı oluşturalım(vi, touch veya nano ile).
4. dev/sda2 diskimizim uuid değerimizi bulalım.

## Sistem Kurulumu

```
$ blkid | grep /dev/sda2  
/dev/sda2: UUID="..." BLOCK_SIZE="4096" TYPE="ext4" PARTUUID="..."
```

Şimdi aşağıdaki gibi bir yapılandırma dosyası yazalım ve /boot/grub/grub.cfg dosyasına kaydedelim. Burada uuid değerini ve çekirdek sürümünü düzenleyin.

```
linux /vmlinuz-<çekirdek-sürümü>          root=UUID=<uuid-değeri> rw quiet  
initrd /initrd.img-<çekirdek-sürümü>  
boot
```

Ayrıca otomatik yapılandırma da oluşturabiliriz.

```
$ grub-mkconfig -o /boot/grub/grub.cfg
```

### OpenRc Disk İşlemi

Kullandığımız servis yöneticisi openrc ise **/etc/fstab** komunundaki dosyaya bakarak diske erişim sağlamaktadır. Bundan dolayı **fstab** dosyamızı aşağıdaki gibi yapılandırmalıyız.

### Fstab dosyası

Bu dosyayı doldurarak açılışta hangi disklerin bağlanacağını ayarlamalıyız. /etc/fstab dosyasını aşağıdakine uygun olarak doldurun.

#	<fs>	<mountpoint>	<type>	<opts>	<dump/pass>
	/dev/sda1	/boot	vfat	defaults,rw	0 1
	/dev/sda2	/	ext4	defaults,rw	0 1

**Not:** Disk bölümü konumu yerine **UUID=<uuid-değeri>** şeklinde yazmanızı öneririm. Bölüm adları değişebilirken uuid değerleri değişmez.

### Tek Bölüm Kurulum

Diskler üzerinde işlem yapabilmek için evdev veya udevd servisi çalışıyor olmalı. Ayrıca aşağıdaki modüllerin yüklü olduğundan emin olun. Anlatım boyunca **/dev/sda** diski üzerinden örnekleme yapılmıştır. Siz kendi diskinize göre düzenleyebilirsiniz. Disk ve isoya erişim için aşağıdaki modüllerin yüklü olduğundan emin olun.

- loop
- squashfs
- ext4 modülleri **modprobe** komutuyla yüklenmeli.

Disk Hazırlanmalı(legacy)

Öncelikle **cfdisk** veya **fdisk** komutları ile diski bölümlendirelim. Ben bu anlatımda **cfdisk** kullanacağım.

0. cfdisk komutuyla disk bölümlendirilmeli. .. code-block:: shell

```
$ cfdisk /dev/sda
```

1. dos seçilmeli
2. type linux system
3. write
4. quit
5. Bu işlem sonucunda sadece sda1 olur
6. mkfs.ext2 ile disk biçimlendirilir.

```
$ mkfs.ext2 /dev/sda1
```

Dosya sistemini kopyalama

Kurulum medyası **/cdrom** dizinine bağlanır. Kurulacak sistemin imajını bir dizine bağlayalım.

```
$ mkdir -p cdrom
$ mkdir -p source
$ mount -t iso9660 -o loop /dev/sr0 /cdrom/
$ mount -t squashfs -o loop /cdrom/live/filesystem.squashfs /source
```

Şimdi de disk bölümümüzü bağlayalım.

```
$ mkdir -p target
$ mount /dev/sda1 /target
$ mkdir -p /target/boot
```

Ardından dosyaları kopyalayalım.

```
# -p dosya izinlerini korur
# -r alt dizinlerle beraber kopyalar
# -f soru sormayı kapatır
# -v detaylı çıktıları gösterir
$ cp -prfv /source/* /target
```

Daha sonra diski senkronize edelim.

## Sistem Kurulumu

```
$ sync
```

### Bootloader kurulumu

grub kurulumu yapmak için grub paketini kurulu olduğundan emin olun.

```
$ mkdir -p /target/dev
$ mkdir -p /target/sys
$ mkdir -p /target/proc
$ mkdir -p /target/run
$ mkdir -p /target/tmp
$ mount --bind /dev /target/dev
$ mount --bind /sys /target/sys
$ mount --bind /proc /target/proc
$ mount --bind /run /target/run
$ mount --bind /tmp /target/tmp

# Bunun yerine aşağıdaki gibi de girilebilir.
for dir in /dev /sys /proc /run /tmp ; do
    mount --bind $dir /target/$dir
done
$ chroot /target
```

### Grub Kuralım

```
$ grub-install --boot-directory=/boot /dev/sda
```

### Grub yapılandırması

1. /boot bölümünde initrd.img-<çekirdek-sürümü> dosyamızın olduğundan emin olalım.
2. /boot bölümünde vmlinuz-<çekirdek-sürümü> kernel dosyamızın olduğundan emin olalım.
3. /boot/grub/grub.cfg konumunda dostamızı oluşturalım(vi, touch veya nano ile).
4. dev/sda1 diskimizin uuid değerimizi bulalım.

```
$ blkid | grep /dev/sda1
/dev/sda1: UUID="..." BLOCK_SIZE="4096" TYPE="ext4" PARTUUID="..."
```

Şimdi aşağıdaki gibi bir yapılandırma dosyası yazalım ve /boot/grub/grub.cfg dosyasına kaydedelim. Burada uuid değerini ve çekirdek sürümünü düzenleyin.

```
linux /boot/vmlinuz-<çekirdek-sürümü> root=UUID=<uuid-değeri> rw quiet
initrd /boot/initrd.img-<çekirdek-sürümü>
boot
```

Ayrıca otomatik yapılandırma da oluşturabiliriz.

```
$ grub-mkconfig -o /boot/grub/grub.cfg
```



## Sistem Kurulumu

### OpenRc Disk İşlemi

Kullandığımız servis yöneticisi openrc ise **/etc/fstab** komunundaki dosyaya bakarak diske erişim sağlamaktadır. Bundan dolayı **fstab** dosyamızı aşağıdaki gibi yapılandırmalıyız.

### Fstab dosyası

Bu dosyayı doldurarak açılışta hangi disklerin bağlanacağını ayarlamalıyız. /etc/fstab dosyasını aşağıdakine uygun olarak doldurun.

#	<fs>		<mountpoint>	<type>		<opts>	<dump/pass>
	/dev/sda1	/boot	vfat	defaults,rw	0	1	
	/dev/sda2	/	ext4	defaults,rw	0	1	

**Not:** Disk bölümü konumu yerine **UUID=<uuid-değeri>** şeklinde yazmanızı öneririm. Bölüm adları değişebilirken uuid değerleri değişmez.

### Uefi Sistem Kurulumu

Bu bölümde **Ext4** dosya sistemine grub kullanarak kurulum anlatılacaktır. Anlatım boyunca **/dev/sda** diski üzerinden örneklemeye yapılmıştır. Siz kendi diskinize göre düzenleyebilirsiniz. Diskler üzerinde işlem yapabilmek için evdev veya udevd servisi çalışıyor olmalı. Disk ve isoya erişim için aşağıdaki modüllerin yüklü olduğundan emin olun.

- loop
- squashfs
- ext4 modülleri **modprobe** komutuyla yüklenmeli.

Uefi - Legacy tespiti

**/sys/firmware/efi** dizini varsa uefi, yoksa legacy sisteme sahipsinizdir. Eğer uefi ise ia32 veya x86\_64 olup olmadığını anlamak için **/sys/firmware/efi/fw\_platform\_size** içeriğine bakın.

```
[[ -d /sys/firmware/efi/ ]] && echo UEFI || echo Legacy
[[ "64" == $(cat/sys/firmware/efi/fw_platform_size) ]] && echo x86_64 || ia32
```

Disk Hazırlanmalı

Uefi kullananlar ayrı bir disk bölümüne ihtiyaç duyarlar. Bu bölümü **fat32** olarak bölümlendirmeliler.

Bu anlatımda kurulum için **/boot** dizinini ayırmayı ve efi bölümü olarak aynı diski kullanmayı tercih edeceğiz. Öncelikle **cgdisk** veya **fdisk** komutları ile diski bölümlendirelim. Ben bu anlatımda **cgdisk** kullanacağım.

1. cgdisk komutuyla disk bölümlendirilmeli.

```
$ cgdisk /dev/sda
```

1. gpt seçilmeli
2. 512 MB type uefi alan(sda1)
3. geri kalanı type linux system(sda2)
4. write
5. quit
6. Bu işlem sonucunda sadece sda1 sda2 olur
7. mkfs.vfat ve mkfs.ext4 ile diskler biçimlendirilir.

```
$ mkfs.vfat /dev/sda1
$ mkfs.ext4 /dev/sda2
```

e2fsprogs Paketi

e2fsprogs paket sistemde mkfs.ext4, e2fsck, tune2fs vb sistem araçlarının yüklenmesini sağlar. Eğer sistemde bu sistem uygulamaları yoksa bu paketin yüklenmesi veya derlenmesi gerekmektedir.

Eğer /boot bölümünü ayırmayacaksanız grub yüklenirken **unknown filesystem** hatası almanız durumunda aşağıdaki yöntemi kullanabilirsiniz.

## Sistem Kurulumu

```
$ e2fsck -f /dev/sda2
$ tune2fs -O ^metadata_csum /dev/sda2
```

Dosya sistemini kopyalama

Kurulum medyası **/cdrom** dizinine bağlanır. Kurulacak sistemin imajını bir dizine bağlayalım.

```
$ mkdir -p cdrom
$ mkdir -p source
$ mount -t iso9660 -o loop /dev/sr0 /cdrom/
$ mount -t squashfs -o loop /cdrom/live/filesystem.squashfs /source
```

Şimdi de disk bölümümüzü bağlayalım.

```
$ mkdir -p target || true
$ mkdir -p /target/boot || true
$ mkdir -p /target/boot/efi || true
$ mount /dev/sda2 /target || true
$ mount /dev/sda1 /target/boot/efi
```

Ardından dosyaları kopyalayalım.

```
# -p dosya izinlerini korur
# -r alt dizinlerle beraber kopyalar
# -f soru sormayı kapatır
# -v detaylı çıktıları gösterir
$ cp -prfv /source/* /target
```

Daha sonra diski senkronize edelim.

```
$ sync
```

## Bootloader kurulumu

grub kurulumu yapmak için grub paketini kurulu olduğundan emin olun.

```
$ mkdir -p /target/dev
$ mkdir -p /target/sys
$ mkdir -p /target/proc
$ mkdir -p /target/run
$ mkdir -p /target/tmp
$ mount --bind /dev /target/dev
$ mount --bind /sys /target/sys
$ mount --bind /proc /target/proc
$ mount --bind /run /target/run
$ mount --bind /tmp /target/tmp
#efi alan bağlanıyor. Eğer uefi aktif edilmişse kernel **/sys/firmware/efi** tarafından budizinler ve dosyalar oluşuyor.
#sistem uefi değilse **/sys/firmware/efi** konumunda dosyalar olmayacaktır.
$ if [[ -d /sys/firmware/efi ]] ; then
    mount --bind /sys/firmware/efi/efivars /target/sys/firmware/efi/efivars
fi
# Bunun yerine aşağıdaki gibi de girilebilir.
for dir in /dev /sys /proc /run /tmp ; do
    mount --bind /$dir /target/$dir
done
$ chroot /target
```

Şimdi de uefi kullandığımız için efivar bağlayalım. .. code-block:: shell

```
$ mount -t efivarfs efivarfs /sys/firmware/efi/efivarfs
```

## Sistem Kurulumu

### Grub Kuralım

```
# biz /boot ayırdığımız ve efi bölümü olarak kullanacağız.  
# uefi kullanmayanlar --efi-directory belirtmemeliler.  
# kurulu sistemden bağımsız çalışması için --removable kullanılır.  
$ grub-install --removable --boot-directory=/boot --efi-directory=/boot --target=x86_64-efi /dev/sda
```

### Grub yapılandırması

1. /boot bölümünde initrd.img-<çekirdek-sürümü> dosyamızın olduğundan emin olalım.
2. /boot bölümünde vmlinuz-<çekirdek-sürümü> kernel dosyamızın olduğundan emin olalım.
3. /boot/grub/grub.cfg konumunda dostamızı oluşturalım(vi, touch veya nano ile).
4. dev/sda2 diskimizim uuid değerimizi bulalım.

```
$ blkid | grep /dev/sda2  
/dev/sda2: UUID="..." BLOCK_SIZE="4096" TYPE="ext4" PARTUUID="..."
```

Şimdi aşağıdaki gibi bir yapılandırma dosyası yazalım ve /boot/grub/grub.cfg dosyasına kaydedelim. Burada uuid değerini ve çekirdek sürümünü düzenleyin.

```
linux /vmlinuz-<çekirdek-sürümü>          root=UUID=<uuid-değeri> rw quiet  
initrd /initrd.img-<çekirdek-sürümü>  
boot
```

Ayrıca otomatik yapılandırma da oluşturabiliriz.

```
$ grub-mkconfig -o /boot/grub/grub.cfg
```

### OpenRc Disk İşlemi

Kullandığımız servis yöneticisi openrc ise **/etc/fstab** komunundaki dosyaya bakarak diske erişim sağlamaktadır. Bundan dolayı **fstab** dosyamızı aşağıdaki gibi yapılandırmalıyız.

### Fstab dosyası

Bu dosyayı doldurarak açılışta hangi disklerin bağlanacağını ayarlamalıyız. /etc/fstab dosyasını aşağıdakine uygun olarak doldurun.

#	<fs>		<mountpoint>	<type>		<opts>	<dump/pass>
/dev/sda1		/boot	vfat	defaults,rw	0	1	
/dev/sda2		/	ext4	defaults,rw	0	1	

**Not:** Disk bölümü konumu yerine **UUID=<uuid-değeri>** şeklinde yazmanızı öneririm. Bölüm adları değişebilirken uuid değerleri değişmez.

## Yardımcı Konular

### Qemu Kullanımı



Qemu Nedir?

Açık kaynaklı sanallaştırma aracıdır.

Kaynak dosyalarından kurulum için;

```
git clone https://gitlab.com/qemu-project/qemu.git
cd qemu
./configure
make
sudo make install
```

#### Sisteme Kurulum

```
sudo apt update
sudo apt install qemu-system-x86 qemu-utils
```

- 30GB bir disk oluşturup etahta.iso dosyamızı 2GB ramdan oluşan bir makina çalıştıralım.

```
qemu-img create disk.img 30G #30GB disk oluşturuldu.
qemu-system-x86_64 --enable-kvm -hda disk.img -m 2G -cdrom etahta.iso
```

- Oluşturulan sanal disk ve 2GB ram ile açma.

```
qemu-system-x86_64 --enable-kvm -hda disk.img -m 2G #sadece disk ile çalıştırılıyor
```

- Sistemi etahta.iso dosyamızı 2GB ramdan oluşan bir makina olarak çalıştıralım.

```
qemu-system-x86_64 -m 2G -cdrom etahta.iso #sadece iso doayası ile çalıştırma
```

#### Sistem Hızlandırılması

**--enable-kvm** eğer sistem disk ile çalıştırıldığında bu parametre eklenmezse yavaş çalışacaktır.

#### Boot Menu Açma

Sistemin diskten mi imajdan mı başlayacağını başlangıçta belirlemek için boot menu gelmesini istersek aşağıdaki gibi komut satırına seçenek eklemeliyiz.

```
qemu-system-x86_64 --enable-kvm -cdrom distro.iso -hda disk.img -m 4G -boot menu=on
```

## Yardımcı Konular

Uefi kurulum için:

```
sudo apt-get install ovmf
```

```
qemu-system-x86_64 --enable-kvm -bios /usr/share/ovmf/OVMF.fd -cdrom distro.iso -hda  
disk.img -m 4G -boot menu=on
```

qemu Host Erişimi:

kendi ipsi:10.0.2.15

ana bilgisayar 10.0.0.2 olarak ayarlıyor.

Kaynak: | <https://www.ubuntubuzz.com/2021/04/how-to-boot-uefi-on-qemu.html>

### Live Sistem Oluşturma

Canlı sistem oluşturma veya ram üzerinden çalışan sistem hazırlamak için SquashFS dosya sisteminde dağıtım sıkıştırılmalıdır. Bu bağlamda SquashFS dosya sistemi ve sıkıştırma nasıl yapılır bu dokümanda anlatılmaktadır.

#### SquashFS Nedir?

SquashFS, Linux işletim sistemlerinde sıkıştırılmış bir dosya sistemidir. Bu dosya sistemi, sıkıştırma algoritması kullanarak dosyaları sıkıştırır ve ardından salt okunur bir dosya sistemine dönüştürür. SquashFS, özellikle gömülü sistemlerde ve Linux dağıtımlarında kullanılan bir dosya sistemidir.

#### SquashFS Oluşturma

```
#mksquashfs input_source output/filesystem.squashfs -comp xz -wildcards mksquashfs initrd $HOME/distro/filesystem.squashfs -comp xz -wildcards
```

### Cdrom Erişimi

/dev/sr0, Linux işletim sistemlerinde bir CD veya DVD sürücüsünü temsil eden bir aygıt dosyasıdır. Bu dosya, CD veya DVD sürücüsünün fiziksel cihazını temsil eder ve kullanıcıların bu sürücüye erişmesini sağlar.

/dev/sr0 dosyası, Linux'un aygıt dosyası sistemi olan /dev dizininde bulunur. Bu dosya, Linux çekirdeği tarafından otomatik olarak oluşturulur ve sürücüye bağlı olarak farklı bir isim alabilir. Örneğin, ikinci bir CD veya DVD sürücüsü /dev/sr1 olarak adlandırılabilir.

Bu aygıt dosyası, kullanıcıların CD veya DVD'leri okumasına veya yazmasına olanak tanır. Örneğin, bir CD'yi okumak için aşağıdaki gibi bir komut kullanabilirsiniz:

```
$ cat /dev/sr0
```

#### Cdrom Bağlama

```
mkdir cdrom mount /dev/sr0 /cdrom
```

Bu işlem sonucunda cdrom bağlanmış olacaktır. iso dosyamızın içerisine erişebiliriz.

#### squashfs Dosyasını Bulma

Genellikle isoların içine squashfs dosyası oluşturulur. Bu sayede live yükleme yapılabilir. Örneğin /live/filesystem.squashfs benim imajlarımda böyle konumlandırıyorum.

#### squashfs Bağlama

squashfs dosyasını bağlamadan önce loop modülünün yüklü olması gerekmektedir. eğer yüklemeyerseniz

```
modprobe loop #loop modülü yüklenir.
```

```
mkdir canli mount -t squashfs -o loop cdrom/live/filesystem.squashfs /canli
```

### squashfs Sistemine Geçiş

Yukarıdaki adımlarda squashfs doayamızı /canli adında dizine bağlamış olduk. Bu aşamadan sonra sistemimizin bir kopyası olan squashfs canlıdan erişebilir veya sistemi buradan başlatabiliriz.

squashfs dosya sistemimize bağlanmak için;

```
chroot canli /bin/bash
```

Bu işlemin yerine `exec` komutuyla bağlanırsak sistemimiz id "1" değeriyle çalıştıracaktır. Eğer sistemin bu dosya sistemiyle açılmasını istiyorsak `exec` ile çalıştırıp id=1 olmasına dikkat etmeliyiz.

### busybox Nedir?

Busybox tek bir dosya halinde bulunan birçok araç seçeneğine sahip olan bir programdır. Bu araçlar `initramfs` sisteminde ve sistem genelinde sıkça kullanılabilir. Busybox aşağıdaki gibi kullanılır. Örneğin, dosya listelemek için `ls` komutunu kullanmak isterseniz:

```
$ busybox ls
```

Busyboxtaki tüm araçları sisteme sembolik bağ atmak için aşağıdaki gibi bir yol izlenebilir. Bu işlem var olan dosyaları sildiği için tehlikeli olabilir. Sistemin tasarımına uygun olarak yapılmalıdır.

```
$ busybox --install -s /bin # -s parametresi sembolik bağ olarak kurmaya yarar.
```

Busybox **static** olarak derlenmediği sürece bir libc kütüphanesine ihtiyaç duyar. `initramfs` içerisinde kullanılacaksa içerisine libc dahil edilmelidir. Bir dosyanın static olarak derlenip derlenmediğini öğrenmek için aşağıdaki komut kullanılır.

```
$ ldd /bin/busybox # static derlenmişse hata mesajı verir. Derlenmemişse bağımlılıklarını listeler.
```

Busybox derlemek için öncelikle **make defconfig** kullanılarak veya önceden oluşturduğumuz yapılandırma dosyasını atarak yapılandırma işlemi yapılır. Ardından eğer static derleme yaparsak yapılandırma dosyasına müdahale edilir. Son olarak **make** komutu kullanarak derleme işlemi yapılır.

```
$ make defconfig
$ sed -i "s|.*CONFIG_STATIC_LIBGCC.*|CONFIG_STATIC_LIBGCC=y|" .config
$ sed -i "s|.*CONFIG_STATIC.*|CONFIG_STATIC=y|" .config
$ make
```

Derleme bittiğinde kaynak kodun bulunduğu dizinde busybox dosyamız oluşmuş olur.

Static olarak derlemiş olduğumuz busyboxu kullanarak minimal kök dizin oluşturabiliriz. Burada static yapı kullanılmayacaktır. Sistemdeki `/bin/busybox` kullanılacaktır. Eğer yoksa busybox sisteme yüklenmelidir.

### kmod Nedir?

Linux çekirdeği ile donanım arasındaki haberleşmeyi sağlayan kod parçalarıdır. Bu kod parçalarını kernele eklediğimizde kerneli tekrardan derlememiz gerekmektedir. Her eklenen koddan sonra kernel derleme, kod çıkarttığımızda kernel derlemek ciddi bir iş yükü ve karmaşa yaratacaktır.

Bu sorunların çözümü için modul vardır. moduller kernele istediğimiz kod parçalarını ekleme ya da çıkartma yapabiliyoruz. Bu işlemleri yaparken kernel derleme işlemi yapmamıza gerek yok.

Kernele modul yükleme kaldırma için `kmod` aracı kullanılmaktadır. `kmod` aracı;

```
ln -s kmod /bin/depmod
ln -s kmod /bin/insmod
ln -s kmod /initrd/bin/lsmmod
ln -s kmod /bin/modinfo
```



```
ln -s kmod /bin/modprobe
ln -s kmod /bin/rmmod
```

şeklinde sembolik bağlarla yeni araçlar oluşturulmuştur.

**lsmod** : yüklü modülleri listeler

**insmod**: tek bir modul yükler

**rmmod**: tek bir modul siler

**modinfo**: modul hakkında bilgi alınır

**modprobe**: insmod komutunun aynısı fakat daha işlevseldir. module ait bağımlı olduğu modülleri de yüklemektedir. modprobe modülü /lib/modules/ dizini altında aramaktadır.

**depmod**: /lib/modules dizinindeki modüllerin listesini günceller. Fakat başka bir dizinde ise basedir=konum şeklinde belirtmek gerekir. konum dizininde /lib/modules/\*\* şeklinde kalsörler olmalıdır.

## Modul Yazma

hello.c dosyamız

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
MODULE_DESCRIPTION("Hello World examples");
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Bayram");
static int __init hello_init(void)
{
    printk(KERN_INFO "Hello world!\n");
    return 0;
}
static void __exit hello_cleanup(void)
{
    printk(KERN_INFO "remove module.\n");
}
module_init(hello_init);
module_exit(hello_cleanup);
```

Makefile dosyamız

```
obj-m+=my_module.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

modülün derlenmesi ve eklenip kaldırılması

```
make
```

```
insmod my_modul.ko // modül kernele eklendi.  
lsmod | grep my_modul //modül yüklendi mi kontrol ediliyor.  
rmmod my_modul // modül kernelden çıkartılıyor.
```

Not:

dmesg ile log kısmında eklendiğinde Hello Word yazısını ve kaldırıldığında modul ismini görebiliriz.

### sfdisk Nedir

sfdisk, Linux işletim sistemlerinde disk bölümlerini yönetmek için kullanılan bir komuttur. Disk bölümlerini oluşturmak, düzenlemek, silmek veya görüntülemek için sfdisk'i kullanabilirsiniz.

Disk bölümlerini görüntüleme:

Diskinizdeki mevcut bölümleri görüntülemek için sfdisk komutunu kullanabilirsiniz. Aşağıdaki komutu kullanarak mevcut bölümleri listeleyebilirsiniz:

```
sfdisk -l /dev/sda
```

Disk bölümleri oluşturma:

Yeni bir disk bölümü oluşturmak için sfdisk komutunu kullanabilirsiniz. Örneğin, /dev/sda üzerinde yeni bir bölüm oluşturmak için aşağıdaki komutu kullanabilirsiniz:

```
echo ".,L" | sfdisk /dev/sda
```

Bu komut, kullanılabilir tüm alanı kullanarak bir bölüm oluşturacaktır.

Disk bölümlerini silme:

Bir disk bölümünü silmek için sfdisk komutunu kullanabilirsiniz. Örneğin, /dev/sda üzerindeki bir bölümü silmek için aşağıdaki komutu kullanabilirsiniz:

```
echo ".,L" | sfdisk --delete /dev/sda
```

Bu komut, belirtilen bölümü silecektir.

sfdisk komutunun daha fazla seçeneği ve kullanımı vardır. Daha fazla bilgi için sfdisk komutunun man sayfasını inceleyebilirsiniz:

Bazı Örnekler

Örnek1:

```
sfdisk /dev/sdf <<EOF
```

```
0,512 ,512 ; EOF
```

```
/dev/sdf1 0+ 511 512- 4112639+ 83 Linux /dev/sdf2 512 1023 512 4112640 83 Linux /dev/sdf3  
1024 1043 20 160650 83 Linux
```

Örnek2:

```
sfdisk /dev/vda << EOF label: dos label-id: 0xaaaaaaaa # comment: start=, size= 50M, type=  
7 , bootable start=, size= 650M, type= 27 start=, size= 45G , type= 7 EOF
```

## Yardımcı Konular

### Örnek3:

Bu örnekte ilk bölüm 1GB ve ikinci bölüm ise diskin geri kalan kısmıdır. `echo -e "label: gptn,1GiBn," | sudo sfdisk /dev/vda veya sfdisk /dev/vda << EOF label: gpt ,1GiB , EOF`

### Örnek4:

`my.layout # partition table of /dev/sda unit: sectors`

`/dev/sda1 : start= 2048, size= 497664, Id=83, bootable /dev/sda2 : start= 501758, size=1953021954, Id= 5 /dev/sda3 : start= 0, size= 0, Id= 0 /dev/sda4 : start= 0, size= 0, Id= 0 /dev/sda5 : start= 501760, size=1953021952, Id=8e`

Aynı disk bölümlemesini ve düzenini başka bir aygıtı uygulamak için:

`sfdisk /dev/sdb < my.layout`

## İmza Doğrulama

GPG (GNU Privacy Guard), dosyaların ve iletişimin güvenliğini sağlamak için kullanılan bir şifreleme aracıdır. Bu araçla, dosyaları şifreleyebilir, imzalayabilir ve imzaları doğrulayabiliriz.

### İmza Oluşturma

GPG ile anahtar oluşturmak oldukça basittir. İşte adım adım nasıl yapılacağı: İlk olarak, GPG yazılımını sisteminize yüklemeniz gerekmektedir. Linux tabanlı bir işletim sistemi kullanıyorsanız, terminali açın ve aşağıdaki komutu çalıştırın ve GPP kurulumunu yapınız.

`language-bash`

`sudo apt-get install gnupg`

GPG anahtar çiftini oluşturmak için aşağıdaki komutu kullanın:

`language-bash`

`gpg --full-generate-key`

### Belge İmzalama

Anahtar çifti oluşturulduktan sonra, imzalamak istediğiniz belgeyi seçin ve aşağıdaki komutu kullanarak belgeyi imzalayın:

`language-bash`

`gpg --sign belge.txt`

1. İmzalanan belge, aynı dizinde "belge.txt.asc" uzantısıyla kaydedilecektir. Bu imzalı belgeyi başkalarıyla paylaşabilirsiniz.

### İmzalı Belge Doğrulama

- İmzalı belgeyi doğrulamak istediğinizde, aşağıdaki komutu kullanarak GPG'yi kullanabilirsiniz:

`language-bash`

`gpg --verify belge.txt.asc`

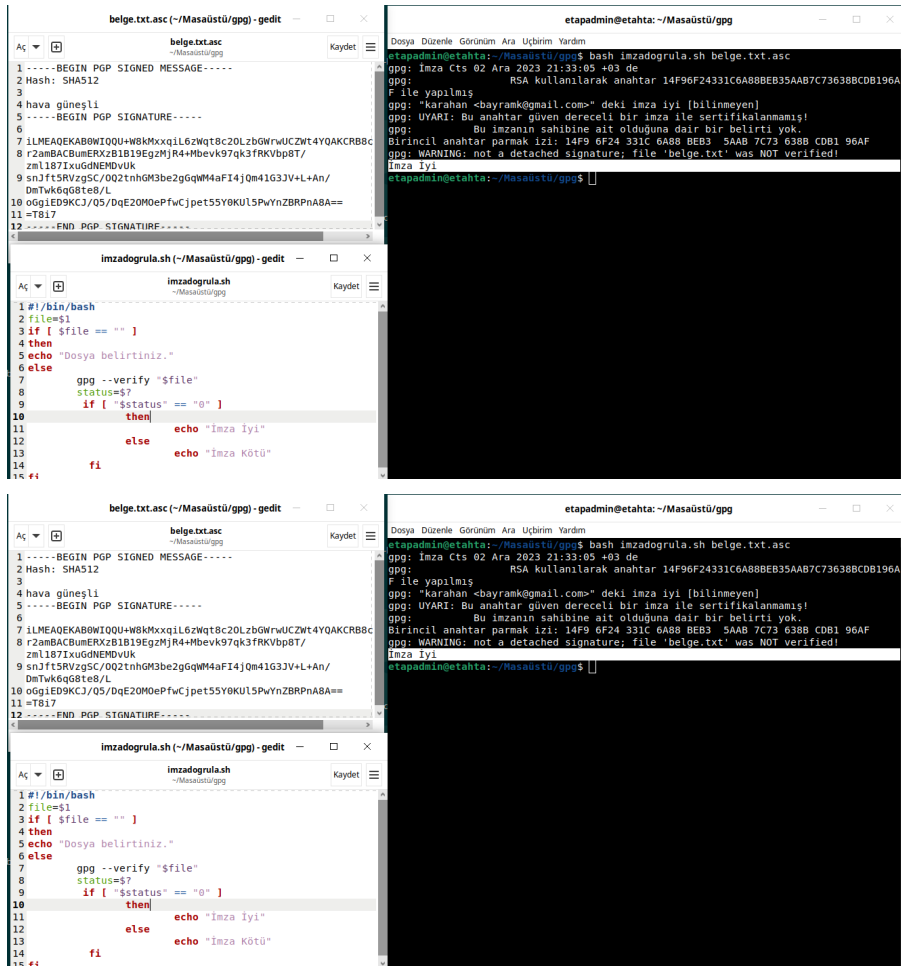
Bu komut, belgenin doğruluğunu kontrol edecek ve imzanın geçerli olup olmadığını size bildirecektir. İmza doğrulama işlemleri daha detaylı bir şekilde aşağıda anlatılmıştır.

### bash ile Doğrulama

bash script ile imza doğrulaması aşağıdaki kodlarla yapılabilir.

## Yardımcı Konular

```
#!/bin/bash
file=$1
if [ $file == "" ]
then
echo "Dosya belirtiniz."
else
    gpg --verify "$file"
    status=$?
    if [ "$status" == "0" ]
    then
        echo "İmza İyi"
    else
        echo "İmza Kötü"
    fi
fi
```



```
etapadmin@etahta: ~/Masaüstü/gpg
Dosya Düzenle Görünüm Ara Uçbirim Yardım
etapadmin@etahta:~/Masaüstü/gpg$ bash imzadogrula.sh belge.txt.asc
gpg: İmza Cts 02 Ara 2023 21:33:05 +03 de
gpg: RSA kullanılarak anahtar 14F96F24331C6A88BEB35AAB7C73638BCDB196A
F ile yapılmış
gpg: "karahan <bayramk@gmail.com>" deki imza iyi [bilinmeyen]
gpg: UYARI: Bu anahtar güven dereceli bir imza ile sertifikalanmamış!
gpg: Bu imzanın sahibine ait olduğuna dair bir belirti yok.
Birincil anahtar parmak izi: 14F9 6F24 331C 6A88 BEB3 5AAB 7C73 638B CDB1 96AF
gpg: WARNING: not a detached signature; file 'belge.txt' was NOT verified!
İmza İyi
etapadmin@etahta:~/Masaüstü/gpg$
```

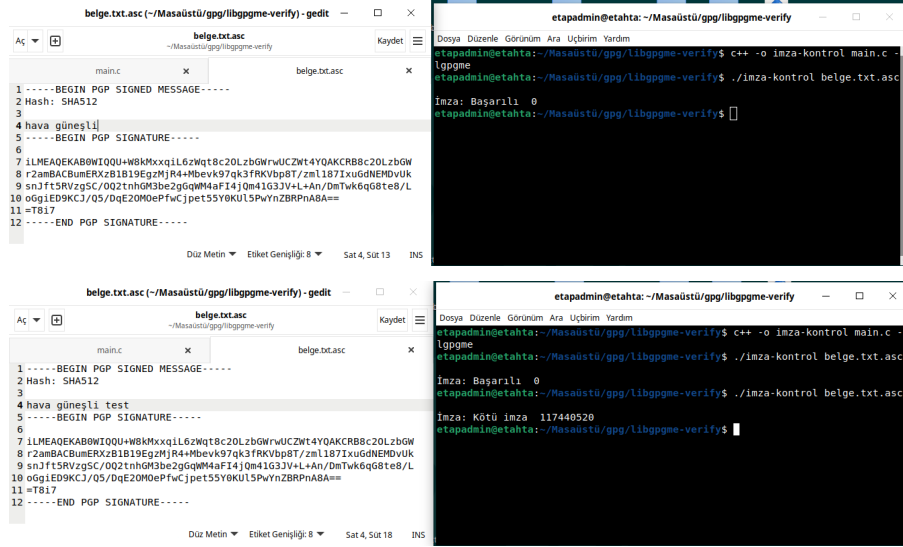
```
belge.txt.asc (~/Masaüstü/gpg) - gedit
Aç Kaydet
1 -----BEGIN PGP SIGNED MESSAGE-----
2 Hash: SHA512
3
4 hava güneşli
5 -----BEGIN PGP SIGNATURE-----
6
7 lLMEAEKABOWIQU+WBkMxqilL6Zwqt8c20Lz6WrwUCZwt4YQAKCRB8C
8 r2amBACBumERXzB1B19EgZMJR4+Hbevk97qk3fRKVbp8T/
9 zml187ixugdNEHdvUk
9 snJftSRVzgsC/Q0ZtnhGM3be2GqW4aFI4j0m4IG3JV+L+An/
DmTwk6qG8te8/L
10 oGgiED9KCJ/Q5/DqE2OMePfwCjpet5SY8KUL5PwYnZBRPnA8A==
11 =T817
12 -----END PGP SIGNATURE-----

imzadogrula.sh (~/Masaüstü/gpg) - gedit
Aç Kaydet
1 #!/bin/bash
2 file=$1
3 if [ $file == "" ]
4 then
5 echo "Dosya belirtiniz."
6 else
7     gpg --verify "$file"
8     status=$?
9     if [ "$status" == "0" ]
10    then
11        echo "İmza İyi"
12    else
13        echo "İmza Kötü"
14    fi
15 fi
```

## Yardımcı Konular

### c++ ile Doğrulama

c kullanarak özünde bash komutunu sonucunu kontrol eden imza doğrulaması aşağıdaki kodlarla yapılabilir.



```
etapadmin@etahta: ~/Masaüstü/gpg/libpgpgme-verify
gpg --verify belge.txt.asc
İmza: Başarılı 0
etapadmin@etahta: ~/Masaüstü/gpg/libpgpgme-verify
```

```
#include <iostream> #include <cstdlib>
```

```
int main() {
```

```
    int result = system("gpg --verify belge.txt.asc"); if (result == 0) {
```

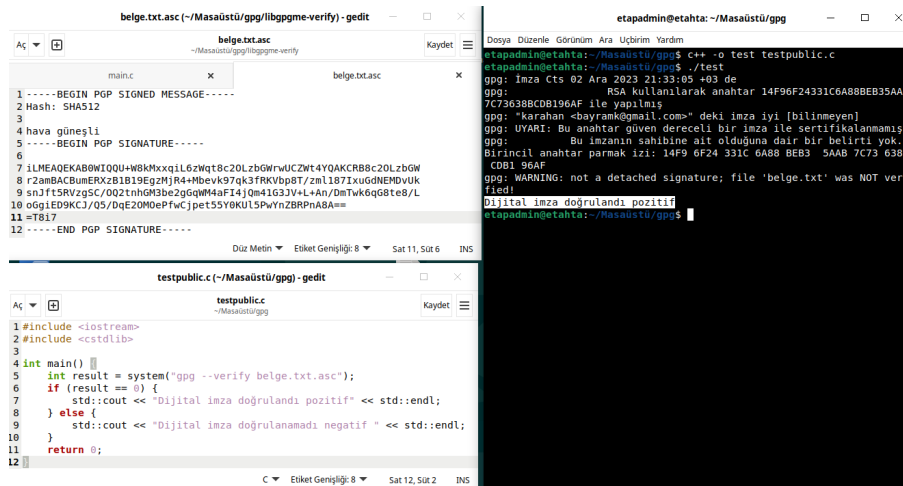
```
        std::cout << "Dijital imza doğrulandı pozitif" << std::endl;
```

```
    } else {
```

```
        std::cout << "Dijital imza doğrulanamadı negatif " << std::endl;
```

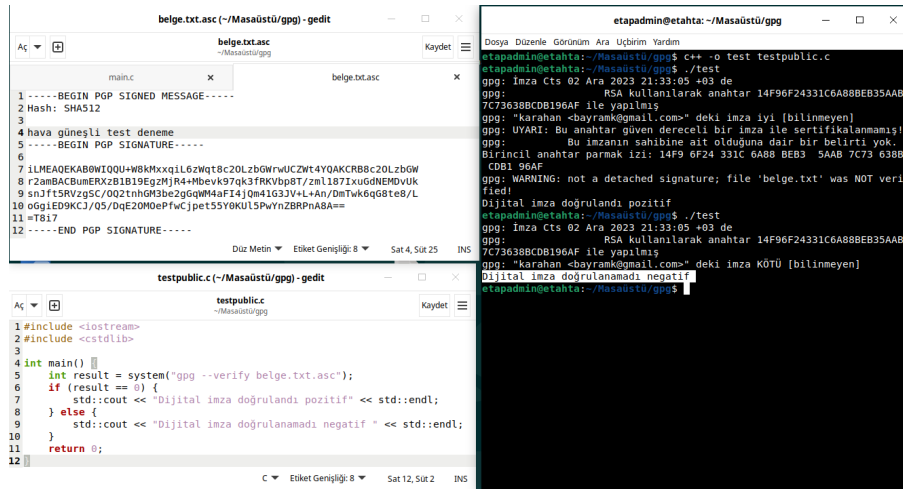
```
    } return 0;
```

```
}
```



```
etapadmin@etahta: ~/Masaüstü/gpg
gpg --verify belge.txt.asc
İmza: Başarılı 0
etapadmin@etahta: ~/Masaüstü/gpg
```

## Yardımcı Konular



```
belge.txt.asc (~/.Masaüstü/gpg) - gedit
1 -----BEGIN PGP SIGNED MESSAGE-----
2 Hash: SHA512
3
4 hava güneşli test deneme
5 -----BEGIN PGP SIGNATURE-----
6
7 iLMEAOEKAB0W1QOU+H8kMxqiL6zwt8c20LzbGWwUCZwt4Y0AKCRB8c20LzbGW
8 r2anBACBumERXz81B19EgZMjR4+HbeV97q3fRKVbp8T/zm1187IXuGdNEMDVUk
9 snJftSRVz9Sc/QQ2tnhgM3be2G6qM4aFI4j0m41G3JV+L+An/DmTwk6qG8te8/L
10 oGgiED9KCJ/05/DqE20M0ePfwCjpet5SY0KUL5PwYnZBRPnA8A==
11 =T817
12 -----END PGP SIGNATURE-----

testpublic.c (~/.Masaüstü/gpg) - gedit
1 #include <iostream>
2 #include <cstdlib>
3
4 int main() {
5     int result = system("gpg --verify belge.txt.asc");
6     if (result == 0) {
7         std::cout << "Dijital imza doğrulandı pozitif" << std::endl;
8     } else {
9         std::cout << "Dijital imza doğrulanamadı negatif" << std::endl;
10    }
11    return 0;
12 }
```

```
etapadmin@etahta: ~/.Masaüstü/gpg
Dosya Düzenle Görünüm Ara Uçbirim Yardım
etapadmin@etahta: ~/.Masaüstü/gpg$ c++ -o test testpublic.c
etapadmin@etahta: ~/.Masaüstü/gpg$ ./test
gpg: Imza Cts 02 Ara 2023 21:33:05 +03 de
gpg:
gpg: RSA kullanılarak anahtar 14F96F24331C6A88BE835AAB
7C73638BCDB196AF ile yapılmış
gpg: "karahan <bayramk@gmail.com>" deki imza iyi [bilinmeyen]
gpg: UYARI: Bu anahtar güven dereceli bir imza ile sertifikalanmamış!
gpg: Bu imzanın sahibine ait olduğuna dair bir belirti yok.
Birincil anahtar parmak izi: 14F9 6F24 331C 6A88 BEB3 5AAB 7C73 638B
CDB1 96AF
gpg: WARNING: not a detached signature; file 'belge.txt' was NOT veri-
fied!
Dijital imza doğrulandı pozitif
etapadmin@etahta: ~/.Masaüstü/gpg$ ./test
gpg: Imza Cts 02 Ara 2023 21:33:05 +03 de
gpg:
gpg: RSA kullanılarak anahtar 14F96F24331C6A88BE835AAB
7C73638BCDB196AF ile yapılmış
gpg: "karahan <bayramk@gmail.com>" deki imza KÖTÜ [bilinmeyen]
Dijital imza doğrulanamadı negatif
etapadmin@etahta: ~/.Masaüstü/gpg$
```

## Yardımcı Konular

### c++ libpgpme ile Doğrulama

libpgpme kütüphanelerini kullanarak bir belge doğrulama yapabiliriz.

```
#include <stdio.h>
#include <pgpme.h>
#include <locale.h>
#include <stdlib.h>
#include <string.h>

int print_engine_info() {
    pgpme_engine_info_t info;
    pgpme_error_t err;

    err = pgpme_get_engine_info(&info);
    if (err != GPG_ERR_NO_ERROR) {
        fprintf(stderr, "ERROR: Filed to get engine info!\n");
        return -1;
    }
    printf( "Installed engines: {\n" );
    while(info != NULL) {
        printf( "\t* %s Protocol=%s Version=%s Required-Version=%s Home=%s\n",
            info->file_name, pgpme_get_protocol_name(info->protocol),
            info->version, info->req_version, info->home_dir );
        info = info->next;
    }
    printf("}\n");
    return 0;
}

int main(int argc, const char* argv[]) {
    const char *pgpme_version, *pgpme_prot;
    pgpme_error_t err;
    pgpme_ctx_t ctx;
    FILE *fp_sig=NULL, *fp_msg=NULL;
    pgpme_data_t sig=NULL, msg=NULL, plain=NULL, text=NULL;
    pgpme_verify_result_t result;

    pgpme_protocol_t protocol = GPGME_PROTOCOL_OpenPGP;

    /* GPGME version check and initialization */
    setlocale(LC_ALL, "");

    pgpme_version = pgpme_check_version(GPGME_VERSION);    // developed for 1.5.1
    if (!pgpme_version) {
        fprintf(stderr, "ERROR: Wrong library on target! Please "
            "install at least version %s!\n", GPGME_VERSION);
        exit(1);
    }
    pgpme_set_locale(NULL, LC_CTYPE, setlocale(LC_CTYPE, NULL));
#ifdef LC_MESSAGES
    pgpme_set_locale(NULL, LC_MESSAGES, setlocale(LC_MESSAGES, NULL));
#endif
}
```

```
/* Protocol check */
pgpme_prot = pgpme_get_protocol_name(protocol);
err = pgpme_engine_check_version(protocol);
if (!pgpme_prot || err != GPG_ERR_NO_ERROR) {
    fprintf(stderr, "ERROR: libpgpme lacks of OpenPGP protocol!\n");
    print_engine_info();
    exit(1);
}

fp_sig = fopen(argv[1], "rb");
if (!fp_sig) {
```

```
    fprintf(stderr, "ERROR: Failed to open '%s'!\n", argv[0]);
    exit(1);
}
```

## Yardımcı Konular

```
}
if (argc > 2)
{
    fp_msg = fopen(argv[2], "rb");
    if (!fp_msg)
    {
        fprintf(stderr, "ERROR: Failed to open '%s'!\n", argv[1]);
        exit(1);
    }
}

err = gpgme_new(&ctx);
if (err != GPG_ERR_NO_ERROR) {
    char buf[4096];
    gpgme_strerror_r(err, buf, 4096);
    fprintf(stderr, "ERROR: %s\n", buf);
    exit(1);
}

gpgme_set_protocol(ctx, protocol);

err = gpgme_data_new_from_stream(&sig, fp_sig);
if (err) {
    fprintf(stderr, "ERROR allocating data object: %s\n", gpgme_strerror(err));
    exit(1);
}

if (fp_msg)
{
    err = gpgme_data_new_from_stream(&msg, fp_msg);
    if (err) {
        fprintf(stderr, "ERROR allocating data object: %s\n", gpgme_strerror(err));
        exit(1);
    }
    printf("Loaded message from '%s'\n", argv[2]);
}
else
{
    err = gpgme_data_new(&plain);
    if (err) {
        fprintf(stderr, "ERROR allocating data object: %s\n", gpgme_strerror(err));
        exit(1);
    }
    ///printf("Allocated 'plain' data\n");
}

err = gpgme_op_verify(ctx, sig, msg, plain);
if (err)
{
    fprintf(stderr, "ERROR: signing failed: %s\n", gpgme_strerror(err));
    exit(1);
}

result = gpgme_op_verify_result(ctx);
int count = 0;
```

```
if (result) {
    gpgme_signature_t sig;

    for(sig = result->signatures; sig; sig = sig->next)
    {
        count += 1;
        if ( !(sig->summary & GPGME_SIGSUM_VALID) ) {
            printf("İmza: %s %d\n", gpgme_strerror(sig->status), sig->status);

            exit(1);
        }
    }
}
```

```
}
```



```
if (count < 1) {
    printf( "Error: Cannot find matching signature!\n" );
    return 1;
}

printf( "\nSignature verification successful. Plaintext:\n" );

text = plain ? plain : msg;
gpgme_data_seek(text, 0, SEEK_SET);
size_t bytes;
do {
    char buffer[256];
    bytes = gpgme_data_read(text, buffer, 256-1);
    buffer[bytes] = '\0';

    printf( "%s", buffer );
} while( bytes > 0 );

gpgme_data_release(plain);
gpgme_data_release(msg);
gpgme_data_release(sig);

gpgme_release(ctx);

return 0;
}
```

## Kernel Modul Derleme

Kernel linux sistemlerinin temel dosyasıdır.

Kaynak Dosya İndirme

# Linux çekirdeğinin kaynak kodunu <https://kernel.org> üzerinden indirin.

```
wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.6.10.tar.xz
tar -xvf linux-6.6.10.tar.xz
cd linux-6.6.10
```

### Kernel Ayarları

Arşiv açıldıktan sonra arşivin açıldığı dizinde **.config** dosyası oluşturmamızdır. Bu dosyada hangi ayarlara göre derleme yapılacağını belirten parametreler var. Eğer temel(varsayılan) ayarlarda derlenmesini istersek **make defconfig** komutuyla **.config** adında bir dosya oluşacaktır.

Ben kendim belirleyeceğim bu ayarları diyorsak **make menuconfig** komutuyla açılan ekrandan ayarlamaları yapıp ayarları kaydetmeliyiz. Ayarlar kaydedilince **.config** dosyası oluşacaktır.

Bunların dışında ben Arch, Debian vb. dağıtımların **.config** dosyasını kullanacağım da diyebilirsiniz. Burada Arch Linux **.config** dosyasını kullanacağız.

[https://gitlab.archlinux.org/archlinux/packaging/packages/linux/-/blob/main/config?ref\\_type=heads](https://gitlab.archlinux.org/archlinux/packaging/packages/linux/-/blob/main/config?ref_type=heads)

Bu adresten indirilen **config** dosyasını **.config** olarak tarball(tar uzantılı sıkıştırılmış) dosyasının açıldığı dizine kopyalayalım.

### Kernel Derleme

**.config** ayarlamaları yapıldıktan sonra derleme yapılacak. Derleme aşağıdaki komutla yapılır.

```
make bzImage # tek çekirdekle derleme yapacak yavaş olur
#make bzImage -j$(proc) #işlemci çekirdek sayısını sistemden alarak yapıyor, hızlı olur
```

Derleme işlemi biraz zaman alacaktır.

Derleme tamamlandığında Kernel: arch/x86/boot/bzImage is ready (#1) şeklinde bir satır yazmalıdır. Kernelimizin düzgün derlenip derlenmediğini anlamak için aşağıdaki komutu kullanabilirsiniz.

```
file arch/x86/boot/bzImage arch/x86/boot/bzImage: Linux kernel x86 boot executable bzImage,
version 6.6.2 (etapadmin@etahta) #1 SMP PREEMPT_DYNAMIC Sat Nov 25 19:53:19 +03 2023,
RO-rootFS, swap_dev 0XC, Normal VGA
```

Kernel derledikten sonra kendi sistemimizde kullanmak için **/arch/x86/boot/bzImage** konumundan alıp kullanabiliriz. Derlenen kernel'e uygun modüllerin derlenmesi gerekmektedir.

### Modul Derleme

Modul ise kernel ile donanım arasında iletişim kuran yazılımlardır. Modüller kernel versiyonuyla aynı versiyonda olmalıdır. Başka versiyon kernel'ler derlenen modül kernel tarafından kullanılamaz. Bundan dolayı kullanılacak modüllerin kernel versiyonuna uygun derlenmesi lazımdır.

```
make modules # tek çekirdekle derleme yapacak yavaş olur
#make modules -j$(proc) #işlemci çekirdek sayısını sistemden alarak yapıyor, hızlı olur
```

## Servis Yönetimi

Derleme işlemi biraz zaman alacaktır. Modüller derlendikten sonra sisteme aşağıdaki komutla yüklenir.

### Modül Yükleme

Modüller istenilen konuma yüklenebilir. Yükleme konumunu **INSTALL\_MOD\_PATH** parametresiyle belirleyebiliriz. Eğer belirmezsek /lib/konumuna yüklenecektir.

```
INSTALL_MOD_PATH="$HOME/distro/initrd" make modules_install
```

# Terminal Yönlendirmesi

### İlk terminalde :

```
$ tty /dev/pts/0 $ <no need to run any command here, just see the output>
```

### İkinci terminalde :

```
$ ls > /dev/pts/0
```

Artık çıktığı ilk terminalde alıyorsunuz

**Başka Yol:** \$ tty /dev/pts/0

```
$ tty /dev/pts/1
```

Bu TTY'leri varsayarak, birincinin stdout'unu ikinciye yönlendirmek için bunu ilk terminalde çalıştırın:

```
exec 1>/dev/pts/1
```

Not: Artık her komut çıktısı pts/1'de gösterilecektir.

pts/0'ın varsayılan davranış stdout'unu geri yüklemek için:

```
exec 1>/dev/pts/0
```

### Gerçek Zamanlı Terminal Yansıtma

terminal 1'de:

```
$ tty /dev/pts/0
```

terminal 2'de:

```
$ tty /dev/pts/1
```

### terminal 2'de:

```
$ exec &> >(tee >(cat >&/dev/pts/0)) ls
```

Çıktı, siz yazarken bile her iki terminalde de gerçek zamanlı olarak gösterilecektir.

## Servis Yönetimi

### OpenRC

Openrc sistem açılışında çalışacak uygulamaları çalıştıran servis yöneticisidir.

### Kurulum

Kaynak koddan derlemek için aşağıdaki adımları izlemelisiniz:

```
$ git clone https://github.com/OpenRC/openrc
$ cd openrc
```

## Servis Yönetimi

```
$ meson setup build --prefix=/usr
$ ninja -C build install
```

### Çalıştırılması

Openrc servis yönetiminin çalışması için boot parametrelerine yazılması gerekmektedir. **/boot/grub.cfg** içindeki **linux /vmlinuz init=/usr/sbin/openrc-init root=/dev/sdax** olan satırda **init=/usr/sbin/openrc-init** yazılması gerekmektedir. Artık sistem openrc servis yöneticisi tarafından uygulamalar çalıştırılacak ve sistem hazır hale getirilecek.

### Basit kullanım

Servis etkinleştirip devre dışı hale getirmek için **rc-update** komutu kullanılır. Aşağıda **udhcpc** internet servisi örnek olarak gösterilmiştir. **/etc/init.d/** konumunda **udhcpc** dosyamızın olması gerekmektedir.

```
# servis etkinleştirmek için
$ rc-update add udhcpc boot
# servisi devre dışı yapmak için
$ rc-update del udhcpc boot
# Burada udhcpc servis adı boot ise runlevel adıdır.
```

Servisleri başlatıp durdurmak için ise **rc-service** komutu kullanılır.

```
$ rc-service udhcpc start
# veya şu şekilde de çalıştırılabilir.
$ /etc/init.d/udhcpc start
```

Servislerin durumunu öğrenmek için **rc-status** komutu kullanılır. Ayrıca sistemdeki servislerin sonraki açılışta hangisinin başlatılacağını öğrenmek için ise parametresiz olarak **rc-update** kullanabilirsiniz.

```
# şu an hangi servislerin çalıştığını gösterir
$ rc-status
# sonraki açılışta hangi servislerin çalışacağını gösterir
$ rc-update
```

Sistemi kapatmak veya yeniden başlatmak için **openrc-shutdown** komutunu kullanabilirsiniz.

```
# kapatmak için
$ openrc-shutdown -p 0
# yeniden başlatmak için
$ openrc-shutdown -r 0
```

### Servis dosyası

Openrc servis dosyaları basit birer **bash** betiğidir. Bu betikler **openrc-run** komutu ile çalıştırılır ve çeşitli fonksiyonlardan oluşabilir. Servis dosyaları **/etc/init.d** içerisinde bulunur. Servisleri ayarlamak için ise **/etc/conf.d** içerisine aynı isimle ayar dosyası oluşturabiliriz.

Çalıştırılacak komut komut parametreleri ve **pidfile** dosyamızı aşağıdaki gibi belirtebiliriz.

```
description="Örnek servis"
command=/usr/bin/ornek-servis
```

## Servis Yönetimi

```
command_args=--parametre  
pidfile=/run/ornek-servis.pid
```

Bununla birlikte **start**, **stop**, **status**, **reload**, **start\_pre**, **stop\_pre** gibi fonksiyonlar da yazabiliriz.

```
...  
start(){  
    ebegin "Starting ${RC_SVCNAME}"  
    start-stop-daemon --start --pidfile "/run/servis.pid" --exec /usr/bin/ornek-servis --parametre  
}  
...
```

Servis bağımlılıklarını belirtmek için ise **depend** fonksiyonu kullanılır.

```
...  
depend() {  
    need localmount  
    after dbus  
}  
...
```