

distro Dokümanı

version

distro Linux

Temmuz 29, 2024

Contents

Dağıtım	1
Temel Kavramlar	1
Temel Kavramlar	1
Linux Nedir?	1
Açık Kaynak Nedir?	1
Dağıtım Nedir ve Türleri	1
Ön Hazırlık	2
Dağıtım Ortamın Hazırlanması	2
Minimal Dağıtım Oluşturma	3
Dizinlerin Oluşturulması	3
\$HOME/distro/rootfs/busybox	4
\$HOME/distro/rootfs/init	4
\$HOME/distro/iso/boot/initrd.img	5
\$HOME/distro/iso/boot/vmlinuz	5
\$HOME/distro/iso/boot/grub/grub.cfg	5
\$HOME/distro/distro.iso	5
Dağıtımın Test Edilmesi	5
busybox Nedir?	7
Busybox Derleme	7
Chroot Nedir?	8
Bağımlılık Scripti	9
Basit Sistem Oluşturma	9
Sistem Dizin Oluşturulması	10
ls Komutu	10
rmdir Komutu	10
mkdir Komutu	11
bash Komutu	11
chroot Sistemde Çalışma	11
Temel Paketleri Derleme	12
Temel Paketler	12
glibc Nedir?	14
glibc Derleme	14
glibc Yükleme	14
glibc Test Etme	14
Program Derleme	14
Program Yükleme	14
Programı Test Etme	14
Hata Çözümü	15
libreadline	16
libreadline Derleme	16
Program Derleme	16

Program Derleme	16
Program Test Etme	16
ncurses	17
ncurses Derleme	17
kmod	18
kmod Derleme	18
Kmod'u derleme için hazırlayalım:	18
kmod Araçlarını Oluşturma	19
kmod Test Edilmesi	19
util-linux	20
util-linux Derleme	20
eudev	21
eudev Derleme	21
busybox Nedir?	22
e2fsprogs Paketi	23
Grub Nedir?	24
grub Derleme	24
zlib Nedir?	25
zlib Derleme	25
initramfs-tools Nedir	26
initramfs-tools Ayarları	26
initramfs-tools Güncelleme	26
initrd açılma Süreci	26
initrd script İçeriği	27
OpenRC	28
Derleme	28
Çalıştırılması	28
Basit kullanım	28
Paket Sistemi Tasarlama	29
Paket Sitemi	29
Binary Paket Sistemi	29
Source Paket Sistemi	29
Paket sisteminin temel yapısı	29
bps Paket Sistemi	29
Paket Oluşturma	30
bpsbuild Dosyası	30
bpspaketle Dosyası	30
bpsbuild Dosyamızın Son Hali	31
bpspaketle Dosyamızın Son Hali	32
Paket Yapma	33
Depo indexleme	34
bps Paket Liste İndexi Güncelleme	34
index.lst Dosyasını Oluşturma	35

index.lst Dosyasını Güncelleme	35
Paket Kurma	36
bps Paket Kurma Scripti Tasarlama	36
bpskur Scripti	37
bpskur Scriptini Kullanma	37
Paket Kaldırma	38
bps Paket Kaldırma Scripti Tasarlama	38
bpskaldır scripti	38
bpskaldır Kullanma	39
initrd Hazırlama	40
initrd	40
Temel Dosyalar	40
linux Açılış Süreci	40
initrd Dosya İçeriği	41
initrd Scripti	42
S1- \$boot/bin/busybox	42
S2-S8 \$boot/bin/kmod	43
S9- \$boot/lib/modules/\$(uname -r)/moduller	43
S10-S13- \$boot/bin/udevadm	43
S14- distro/initrd/bin/init	44
init Dosyası	44
S15- distro/iso/initrd.img	45
S16- distro/iso/vmlinuz	45
S17- distro/iso/grub/grub.cfg	45
iso Hazırlama	46
İso Hazırlama	46
filesystem.squashfs Hazırlama	46
İso Dosyasının Oluşturulması	46
Sistem Kurulumu	47
Sistem Kurma	47
Tek Bölüm Kurulum	48
Disk Hazırlanmalı(legacy)	48
Dosya sistemini kopyalama	48
Bootloader kurulumu	49
Grub Kuralım	49
Grub yapılandırması	49
OpenRc Disk İşlemi	50
Fstab dosyası	50
İki Bölüm Kurulum	51
Disk Hazırlanmalı	51
e2fsprogs Paketi	51
Dosya sistemini kopyalama	51
Bootloader kurulumu	52

Grub Kuralım	52
Grub yapılandırması	52
OpenRc Disk İşlemi	53
Fstab dosyası	53
Tek Bölüm Kurulum	54
Disk Hazırlanmalı(legacy)	54
Dosya sistemini kopyalama	54
Bootloader kurulumu	55
Grub Kuralım	55
Grub yapılandırması	55
OpenRc Disk İşlemi	56
Fstab dosyası	56
Uefi Sistem Kurulumu	57
Uefi - Legacy tespiti	57
Disk Hazırlanmalı	57
e2fsprogs Paketi	57
Dosya sistemini kopyalama	58
Bootloader kurulumu	58
Grub Kuralım	59
Grub yapılandırması	59
OpenRc Disk İşlemi	59
Fstab dosyası	59
Uefi Sistem Kurulumu	60
Uefi - Legacy tespiti	60
Disk Hazırlanmalı	60
e2fsprogs Paketi	60
Dosya sistemini kopyalama	61
Bootloader kurulumu	61
Grub Kuralım	62
Grub yapılandırması	62
OpenRc Disk İşlemi	62
Fstab dosyası	62
Yardımcı Konular	63
Chroot Nedir?	63
Bağımlılık Scripti	64
Basit Sistem Oluşturma	64
Sistem Dizinini Oluşturulması	65
ls Komutu	65
mkdir Komutu	65
mkdir Komutu	66
bash Komutu	66
chroot Sistemde Çalışma	66
Qemu Kullanımı	67

Qemu Nedir?	67
Sisteme Kurulum	67
Sistem Hızlandırılması	67
Boot Menu Açma	67
Uefi kurulum için:	67
qemu Host Erişimi:	68
vmlinuz ve initrd	68
qemu Terminal Yönlendirmesi	68
Diskteki Sistemin Açılışını Terminale Yönlendirme	68
Live Sistem Oluşturma	69
SquashFS Nedir?	69
SquashFS Oluşturma	69
Cdrom Erişimi	69
Cdrom Bağlama	69
squashfs Dosyasını Bulma	69
squashfs Bağlama	69
squashfs Sistemine Geçiş	69
kmod Nedir?	71
Modul Yazma	71
hello.c dosyamız	71
Makefile dosyamız	72
modülün derlenmesi ve eklenip kaldırılması	72
Not:	72
sfdisk Nedir	73
Disk Bölümlerini Görüntüleme:	73
Disk Bölümleri Oluşturma:	73
Disk Bölümlerini Silme:	73
Disk Etiketi Belirleme	73
Bazı Örnekler	73
Örnek1:	73
Örnek2:	74
Örnek3:	74
Örnek4:	74
İmza Doğrulama	75
İmza Oluşturma	75
Belge İmzalama	75
İmzalı Belge Doğrulama	75
bash ile Doğrulama	75
c++ ile Doğrulama	77
c++ libpgpme ile Doğrulama	79
Kernel Modul Derleme	82
Kaynak Dosya İndirme	82
Kernel Ayarları	82

Kernel Derleme	82
Modul Derleme	82
Modül Yükleme	83
Strip Yapma	83
Kernel Yükleme	83
Header Yükleme	83
libc Yükleme	83
Terminal Yönlendirmesi	84
initrd Tasarımı	85
Sistem İçin Gerekli Olan Dosyalar Ve Açılış Süreci	85
initrd Nedir? Nasıl Hazırlanır?	85
Dizin Yapısının oluşturulması	86
S1- distro/initrd/bin/busybox	86
S2-S8 distro/initrd/bin/kmod	86
S9- distro/initrd/lib/modules/\$(uname -r)/moduller	87
S9- distro/initrd/bin/systemd-udevd	87
S10- distro/initrd/bin/udevadm	87
S12- distro/etc/udev/rules.d--S13- distro/lib/udev/rules.d	88
S14- distro/initrd/bin/init	88
S15- distro/iso/initrd.img - S16- distro/iso/vmlinuz	89
S17- distro/iso/grub/grub.cfg	89
İso Oluşturma	89
Bağımlılıkların Tespiti	89
strip	91
ld(linker)	92
Linker Türleri	92
cmake	93
sudoers	94
polkit	95
Tüm Uygulamalarda İzin Verme	95
Bir Gruba İzin Verme	95
Bir Grub-User-Uygulamaya İzin Verme	96
user-dirs	97
Ek Konular	98
openrc Servis Yönetimi	98
Servisi Başlangıca Ekleme	98
Servisi Başlangıçtan Kaldırma	98
Servislerin Çalışma Sırası	98
Basit Servis Komutlarını Çalıştırma	98
Sistem Dili Değiştirme	98
1. Yöntem	99
2. Yöntem	99
profile	99

3.Yöntem	99
Kullanıcı Sistem Ayarları	100
Profile Dosyası	100
** profile**	100
adduser ve useradd Kullanımı	100

Dağıtım

Temel Kavramlar

Temel Kavramlar

Linux Nedir?

Linux; Linux çekirdeğine dayalı, açık kaynak kodlu, Unix benzeri bir işletim sistemi ailesidir. GNU Genel Kamu Lisansı versiyon 2 ile sunulan ve Linux Vakfı çatısı altında geliştirilen bir özgür yazılım projesidir. Linux, "Linus Torvalds" tarafından geliştirilmeye başlanmıştır. Günümüzde bilgisayarlarda, akıllı cihazların ve internet altyapısında kullanılan cihazların işletim sistemlerinde yaygın olarak kullanılmaktadır.

Açık Kaynak Nedir?

Açık kaynak terimi, yazılım geliştirme sürecinde kaynak kodunun genel olarak erişilebilir ve değiştirilebilir olduğu bir modeli ifade eder. Bu yaklaşım, yazılımın geliştirilmesine katkıda bulunanların kodu incelemesine, değiştirmesine ve geliştirmesine olanak tanır. Açık kaynaklı yazılımlar genellikle topluluklar tarafından desteklenir ve geliştirilir, bu da geniş bir bilgi ve deneyim havuzundan faydalanmayı sağlar. Bu model, yazılımın sürekli olarak gelişmesine ve iyileştirilmesine olanak tanırken, kullanıcılar için de esneklik ve özgürlük sağlar.

Dağıtım Nedir ve Türleri

GNU, uygulamalar, kütüphaneler, geliştirici araçları, hatta oyunların bir arada olduğu Unix benzeri bir işletim sistemidir. Linux çekirdeği kullanılarak oluşturulan işletim sistemlerini tanımlamak için yaygın olarak Linux ismi kullanılmaktadır. Mesela, Linux çekirdeği ve GNU araçları bir araya getirilerek oluşturulan bir işletim sistemi "GNU/Linux dağıtımı" olarak adlandırılır, ancak konuşma dilinde kısaca (yanlış da olsa) Linux olarak ifade edilmektedir.

1. Clonezilla İmaj(tahta imajları)
2. Dağıtım Editleme(etahta)
3. Paket sistemi yazma ve paketleri yazma(ahenk,debian)

Ön Hazırlık

Dağıtım 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 xorriso mtools make squashfs-tools gcc wget unzip xz-utils tar zstd -y
```

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

\$HOME: Bu ifade linux ortamında açık olan kullanıcının ev dizinini ifade eder. Örneğin sisteme giriş yapan kullanıcı **ogrenci** adında bir kullanıcı olsun. **\$HOME** ifadesi **/home/ogrenci** konumunu ifade eder.

Minimal Dağıtım Oluşturma

Bu minimal dağıtım oluşturmamızın amacı bu dokümanda anlatılacak dağıtım hazırlama aşamalarını anlaşılması içindir. Dağıtım oluşturmak için kernel ve busybox dosyalarımızın olması yeterlidir. **busybox** dosyasının nasıl elde edileceği busybox bölümünde anlatılmıştır. **kernel** ise mevcut sistemimin kernelini kullanacağız. kernel **/boot/vmlinuz** konumundan alabiliriz.

Bu sistemi hazırlarken **chroot** ve **busybox** komutlarını kullanarak sistemi hazırlayacağız ve test edeceğiz. **chroot** ve **busybox** kullanımı için dokümandanki ilgili konu başlığına bakınız.

Sistemimiz için aşağıdaki gibi bir yapı oluşturmamız gerekir.

1. \$HOME/distro/rootfs/busybox
2. \$HOME/distro/rootfs/init
3. \$HOME/distro/iso/boot/initrd.img
4. \$HOME/distro/iso/boot/vmlinuz
5. \$HOME/distro/iso/boot/grub/grub.cfg
6. \$HOME/distro/distro.iso

Linux sisteminin açılabilmesi için 3., 4. ve 5. sıradaki gösterilen konumdaki dosyaların olması yeterli. Bu üç dosyayı yukarıda belirtildiği gibi dizin konumlarına koyduktan sonra, **grub-mkrescue \$HOME/distro/iso/ -o \$HOME/distro/distro.iso** komutuyla **distro.iso** dosyası elde ederiz. Artık 6. sırada belirtilen iso dosyamız boot edebilen şekilde hazırlanmış olur.

Sistemi oluşturan 3., 4. ve 5. sırada belirtilen dosyalarımız görevi şunlardır.

\$HOME/distro/iso/boot/initrd.img: Dosyasını sistemin açılış sürecinden ön işlemleri yapmak ve gerçek sisteme geçiş sürecini yöneten bir dosyadır. Yazın devamında nasıl hazırlanacağı anlatılacaktır.

\$HOME/distro/iso/boot/vmlinuz: Dosyamız kernelimiz oluyor. Ben kullandığım debian sisteminin mevcut kernelini kullandım. İstenirse kernel derlenebilir.

\$HOME/distro/iso/boot/grub/grub.cfg: Dosyamız ise initrd.img ve vmlinuz dosyalarının grub yazılımının nereden 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**

Yazının devamında sistem için gerekli olan 3 temel dosyanın hazırlanması ve iso yapılma süreci anlatılacaktır. Şimdi sırasıyla dosya konumlarına uygun şekilde dizin ve dosyalarımızı oluşturalım.

Dizinlerin Oluşturulması

Sitemimizi istediğimiz bir yerde terminal açarak aşağıdaki komutları sırasıyla çalıştıralım. Komutları kopyala yapıştır yapabilirsiniz.

```
mkdir $HOME/distro
mkdir $HOME/distro/rootfs
mkdir $HOME/distro/iso
mkdir $HOME/distro/iso/boot
mkdir $HOME/distro/iso/boot/grub
```

\$HOME/distro/rootfs/busybox

busybox aşağıdaki komutlarla **\$HOME/distro/rootfs/busybox** konumuna kopyalanak sistemimiz için hazır hale getirilir. Bu sistemi debian ortamında hazırlamaktayız. İsterseniz static busybox derleyebilirsiniz. busybox konusu altında anlatılmaktadır. isterseniz derlemeden mevcut sistemin altında bulunan static busybox kullanabiliriz. Eğer yoksa **sudo apt install busybox-static -y** komutuyla debian sistemimize static busybox yükleyebiliriz. Ben debian üzerine yüklediğim busybox'ı kullanacağım. Eğer sistemde yoksa **sudo apt install busybox-static -y** komutuyla yükleyiniz.

```
cd $HOME/distro/rootfs
cp /bin/busybox $HOME/distro/rootfs/busybox
ldd ./busybox
özdevimli bir çalıştırılabilir değil
```

"özdevimli bir çalıştırılabilir değil" dinamik değil diyor yani static kısacası bir bağımlılığı yok demektir. Eğer bağımlılığı olsaydı bağımlı olduğu dosyalarıda konumlarına göre kopyalamamız gerekmekeydi.

\$HOME/distro/rootfs/init

Sistem açılırken iki dosyayı arar kernel ve initrd.img dosyası. Bu dosyaları grub.cfg dosyası içinde verilen konumlarına göre arar. Bu dosyaları bulduktan sonra initrd.img dosyasını açar ve içinde init dosyasını arar. init dosyasındaki komutları sırasıyla çalıştırır.

\$HOME/distro/rootfs/ konumuna **init** text dosyası oluşturulur(nano, gedit vb.). Aşağıdaki içerik oluşturulan **init** dosyası içine eklenir.

```
#!/busybox ash
PATH=/bin
/busybox mkdir /bin
/busybox --install -s /bin
/busybox ash
```

busybox içerisinde linux ortamında kullanılan hemen hemen tüm komutların yerine kullanılabilir. Bulduğumuz ortamda **busybox** dışında hiçbir komutun olmadığını düşünün. Mesela **cp** komutuna ihtiyacınız var bu durumda **busybox cp** yazarak kullanılabilirsiniz. Tüm temel komutları busyboxtan türetmek için **/busybox --install -s /bin** komutunu kullanabilirsiniz.

Buradaki komutları sırayla anlatırsak;

1. **#!/busybox ash**: ash bir kabuk veya terminal programıdır.
2. **PATH=/bin**: komutlar çalışırken nerede arayacağı belirtiliyor
3. **/busybox mkdir /bin**: **busybox mkdir** yardımıyla **/bin** dizini oluşturuluyor.
4. **/busybox --install -s /bin**: **/bin** dizinine tüm komutları kısa yol oluşturur.
5. **/busybox ash**: ash terminalini çalıştırır. Bash yerine alternatif.

Ön Hazırlık

kernel, **initrd.img** dosyasını bellek üzerine açıp **init** dosyasını çalıştırınca sırasıyla numaralandırılarak anlattığımız işlemler yapılacak ve çalışabilecek ortam hazırlanacaktır. Daha sonra ash terminali çalışarak kullanıcının kullanımına hazır hale gelecektir.

init çalıştırılabilir yapılır.

```
chmod +x init #komutu ile çalıştırılır yapılır.
```

\$HOME/distro/iso/boot/initrd.img

initrd.img dosyası için aşağıdaki komutlar çalıştırılır

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

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

\$HOME/distro/iso/boot/vmlinuz

```
cp /boot/vmlinuz* $HOME/distro/iso/boot/vmlinuz #sistemde kullandığım kerneli kopyaladım istenirde kernel derlenebilir.
```

\$HOME/distro/iso/boot/grub/grub.cfg

\$HOME/distro/iso/boot/grub/ konumuna **grub.cfg** dosyası oluşturun. Aşağıdaki komutları **grub.cfg** dosyası içine eklenir.

```
linux /boot/vmlinuz  
initrd /boot/initrd.img  
boot
```

\$HOME/distro/distro.iso

iso oluşturulur.

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

Artık sistemi açabilen ve tty açıp bize suna bir yapı oluşturduk.

Dağıtımın Test Edilmesi

Hazırlanan **\$HOME/distro/distro.iso** dağıtımımız qemu veya virtualbox ile test edilebilir.

Aşağıdaki komutla çalıştırıp test edebiliriz.

```
qemu-system-x86_64 -cdrom $HOME/distro/distro.iso -m 1G
```

Eğer hatasız yapılmışsa sistem açılacak ve tty açacaktır. Birçok komutu çalışan bir dağıtım oluşturmuş olduk.

Dokümanın devamında kendi sistemimizi hazırlarken bu bölümde anlatılan yapıya benzer adımları yapacağız. Minimal sistemden farkı **busybox** ile oluşturduğumuz ikili dosyalarımızı(temel komutlar) kendimiz derleyeceğiz. Minimal sistemde static busybox kullandık.

Static dosyalarda dosyanın çalışması için kütüphanelerin hepsi kendi içerisine gömülmüştür(dahil) bir şekilde gelir. Avantajı hiçbir kütüphaneye ihtiyaç duymaz. Deventajı ise boyutları yüksek

Ön Hazırlık

olur. İstisnalar olsada genel olarak static tercih edilmemektedir. Static olduğunda iso boyutlarımız olması gerekenden 5-10 kat fazla olabilmektedir.

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.

```
$ busybox [komut] (seçenekler)
```

Eğer busyboxu komut adı ile linklersek o komutu doğrudan çalıştırabiliriz. Aşağıda tar uygulamasını busybox dan türettik.

```
$ ln -s /bin/busybox ./tar
$ ./tar
```

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 Derleme

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. Derleme türleri ve detayları için bu dokümandaki derleme konusuna bakınız.

```
$ 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 yapacaksak 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
```

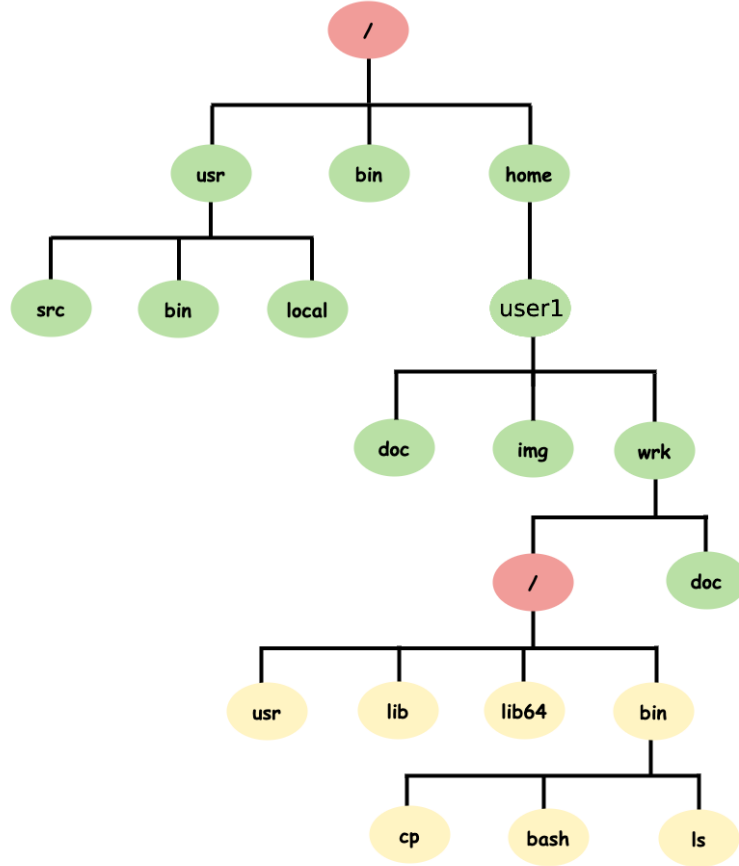
Derleme bittiğinde kaynak kodun bulunduğu dizinde busybox dosyamız oluşmuş olur.

Static olarak derlemiş olduğumuz busyboxu kullanarak minimal bir dağıtım hazırlayabiliriz.

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 bağımlıkları 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
```

Ön Hazırlık

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

Iddscript.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
```

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.

Ön Hazırlık

Sistem Dizinini 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/libpcre2-8.so.0' -> '/home/etapadmin/test/usr/lib/x86_64-linux-gnu/libpcre2-8.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:~$
```

Ön Hazırlık

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 initrd.img oluşturabilen ve grubu yükleyecek bir yapıda olmasını planlamaktayız. Bu yapıda bir dağıtım için aşağıdaki paketlere ihtiyacımız olacak. Bunlar;

- glibc
- readline
- ncurses
- bash
- kmod
- busybox
- acl
- attr
- eudev
- coreutils
- util-linux
- audit
- base-file
- brotli
- dosfstools
- e2fsprogs
- efibootmgr
- efivar
- expat
- file
- findutils
- gawk
- gmp
- grep
- grub
- gzip
- initramfs-tools
- libc6-dev
- libcap
- libmd
- libpcr2
- libstdc++-dev

Temel Paketleri Derleme

- libxml2
- linux-headers
- linux-image
- linux-libc-dev
- live-boot
- live-config
- openrc
- openssl
- p7zip
- pam
- parted
- sed
- xz-utils-debian
- zlib
- zstd

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 ve bağımlılık sadece **bash** ve **kmod** paketleri özelinde anlatılmaya çalışıldı. Diğer paketler içinde bağımlı olduğu paketler olacaktır.

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 kernel 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 kernel 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.

zlib Derleme

```
version="1.3"
name="zlib"

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

wget https://zlib.net/current/${name}.tar.gz
tar -xvf ${name}.tar.gz

mkdir build-${name}-${version}

cd build-${name}-${version}

../${name}-${version}/configure --prefix=/

make

make install DESTDIR=$HOME/rootfs
```

initramfs-tools Nedir

initramfs-tools, Debian tabanlı sistemlerde kullanılan bir araçtır ve initramfs (initial RAM file system) oluşturmak için kullanılır. Bu araç, sistem açılırken kullanılan geçici bir dosya sistemini oluşturur ve gerekli modülleri yükler. initramfs için farklı araçlarda kullanılabilir. Kullanıcı isterse kendi scriptinide kullanabilir. Debian dışında **dracut** aracıda initramfs oluşturmak ve güncellemek için kullanılabilir.

/etc/initramfs-tools/modules

modules dosyası initrd oluşturulma ve güncelleme durumunda isteğe bağlı olarak modüllerin eklenmesini ve **initrd** açıldığında modülün yüklenmesini istiyorsak **/etc/initramfs-tools/modules** komundaki dosyayı aşağıdaki gibi düzenlemeliyiz. Bu dosya içinde **ext4**, **vfat** ve diğer yardımcı modüller eklenmiş durumdadır.

```
### This file is the template for /etc/initramfs-tools/modules.
### It is not a configuration file itself.
###
# List of modules that you want to include in your initramfs.
# They will be loaded at boot time in the order below.
#
# Syntax:  module_name [args ...]
#
# You must run update-initramfs(8) to effect this change.
#
# Examples:
#
# raid1
# sd_mod
vfat
fat
nls_cp437
nls_ascii
nls_utf8
ext4
```

initramfs-tools Ayarları

/usr/share/initramfs-tools/hooks/ konumundaki dosyaları dikkatlice düzenlemek gerekmektedir. Dosyaları alfabetik sırayla çalıştırdığı için **busybox zzz-busybox** şeklinde ayarlanmıştır.

initramfs-tools Güncelleme

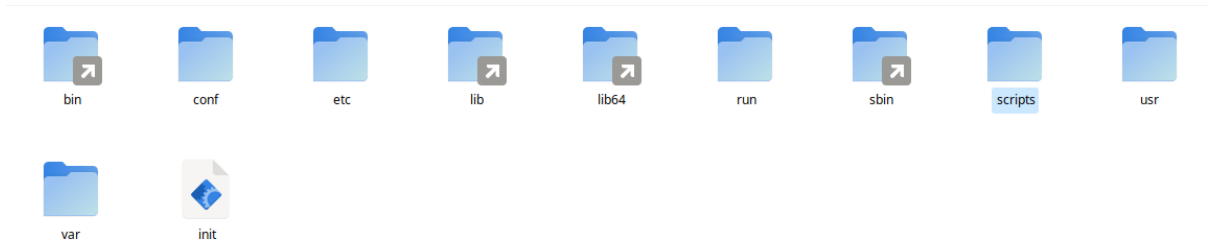
```
/usr/sbin/update-initramfs -u -k $(uname -r) #initrd günceller
```

Güncelleme ve oluşturma aşamasında **/usr/share/initramfs-tools/hooks/** konumundaki dosyayı çalıştırarak yeni initrd dosyasını oluşturacaktır. Oluşturma **/var/tmp** olacaktır. Ayrıca **/boot/config-6.6.0-amd64** gibi sistemde kullanılan kernel versiyonuyla config dosyası olmalıdır. Burada verilen **6.6.0-amd64** örnek amaçlı verilmiştir.

initrd açılma Süreci

Sistemin açılması için **vmlinuz**, **initrd.img** ve **grub.cfg** dosyalarının olması yeterlidir. **initrd.img** sistemin açılma sürecini yürüten bir kernel yardımcı ön sistemidir. **initrd.img** açıldığında aşağıdaki gibi bir dizin yapısı olur. Bu dizinler içindeki **script** dizini çok önemlidir. Bu dizin içindeki scriptler belirli bir sırayla çalışarak sistemin açılması sağlanır.

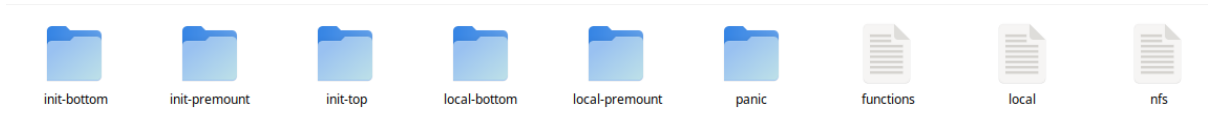
Temel Paketleri Derleme



initrd script içeriği

script içindeki dizinler aşağıdaki gibidir. Bu dizinler içinde scriptler vardır. Bu dizinlerin içeriği sırayla şöyle çalışmaktadır.

1. init-top
2. init-premount
3. init-bottom



Oluşan initrd.img dosyası sistemin açılmasını sağlayamıyorsa script açılış sürecini takip ederek sorunları çözebilirsiniz.

Temel Paketleri Derleme

OpenRC

Openrc sistem açılışında çalışacak uygulamaları çalıştıran servis yöneticisidir.

Derleme

Kaynak koddan derlemek için aşağıdaki adımları izlemelisiniz:

```
git clone https://github.com/OpenRC/openrc
cd openrc
meson setup build \
--sysconfdir=/etc \
--libdir=/lib \
--prefix=/ \
-Ddefault_library=both \
-Dzsh-completions=true \
-Dbash-completions=true \
-Dpkgconfig=true

meson setup build --prefix=/usr
export DESTDIR=${DESTDIR}
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.
```

Paket Sistemi Tasarlama

Paket Sitemi

Paket yönetim sistemleri bir dağıtımda bulunan en temel parçadır. Sistem üzerine paket kurma ve kaldırma güncelleme yapma gibi işlemlerden sorumludur. Başlıca 2 tip paket sistemi vardır:

- Binary (ikili) paket sistemi
- Source (kaynak) paket sistemi

Bir paket sistemi hem binary hem source paket sistemi özelliklerine sahip olabilir. Bununla birlikte son kullanıcı dağıtımlarında genellikle binary paket sistemleri tercih edilir.

Binary Paket Sistemi

Bu tip paket sistemlerinde önceden derlenmiş olan paketler hazır şekilde indirilir ve açılarak sistem ile birleştirilir. Binary paket sistemlerinde paketler önceden derleme talimatları ile oluşturulmalıdır.

Binary paket sistemine örnek olarak **apt**, **dnf**, **pacman** örnek verilebilir.

Source Paket Sistemi

Bu tip paket sistemlerinde derleme talimatları kurulum yapılacak bilgisayar üzerinde kullanarak paketler kurulum yapılacak bilgisayarda oluşturulur ve kurulur.

Source paket sistemine örnek olarak **portage** örnek verilebilir.

Paket sisteminin temel yapısı

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

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

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

Bazı dağıtımların kullandığı paket sistemleri 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 anlaması amaçlandı.

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 yapılıyor.

```
# 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, "source bpsbuild" komutu sonucu gelen fonksiyon
build() {} #bpsbuild dosyasından gelen fonksiyon, "source bpsbuild" komutu sonucu gelen fonksiyon
package() {} #bpsbuild dosyasından gelen fonksiyon, "source bpsbuild" komutu sonucu 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}
}

#paketlenen 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 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.

Depo indexleme

Depo, paket yönetim sistemlerinde kurulacak olan paketleri içeren bir veri topluluğudur. Kaynak depo ve ikili depo olarak ikiye ayrılır. Depo içerisinde hiyerarşik olarak paketler yer alır. Index ise depoda yer alan paketlerin isimleri sürüm numaraları gibi bilgiler ile adreslerini tutan kayıttır. Paket yönetim sistemi index içerisinden gelen veriye göre gerekli paketi indirir ve kurar. Depo indexi aşağıdaki gibi olabilir:

```
Package: hello
Version: 1.0
Dependencies: test, foo, bar
Path: h/hello/hello_1.0_x86_64.zip

Package: test
Version: 1.1
Path: t/test/test_1.1_aarch64.zip

...
```

Yukarıdaki örnekte paket adı bilgisi sürüm bilgisi ve bağımlılıklar gibi bilgiler ile paketin sunucu içerisindeki konumu yer almaktadır. Depo indexi paketlerin içinde yer alan paket bilgileri okunarak otomatik olarak oluşturulur.

Örneğin paketlerimiz zip dosyası olsun ve paket bilgisini **.INFO** dosyası taşıyın. Aşağıdaki gibi depo indexi alabiliriz.

```
function index {
    > index.txt
    for i in $@ ; do
        unzip -p $i .INFO >> index.txt
        echo "Path: $i" >> index.txt
    done
}
index t/test/test_1.0_x86_64.zip h/hello/hello_1.1_aarch64.zip ...
```

Bu örnekte paketlerin içindeki paket bilgisi içeren dosyaları uç uca ekledik. Buna ek olarak paketin nerede olduğunu anlamak için paket konumunu da ekledik.

bps Paket Liste Indexi 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 repository üzerinden çekilmektedir. Paket listemiz ise yapılan her yeni paketi yükleme sırasında güncellenmektedir.

Paket güncelleme için iki script kullanılmaktadır. 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 içerisinden **bpsbuild** dosyalarını çıkartarak pakete ilgili bilgileri alıp **index.lst** dosyası oluşturmaktadır. istersek 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
curl -O /tmp/index.lst https://basitsadigitim.github.io/binary-package/index.lst
```

index.lst dosyamızı github üzerinden indiren scriptimiz tek bir satırdan oluşmaktadır. Bu komut <https://basitsadigitim.github.io/binary-package/index.lst> adresindeki dosyayı index.lst dosyasını /tmp/index.lst konumuna indirecektir.

Paket Kurma

Paket kurulurken paket içerisinde bulunan dosyalar sisteme kopyalanır. Daha sonra istenirse silinebilmesi için paket içeriğinde dosyaların listesi tutulur. Bu dosya ayrıca paketin bütünlüğünü kontrol etmek için de kullanılır.

Örneğin bir paketimiz zip dosyası olsun ve içinde dosya listesini tutan **.LIST** adında bir dosyamız olsun. Paketi aşağıdaki gibi kurabiliriz.

```
cd /onbellek/dizini
unzip /dosya/yolu/paket.zip
cp -rfp ./* /
cp .LIST /paket/veri/yolu/paket.LIST
```

Bu örnekte ilk satırda geçici dizine gittik ve paketi oraya açtık. Daha sonra paket içeriğini kök dizine kopyaladık. Daha sonra paket dosya listesini verilerin tutulduğu yere kopyaladık. Bu işlemden sonra paket kurulmuş oldu.

bps Paket Kurma Scripti Tasarlama

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.

Paket Sistemi Tasarlama

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 $indexpaket|cut -d":" -f2)
    dependsx=$(echo $indexpaket|cut -d":" -f3)
    name=${namex:6:-1}
    version=${versionx:9:-1}
    depends=${dependsx: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=\"${depends}\"">>var/bps/index.lst
#paket içinde gelen paket dosyalarının dosya ve dizin yapısını tutan file index dosyasının /var/bps/ konumuna kopyalanması
cp file.lst /var/bps/${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 paket.LIST dosyası içerisindeki dosyalar, dizinler kaldırılır.
3. Kaldırılan dosyalardan sonra /paket/veri/yolu/paket.LIST dosyasından paket bilgisi kaldırılır.
4. sistemde kurulu paketler index dosyasından ilgili paket satırı kaldırılmalıdır.

Paketi kaldırmak için ise aşağıdaki örnek kullanılabilir.

```
cat /paket/veri/yolu/paket.LIST | while read dosya ; do
    if [[ -f "$dosya" ]] ; then
        rm -f "$dosya"
    fi
done
cat /paket/veri/yolu/paket.LIST | while read dizin ; do
    if [[ -d "$dizin" ]] ; then
        rmdir "$dizin" || true
    fi
done
rm -f /paket/veri/yolu/paket.LIST
```

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.

bps Paket Kaldırma Scripti Tasarlama

Dokumanda örnek olarak verilen bps paket sistemi için yukarıdaki paket kaldırma bilgilerini kullanarak tasarlanmıştır.

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 dizin ; 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
```


Paket Sistemi Tasarlama

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 izinleri 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. Bu bölümü uygulamadan önce uygulam adımlarını daha anlayabilmek için mutlaka initr tasalama konusunu okumalısınız. initrd Tasarımı

Temel Dosyalar

Linux sisteminin açılabilmesi için aşağıdaki 3 dosya yeterlidir. Bu dosyalar yardımıyla sistem açılışı yapılır ve diskimizde bulunan sistemi bu 3 dosya yardımıyla çalıştırır ve hazır hale getiririz. Şimdi sırasıyla 3 dosyamızı nasıl hazırlayacağımızı adım adım uygulayalım. Dosyaları oluşturduktan sonra iso haline getirerek sistemi çalışır hale getirelim.

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

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.

linux Açılış Süreci

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 yapılma 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)/moduller #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

#*****modul yukleme*****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 #modul 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.

```
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 doyamız oluşturulur.
```

Bu komut çalışınca **distro.iso** dosyası elde ederiz. Artık iso dosyamız boot edebilen hazırlanmış bir dosyadır.

iso Hazırlama

İso 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/iso/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/iso/live/filesystem.squashfs** konuma kopyalanmalıdır. Aşağıdaki komutlarla **filesystem.squashfs** hazırlanıyor ve **\$HOME/distro/iso/live/** konumuna taşınıyor.

```
cd $HOME/distro/  
mksquashfs $HOME/distro/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

Sistem Kurma

Hazırlanmış bir iso ile çeşitli kurulum araçları gelebilir. Bu araçların farklı kurulum yapma yöntemleri olmaktadır. Sık kullanılan yöntemler şunlardır;

1. Tek bölüme sistem kurma
2. iki bölüme sistem kurma(boot+sistem)
3. uefi sistem kurma(boot+sistem)

Burada üç farklı kurulum yöntemi sırayla anlatılacaktır.

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.

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
```

Şimdi de disk bölümümüzü bağlayalım.

Sistem Kurulumu

```
$ 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.

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	

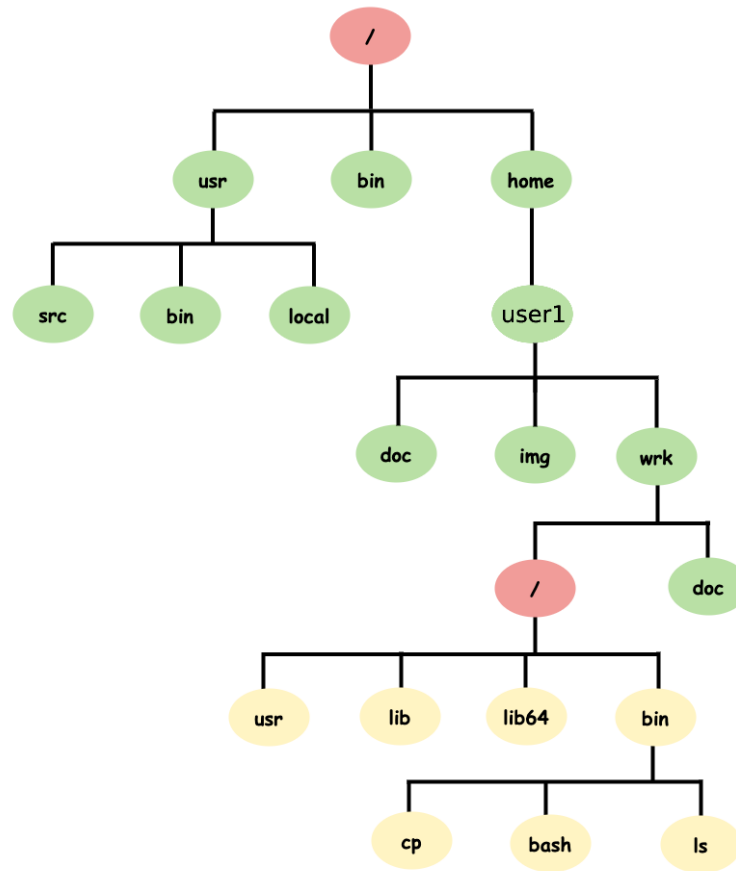
Not: Disk bölümü konumu yerine **UUID=<uuid-değeri>** şeklinde yazmanızı öneririm. Bölüm adları değişebilirken uuid değerleri değişmez.

Yardımcı Konular

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
```

Yardımcı Konular

Yeni sistem tasarlamak ve erişmek için temel komutları ve komut yorumlayıcının 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

Iddscript.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
```

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.

Yardımcı Konular

Sistem Dizinin 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/libpcre2-8.so.0' -> '/home/etapadmin/test/usr/lib/x86_64-linux-gnu/libpcre2-8.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:~$
```

Yardımcı Konular

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>

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
```

Uefi kurulum için:

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

Yardımcı Konular

```
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.

vmlinuz ve initrd

Daha sonra da qemu kullanarak test edelim.

```
qemu-system-x86_64 --enable-kvm -kernel /boot/vmlinuz-5.17 -initrd /home/deneme/initrd.img -append "quiet" -m 512m
```

qemu Terminal Yönlendirmesi

```
qemu-system-x86_64 --enable-kvm -kernel vmlinuz -initrd initrd.img -m 3G -serial stdio -append "console=ttyS0"
```

Diskteki Sistemin Açılışını Terminale Yönlendirme

```
qemu-system-x86_64 -nographic -kernel boot/vmlinuz -hda disk.img -append console=ttyS0
```

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
```

Yardımcı Konular

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.

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 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.

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);
```

Yardımcı Konular

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:

Disk Etiketini Belirleme

```
echo 'label: dos' | sfdisk /dev/vda #dos label  
echo 'label: gpt' | sfdisk /dev/vda #gpt label
```

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
```

Örnek3:

Bu örnekte ilk bölüm 1GB ve ikinci bölüm ise diskin geri kalan kısmıdır.

```
echo -e "label: gpt\n,1GiB\n," | 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 GPG 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.

```
#!/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"
```

Yardımcı Konular

fi

```
belge.txt.asc (~/.Masaüstü/gpg) - gedit
1 ----BEGIN PGP SIGNED MESSAGE-----
2 Hash: SHA512
3
4 hava güneşli
5 ----BEGIN PGP SIGNATURE-----
6
7 iLMEAEKABOWIOU+W8kMxqIL6zWqt8c2OLzb0WrwUCZwt4YQAKRB8c
8 r2amBACBumERXzB1B19EgZMJR4+Mbevk97qk3FRKVbp8T/
9 zml187IxuGdNENdVUK
9 snJftSRVzgsC/OQ2tnhGM3be2gGqM4aFI4J0m41G3JV+L+An/
DmTwk6q8t8e/L
10 oGgiED9KCJ/Q5/DqE2OM0ePfwCJpet5SY0KUL5PwYnZBRPnABA==
11 =T817
12 -----END PGP SIGNATURE-----

imzadogrula.sh (~/.Masaüstü/gpg) - gedit
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 #!
```

```
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 14F96F24331C6A88BE835AAB7C73638BCDB196A
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 BE83 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
1 ----BEGIN PGP SIGNED MESSAGE-----
2 Hash: SHA512
3
4 hava güneşli
5 ----BEGIN PGP SIGNATURE-----
6
7 iLMEAEKABOWIOU+W8kMxqIL6zWqt8c2OLzb0WrwUCZwt4YQAKRB8c
8 r2amBACBumERXzB1B19EgZMJR4+Mbevk97qk3FRKVbp8T/
9 zml187IxuGdNENdVUK
9 snJftSRVzgsC/OQ2tnhGM3be2gGqM4aFI4J0m41G3JV+L+An/
DmTwk6q8t8e/L
10 oGgiED9KCJ/Q5/DqE2OM0ePfwCJpet5SY0KUL5PwYnZBRPnABA==
11 =T817
12 -----END PGP SIGNATURE-----

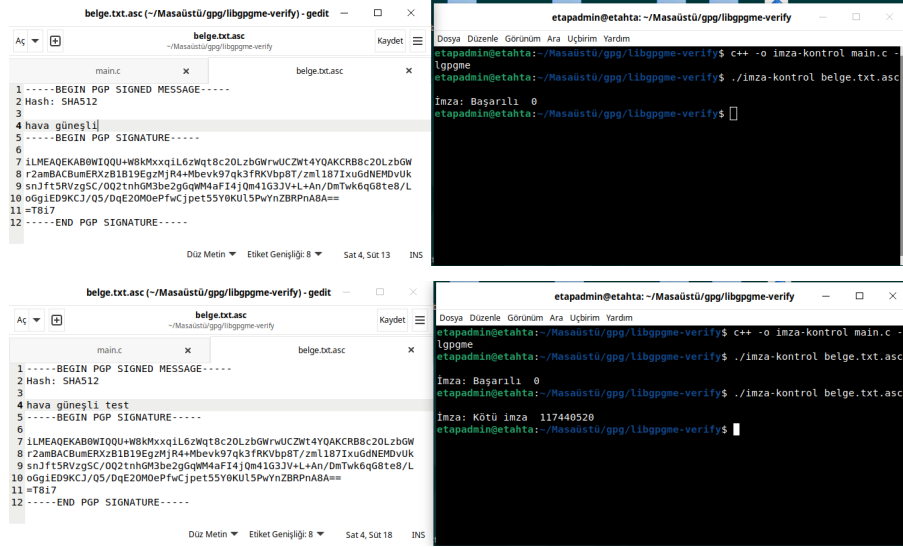
imzadogrula.sh (~/.Masaüstü/gpg) - gedit
1 #!/bin/bash
2 file=$1
3 if [ $file == "" ]
4 then
5 echo "Dosya belirtiniz."
6 else
7 gpg --verify "$file"
8 status=$?
9 if [ "$status" == "0" ]
10 then
11 echo "İmza İyi"
12 else
13 echo "İmza Kötü"
14 fi
15 #!
```

```
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 14F96F24331C6A88BE835AAB7C73638BCDB196A
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 BE83 5AAB 7C73 638B CDB1 96AF
gpg: WARNING: not a detached signature; file 'belge.txt' was NOT verified!
İmza İyi
etapadmin@etahta:~/.Masaüstü/gpg$
```

Yardımcı Konular

c++ ile Doğrulama

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



```
belge.txt.asc (~/.Masaüstü/gpg/libpgpgme-verify) - gedit
main.c x belge.txt.asc x
1 -----BEGIN PGP SIGNED MESSAGE-----
2 Hash: SHA512
3
4 hava güneşli
5 -----BEGIN PGP SIGNATURE-----
6
7 lLMEAEKABOWIQU+WBkMxxq1L6zWqt8c20LzbGwruJCZt4YQAKRBBc20LzbGw
8 r2ambACumERXzB1B19EgZMjR4+Mbevk97qk3fRKVbp8T/zml187IuGdNEMDVUK
9 snJft5SRVzG5C/0QZtnhG3be2GqW4aF14jQm41G3JV+L+An/DmTwk6qG8te8/L
10 oGg1ED9KCj/Q5/DqE2OM0ePfwCjpet55Y8KUL5PwYnZBRPhA8A==
11 =T817
12 -----END PGP SIGNATURE-----
Düz Metin Etiket Genişliği: 8 Sat 4, Süt 13 INS

etapadmin@etahta: ~/.Masaüstü/gpg/libpgpgme-verify
Dosya Düzenle Görünüm Ara Uçbirim Yardım
etapadmin@etahta: ~/.Masaüstü/gpg/libpgpgme-verify$ c++ -o imza-kontrol main.c -lgpgme
etapadmin@etahta: ~/.Masaüstü/gpg/libpgpgme-verify$ ./imza-kontrol belge.txt.asc
Imza: Başarılı 0
etapadmin@etahta: ~/.Masaüstü/gpg/libpgpgme-verify$

belge.txt.asc (~/.Masaüstü/gpg/libpgpgme-verify) - gedit
main.c x belge.txt.asc x
1 -----BEGIN PGP SIGNED MESSAGE-----
2 Hash: SHA512
3
4 hava güneşli test
5 -----BEGIN PGP SIGNATURE-----
6
7 lLMEAEKABOWIQU+WBkMxxq1L6zWqt8c20LzbGwruJCZt4YQAKRBBc20LzbGw
8 r2ambACumERXzB1B19EgZMjR4+Mbevk97qk3fRKVbp8T/zml187IuGdNEMDVUK
9 snJft5SRVzG5C/0QZtnhG3be2GqW4aF14jQm41G3JV+L+An/DmTwk6qG8te8/L
10 oGg1ED9KCj/Q5/DqE2OM0ePfwCjpet55Y8KUL5PwYnZBRPhA8A==
11 =T817
12 -----END PGP SIGNATURE-----
Düz Metin Etiket Genişliği: 8 Sat 4, Süt 18 INS

etapadmin@etahta: ~/.Masaüstü/gpg/libpgpgme-verify
Dosya Düzenle Görünüm Ara Uçbirim Yardım
etapadmin@etahta: ~/.Masaüstü/gpg/libpgpgme-verify$ c++ -o imza-kontrol main.c -lgpgme
etapadmin@etahta: ~/.Masaüstü/gpg/libpgpgme-verify$ ./imza-kontrol belge.txt.asc
Imza: Başarılı 0
etapadmin@etahta: ~/.Masaüstü/gpg/libpgpgme-verify$ ./imza-kontrol belge.txt.asc
Imza: Kötü imza 117440520
etapadmin@etahta: ~/.Masaüstü/gpg/libpgpgme-verify$
```

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

```
int main() {
```

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

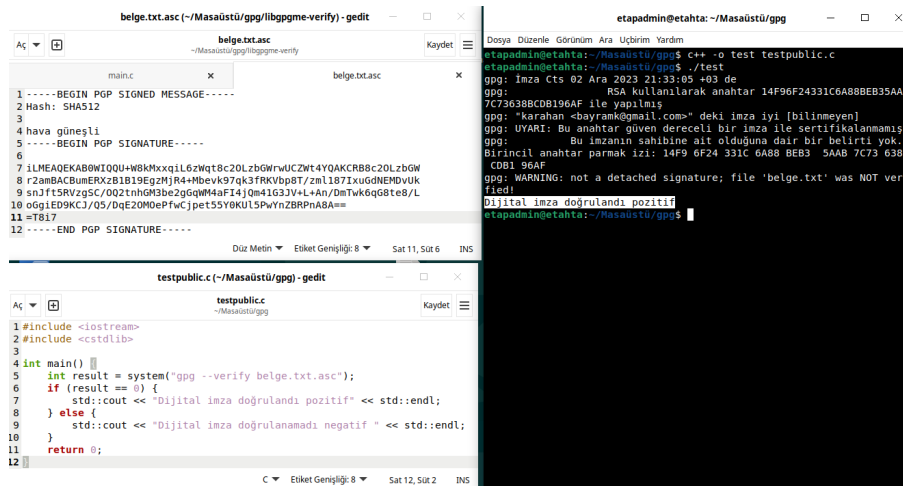
```
        std::cout << "Dijital imza doğrulandı pozitif" << std::endl;
```

```
    } else {
```

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

```
    } return 0;
```

```
}
```

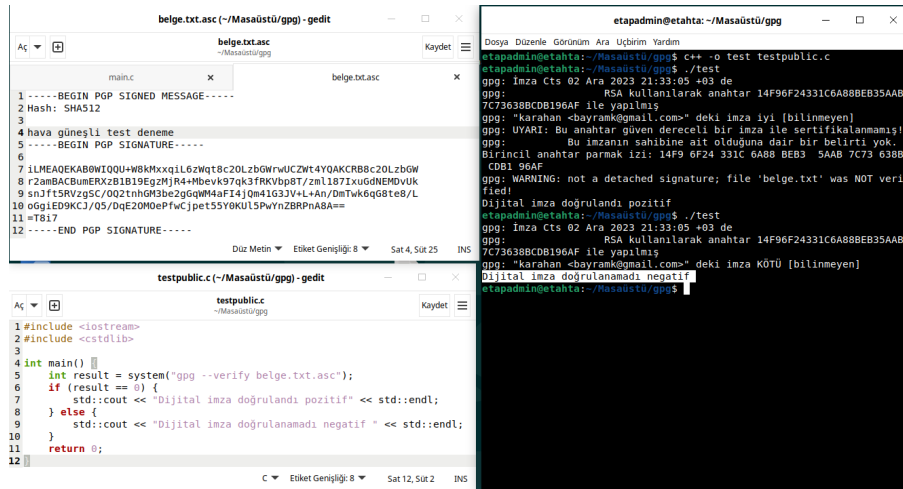


```
belge.txt.asc (~/.Masaüstü/gpg/libpgpgme-verify) - gedit
main.c x belge.txt.asc x
1 -----BEGIN PGP SIGNED MESSAGE-----
2 Hash: SHA512
3
4 hava güneşli
5 -----BEGIN PGP SIGNATURE-----
6
7 lLMEAEKABOWIQU+WBkMxxq1L6zWqt8c20LzbGwruJCZt4YQAKRBBc20LzbGw
8 r2ambACumERXzB1B19EgZMjR4+Mbevk97qk3fRKVbp8T/zml187IuGdNEMDVUK
9 snJft5SRVzG5C/0QZtnhG3be2GqW4aF14jQm41G3JV+L+An/DmTwk6qG8te8/L
10 oGg1ED9KCj/Q5/DqE2OM0ePfwCjpet55Y8KUL5PwYnZBRPhA8A==
11 =T817
12 -----END PGP SIGNATURE-----
Düz Metin Etiket Genişliği: 8 Sat 11, Süt 6 INS

testpublic.c (~/.Masaüstü/gpg) - gedit
main.c x testpublic.c x
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 }
C Etiket Genişliği: 8 Sat 12, Süt 2 INS

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: 7C72638BCD8196AF ile yapılmış
gpg: "karahan <bayram@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.
gpg: Birincil anahtar parmak izi: 14F9 6F24 331C 6A88 BEB3 5AAB 7C73 638B
gpg: CDB1 96AF
gpg: WARNING: not a detached signature; file 'belge.txt' was NOT veri
fied!
Dijital imza doğrulandı pozitif
etapadmin@etahta: ~/.Masaüstü/gpg$
```

Yardımcı Konular



```
belge.txt.asc (~/.Masaüstü/gpg) - gedit
Aç [ ] belge.txt.asc
main.c x belge.txt.asc x
1 -----BEGIN PGP SIGNED MESSAGE-----
2 Hash: SHA512
3
4 hava güneşli test deneme
5 -----BEGIN PGP SIGNATURE-----
6
7 iLMEAOEKAB0W1QOU+H8kMxqiL6zWqT8c2OLzbGWwUCZwt4Y0AKCRB8c2OLzbGW
8 r2anBACBumERXz81B19EgZMjR4+HbeV97q3fRKVbp8T/zm1187IXuGdNEMDVUk
9 snJftSRVz9Sc/QQ2tnhgM3be2G6qW4aFI4j0m41G3JV+L+An/DmTwk6qG8te8/L
10 oGgiED9KCJ/05/DqE20M0ePfwCjpet5SY0KUL5PwYnZBRPnA8A==
11 =T817
12 -----END PGP SIGNATURE-----
Düz Metin Etiket Geniği: 8 Sat 4, Süt 25 INS

testpublic.c (~/.Masaüstü/gpg) - gedit
Aç [ ] testpublic.c
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 }
C Etiket Geniği: 8 Sat 12, Süt 2 INS

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 <gpgme.h>
#include <locale.h>
#include <stdlib.h>
#include <string.h>

int print_engine_info() {
    gpgme_engine_info_t info;
    gpgme_error_t err;

    err = gpgme_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, gpgme_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 *gpgme_version, *gpgme_prot;
    gpgme_error_t err;
    gpgme_ctx_t ctx;
    FILE *fp_sig=NULL, *fp_msg=NULL;
    gpgme_data_t sig=NULL, msg=NULL, plain=NULL, text=NULL;
    gpgme_verify_result_t result;

    gpgme_protocol_t protocol = GPGME_PROTOCOL_OpenPGP;

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

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

```
/* Protocol check */
gpgme_prot = gpgme_get_protocol_name(protocol);
err = gpgme_engine_check_version(protocol);
if (!gpgme_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 oluşturmamız gerekir. Bu dosyada hangi ayarlara göre derleme yapılacağını belirten parametreler var. Eğer temel(varsayılan) ayarlarda derlenmesini istersek **make defconfig** komutuyla **.config** adında bir dosya oluşturacaktır.

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 diyebilirsiniz. Burada Arch Linux **.config** dosyasını kullanacağız.

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 yapılacaktır. Derleme aşağıdaki komutla yapılır.

```
make bzImage # tek çekirdekle derleme yapacak yavaş olur
#make bzImage -j$(nproc) #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 uygun modüllerin derlenmesi gerekmektedir.

Modul Derleme

Modül ise kernel ile donanım arasında iletişim kuran yazılımlardır. Modüller kernel versiyonuyla aynı versiyonda olmalıdır. Başka versiyon kernel'leri 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$(nproc) #işlemci çekirdek sayısını sistemden alarak yapıyor, hızlı olur
```

Yardımcı Konular

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
```

Strip Yapma

Kernel Yükleme

Header Yükleme

libc Yükleme

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.

initrd Tasarımı

Sistem İçin Gerekli Olan Dosyalar Ve Açılış Süreci

Linux sisteminin açılabilmesi için aşağıdaki 3 dosya yeterli.

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

distro/iso/boot/initrd.img dosyasını sistemin açılış sürecinden ön işlemleri yapmak ve 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 yazılımının nereden 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- **vmlinuz initrd.img sistemini belleğe yüklüyor**
- 6- **initrd.img içindeki init dosyasındaki işlem sürecine göre sistem işlemlere devam ediyor**
- 7- **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 hazırlanması ve iso yapılıma süreci anlatılacaktır.

initrd Nedir? Nasıl Hazırlanır?

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.

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- distro/initrd/bin/busybox          #dosya
S2- distro/initrd/bin/kmod             #dosya
S3- distro/initrd/bin/debmod           #dosya
```

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

Dizin Yapısının oluşturulması

Aşağıdaki komutları çalıştırdığımızda izin yapımız oluşacaktır.

```
mkdir -p initrd
mkdir -p initrd/bin/
mkdir -p initrd/lib/
ln -s lib initrd/lib64
mkdir -p initrd/lib/modules/
mkdir -p initrd/lib/modules/${uname -r}
mkdir -p initrd/lib/modules/${uname -r}/moduller
mkdir -p initrd/etc/udev/
mkdir -p initrd/lib/udev/
mkdir -p iso
mkdir -p iso/boot
mkdir -p iso/boot/grub
```

S1- distro/initrd/bin/busybox

busybox hakkında bilgi almak için busybox yazısında anlatılmıştır. Burada sisteme nasıl ekleneceği anlatılacaktır. busybox dosyamızın bağımlılıklarının **lddscript.sh** scripti ile initrd içine kopyalayacağız. Yazının devamında **Bağımlılık Tespiti** konu başlığı altında anlatılmıştır.

```
cp /usr/bin/busybox initrd/bin/busybox #sistemden busybox kopyalandı..
lddscript.sh initrd/bin/busybox initrd/ #sistemden busybox bağımlılıkları initrd dizinimize kopyalar.
```

S2-S8 distro/initrd/bin/kmod

kmod yazısında kmod anlatılmıştır. Burada sisteme nasıl ekleneceği anlatılacaktır.

```
cp /usr/bin/kmod initrd/bin/kmod #sistemden kmod kopyalandı..
lddscript.sh initrd/bin/kmod initrd/ #sistemden kmod kütüphaneleri kopyalandı..
ln -s kmod initrd/bin/depmod      #kmod sembolik link yapılarak depmod hazırlandı.
ln -s kmod initrd/bin/insmod      #kmod sembolik link yapılarak insmod hazırlandı.
ln -s kmod initrd/bin/lsmmod      #kmod sembolik link yapılarak lsmod hazırlandı.
ln -s kmod initrd/bin/modinfo     #kmod sembolik link yapılarak modinfo hazırlandı.
ln -s kmod initrd/bin/modprobe    #kmod sembolik link yapılarak modprobe hazırlandı.
ln -s kmod initrd/bin/rmmod       #kmod sembolik link yapılarak rmmod hazırlandı.
```


S9- distro/initrd/lib/modules/\$(uname -r)/moduller

Bu bölümde modüller hazırlanacak. Burada dikkat etmemiz gereken önemli bir nokta kullandığımız kernel versiyonu neyse **initrd/lib/modules/modules** altında oluşacak dizinimiz aynı olmalıdır. Bundan dolayı **initrd/lib/modules/\$(uname -r)** şeklinde dizin oluşturulmuştur. Aşağıda kullandığımız 2. satırdaki **/sbin/depmod --all --basedir=initrd, initrd/lib/modules/\$(uname -r)/moduller** altındaki modüllerimizin indeksinin oluşturuyor.

```
#döngüyle istediğimiz moduller initrd sistemimize dahil ediliyor.
for directory in {crypto,fs,lib} \
    drivers/{block,ata,md,firewire} \
    drivers/{scsi,message,pcmcia,virtio} \
    drivers/usb/{host,storage};
do
    #echo ${directory}
    find /lib/modules/$(uname -r)/kernel/${directory}/ -type f \
        -exec install {} initrd/lib/modules/$(uname -r)/moduller \;
done
/sbin/depmod --all --basedir=initrd    #modüllerin indeks dosyası oluşturuluyor
```

S9- distro/initrd/bin/systemd-udev

udev, Linux çekirdeği tarafından sağlanan bir altyapıdır ve donanım aygıtlarının dinamik olarak algılanmasını ve yönetilmesini sağlar. systemd-udev ise udev'in bir bileşenidir ve donanım olaylarını işlemek için kullanılır. Daha detaylı bilgi için udev yazısında anlatılmıştır. systemd için **/lib/systemd/systemd-udev**, no systemd için **/sbin/udev** kullanılır. Biz systemd için tasarladığımız için **/lib/systemd/systemd-udev** kullanıyoruz.

```
cp /lib/systemd/systemd-udev initrd/bin/systemd-udev #sistemden kopyalandı..
lddscript initrd/bin/systemd-udev initrd/ #sistemden kütüphaneler kopyalandı..
```

S10- distro/initrd/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.

udevadm komutu, birçok farklı parametreyle kullanılabilir. İşte bazı yaygın kullanımları:

udevadm info: Bu komut, belirli bir aygıt hakkında ayrıntılı bilgi sağlar. Örneğin, udevadm info -a -n /dev/sda komutunu kullanarak /dev/sda aygıtıyla ilgili ayrıntıları alabilirsiniz.

udevadm monitor:* Bu komut, sistemdeki aygıtlarla ilgili olayları izlemek için kullanılır. Örneğin, udevadm monitor --property komutunu kullanarak aygıtların bağlanma ve çıkarma olaylarını izleyebilirsiniz.

udevadm trigger:* Bu komut, udev kurallarını yeniden değerlendirmek ve aygıtları yeniden tanımak için kullanılır. Örneğin, udevadm trigger --subsystem-match=block komutunu kullanarak blok aygıtlarını yeniden tanımlayabilirsiniz.

udevadm control: Bu komut, udev hizmetini kontrol etmek için kullanılır. Örneğin, udevadm control --reload komutunu kullanarak udev kurallarını yeniden yükleyebilirsiniz.

Bu sadece bazı temel kullanımlardır ve udevadm'nin daha fazla özelliği vardır. Daha fazla bilgi için, man udevadm komutunu kullanarak udevadm'nin man sayfasını inceleyebilirsiniz. **Not:** udevadm systemd ve no systemd için aynı kullanımdadır. İki sistem içinde geçerlidir.

```
cp /bin/udevadm initrd/bin/udevadm #sistemden udevadm kopyalandı..
lddscript initrd/bin/udevadm initrd/ #sistemden kütüphaneler kopyalandı..
```

Yardımcı Konular

S12- distro/etc/udev/rules.d--S13- distro/lib/udev/rules.d

"rules" kelimesi, Linux işletim sistemi veya bir programda belirli bir davranışı tanımlayan ve yönlendiren kuralları ifade eder. Bu kurallar, sistem veya programın nasıl çalışacağını belirlemek için kullanılır ve genellikle yapılandırma dosyalarında veya betiklerde tanımlanır.

Linux'ta "rules" terimi, genellikle udev kuralları veya iptables kuralları gibi belirli bileşenlerle ilişkilendirilir.

udev kuralları, Linux çekirdeği tarafından sağlanan bir altyapıdır ve donanım aygıtlarının nasıl tanınacağını ve nasıl işleneceğini belirlemek için kullanılır. Örneğin, bir USB cihazı takıldığında, udev kuralları bu cihazın nasıl adlandırılacağını ve hangi sürücünün kullanılacağını belirleyebilir.

Örnek bir udev kuralı:

```
ACTION=="add", SUBSYSTEM=="usb", ATTR{idVendor}=="1234",  
ATTR{idProduct}=="5678", RUN+="/path/to/script.sh"
```

Bu kural, bir USB cihazı eklendiğinde çalışacak bir betik belirtir. Kural, cihazın üretici kimliği (idVendor) ve ürün kimliği (idProduct) gibi özelliklerini kontrol eder ve belirli bir eylem gerçekleştirir.

Aşağıda sisteme ait kurallar initrd sistemimize kopyalanmaktadır.

```
cp /etc/udev/rules.d -rf initrd/etc/udev/  
cp /lib/udev/rules.d -rf initrd/lib/udev/
```

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. Bu tasarımda script dosya olacaktır. İçeriği aşağıdaki gibi olacaktır.

```
cat > initrd/init << EOF  
#!/bin/busybox ash  
PATH=/bin  
/bin/busybox mkdir -p /bin  
/bin/busybox --install -s /bin  
#*****  
export PATH=/bin:/sbin:/usr/bin:/usr/sbin  
  
[ -d /dev ] || mkdir -m 0755 /dev      #/dev dizini yoksa oluştur  
[ -d /root ] || mkdir -m 0700 /root   #/root dizini yoksa oluştur  
[ -d /sys ] || mkdir /sys             #/sys dizini yoksa oluştur  
[ -d /proc ] || mkdir /proc           #/proc dizini yoksa oluştur  
mkdir -p /tmp /run                    # /tmp ve /run dizinleri oluşturuluyor  
  
# sisteme izinler bağlanıyor(yükleniyor)  
mount -t devtmpfs devtmpfs /dev  
mount -t proc proc /proc  
mount -t sysfs sysfs /sys  
mount -t tmpfs tmpfs /tmp  
  
systemd-udevd --daemon --resolve-names=never #modprobe yerine kullanılıyor  
udevadm trigger --type=subsystems --action=add  
udevadm trigger --type=devices --action=add  
udevadm settle || true  
  
mkdir -p /disk                        # /dev/sdal diskini bağlamak için izin oluşturuluyor  
modprobe ext4                         #ext4 modülü yükleniyor harici olarak yüklememiz gerekiyor  
mount /dev/sdal /disk                 #diski bağlayalım  
  
# dev sys proc taşıyalım  
mount --move /dev /disk/dev  
mount --move /sys /disk/sys  
mount --move /proc /disk/proc  
  
exec switch_root /disk /sbin/init     #sistemi initrd içindeki initten sdal diskinde olan /sbin/init'e devrediyoruz.  
/bin/busybox ash                     #eğer üst satırdaki görev devir işlemi olmazsa bu satır çalışacak ve tty açılacaktır.  
EOF  
chmod +x initrd/init #init dosyasına çalıştırma izni veriyoruz.  
cd initrd  
find |cpio -H newc -o >initrd.img # initrd.img dosyasını initrd dizinine oluşturacaktır.  
cd ..
```

Yardımcı Konular

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

S15- distro/iso/initrd.img - S16- distro/iso/vmlinuz

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. initrd.img dosyası üretmek için

```
cp /boot/vmlinuz-$(uname -r) iso/boot/vmlinuz #sistemde kullandığım kerneli kopyaladım istenirde kernel derlenebilir.
mv initrd/initrd.img iso/boot/initrd.img #daha önce oluşturduğumuz **initrd.img** dosyamızı taşıyoruz.
```

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.

İso Oluşturma

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

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

qemu-system-x86_64 -cdrom distro.iso -m 1G komutuyla çalıştırıp test edebiliriz.

Bağımlılıkların Tespiti

İkili dosyasının iki tür derlenme şekli vardır(statik ve dinamik). Statik derleme gerekli olan kütüphaneleri içerisinde barındıran tek bir dosyadır. Dinamik olan ise gerekli olan kütüphane dosyaları ikili dosya dışında tutulmaktadır. İkili dosyamızın bağımlılıklarının bulunması için aşağıdaki scripti kullanabiliriz. Scripti lddscript.sh dosyası olarak kaydedip kullanabilirsiniz. **bash lddscript.sh /bin/ls /tmp/test** şeklinde kullandığımızda /tmp/test/ dizinine **ls** ikili dosyasının konumunu ve bağımlılıklarını kopyalayacaktır.

```
#!/bin/bash
#bash lddscript binaryPath binaryTarget
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

# copy the binary itself
##echo "---> copy binary itself"
##cp --parents -v "${path_to_binary}" "${target_folder}"

# copy the library dependencies
echo "---> copy libraries"
ldd "${path_to_binary}" | awk -F'[> ]' '{print $(NF-1)}' | while read -r lib
do
    [ -f "$lib" ] && cp -v --parents "$lib" "${target_folder}"
done
```

strip

strip komutu, derlenmiş bir programın veya paylaşılan bir kütüphanenin dosya boyutunu küçültmek için kullanılır. Bu komut, derleme sürecinde oluşturulan fazladan semboller ve hata ayıklama bilgilerini kaldırır. Bu sayede, dosyanın boyutu azalır ve gereksiz veri miktarı azalır.

strip komutunu kullanmak için, terminalde aşağıdaki şekilde bir komut satırı kullanabilirsiniz:

```
strip dosya_adı
```

Burada "dosya_adı" yerine, boyutunu küçültmek istediğiniz dosyanın adını belirtmelisiniz. Örneğin, "strip program" komutunu kullanarak "program" adlı bir dosyanın boyutunu küçültebilirsiniz.

strip komutu, genellikle Linux ve Unix tabanlı işletim sistemlerinde kullanılır. Bu komut, programların dağıtımı veya paylaşımı sırasında dosya boyutunu azaltmak için yaygın olarak kullanılır.

Bu komutun kullanımı, programların performansını artırabilir ve disk alanından tasarruf sağlayabilir. Ancak, hata ayıklama veya sembol bilgilerine ihtiyaç duyduğunuz durumlarda strip komutunu kullanmaktan kaçınmanız önemlidir.

Sonuç olarak, strip komutu, derlenmiş programların veya paylaşılan kütüphanelerin boyutunu küçültmek için kullanılan bir Linux komutudur.

ld(linker)

ld (linker) komutu, Linux sistemlerinde derlenmiş nesne dosyalarını birleştirmek ve yürütülebilir dosyalar veya paylaşılan kütüphaneler oluşturmak için kullanılan bir araçtır. Bu komut, derleyicinin çıktısı olan nesne dosyalarını alır ve bunları birleştirerek çalıştırılabilir bir dosya oluşturur.

ld komutu, bir programın bağımlılıklarını çözmek ve gerekli kütüphaneleri eklemek için kullanılır. Bu sayede program, çalıştırıldığında gereken tüm kütüphanelere erişebilir ve doğru şekilde çalışabilir.

Aşağıda basit bir örnek verilmiştir:

```
gcc -c main.c -o main.o
gcc -c helper.c -o helper.o
ld main.o helper.o -o program
```

Bu örnekte, gcc komutuyla main.c ve helper.c dosyaları derlenir ve nesne dosyaları (main.o ve helper.o) oluşturulur. Ardından, ld komutuyla bu nesne dosyaları birleştirilerek program adında bir çalıştırılabilir dosya oluşturulur.

ld komutu, Linux sistemlerinde programların derlenmesi ve çalıştırılması sürecinde önemli bir rol oynar ve programların doğru şekilde çalışmasını sağlar.

Linker Türleri

Linker, bir programın derlenmiş obje dosyalarını birleştirerek çalıştırılabilir bir dosya oluşturan bir yazılımdır. Linker, programın farklı bölümlerini bir araya getirir, sembol tablolarını oluşturur ve bağımlılıkları çözer.

Linux'ta kullanılan yaygın linker çeşitleri şunlardır:

GNU Linker (ld): GNU Projesi'nin bir parçası olan GNU Linker, Linux sistemlerinde en yaygın olarak kullanılan linker'dir. ld, derlenmiş obje dosyalarını birleştirir ve çalıştırılabilir bir dosya oluşturur. Ayrıca, dinamik bağlantı için gerekli olan dinamik kütüphaneleri de yönetir.

Gold Linker: Gold Linker, GNU Linker'a alternatif olarak geliştirilmiş bir linker'dir. Gold Linker, daha hızlı bağlama süreleri sunar ve büyük projelerde performans avantajı sağlar.

LLD (LLVM Linker): LLD, LLVM projesinin bir parçası olan modern bir linker'dir. LLD, hızlı bağlama süreleri ve düşük bellek tüketimi ile bilinir. Ayrıca, birden fazla platformda çalışabilme özelliğine sahiptir.

Bu linker çeşitleri, Linux sistemlerinde programların birleştirilmesi ve çalıştırılabilir dosyaların oluşturulması için kullanılır. Her bir linker'ın kendine özgü avantajları ve kullanım senaryoları vardır. Projenizin gereksinimlerine ve tercihlerinize bağlı olarak uygun olanı seçebilirsiniz.

```
# GNU Linker (ld) kullanım örneği
ld -o program program.o

# Gold Linker kullanım örneği
gold -o program program.o

# LLD (LLVM Linker) kullanım örneği
lld -o program program.o
```

Yardımcı Konular

cmake

cmake ninja ile derleme

```
cd build_folder  
cmake -G Ninja source_folder  
ninja
```

cmake derleme:

```
cd build_folder  
cmake -G Ninja source_folder  
cmake --build .
```

sudoers

sudoers dosyasını düzenleyerek, belirli kullanıcıların veya kullanıcı gruplarının parola sormadan sudo komutlarını çalıştırabilmesini sağlayabilirsiniz. Bunun için aşağıdaki adımları izleyebilirsiniz:

Terminali açın ve sudo visudo komutunu çalıştırın. Bu komut, sudoers dosyasını düzenlemek için kullanılır. Dosya açıldığında, aşağıdaki gibi bir satır bulun:

```
%sudo    ALL=(ALL:ALL) ALL
```

Bu satırın altına, aşağıdaki gibi bir satır ekleyin:

```
deneme ALL=(ALL) NOPASSWD: ALL
```

Burada deneme yerine parola sormadan sudo komutlarını çalıştırmak istediğiniz kullanıcının adını yazın. Dosyayı kaydedin ve kapatın.

Artık belirtilen kullanıcı, parola sormadan sudo komutlarını çalıştırabilecektir. Ancak, bu işlemi yaparken dikkatli olun. Parola sormadan sudo kullanmak, güvenlik risklerine yol açabilir. Bu nedenle, yalnızca güvendiğiniz kullanıcılar için bu yapılandırmayı kullanmanız önerilir.

Bu yöntemle sudoers dosyasını düzenleyerek, belirli kullanıcıların veya kullanıcı gruplarının parola sormadan sudo komutlarını çalıştırabilmesini sağlayabilirsiniz.

Yukarıdaki yöntem yerine aşağıdaki gibi yapabiliriz.

```
sudo touch /etc/sudoers.d/deneme  
sudo chmod 440 /etc/sudoers.d/deneme
```

/etc/sudoers.d/deneme içeriğine aşağıdaki gibi yapılandırılalım.

```
%sudo    ALL=(ALL) NOPASSWD:ALL  
deneme   ALL=(ALL) NOPASSWD:ALL
```


polkit

Polkit, Linux sistemlerinde yetkilendirme ve erişim kontrolü sağlayan bir altyapıdır. Polkit kuralları, belirli kullanıcıların veya kullanıcı gruplarının belirli işlemleri gerçekleştirmesine izin vermek veya engellemek için kullanılır.

Polkit kurallarını eklemek için aşağıdaki adımları izleyebilirsiniz:

Polkit kurallarının bulunduğu dizine gidin. Genellikle **/etc/polkit-1/rules.d/** dizininde bulunur. Bir metin düzenleyici kullanarak yeni bir dosya oluşturun veya mevcut bir dosyayı düzenleyin. Dosya adı, genellikle **XX-name.rules** formatında olmalıdır. Burada XX, kuralların uygulanma sırasını belirten bir sayıdır ve name ise dosyanın açıklamasını temsil eder. Dosyaya aşağıdaki gibi bir polkit kuralı ekleyin:

```
polkit.addRule(function(action, subject) {  
    if (action.id == "org.example.customaction" && subject.user == "username") {  
        return polkit.Result.YES;  
    }  
});
```

Yukarıdaki örnekte, **org.example.customaction** adlı bir eylem için **username** kullanıcıasına izin veriliyor. Bu kuralı ihtiyaçlarınıza göre düzenleyebilirsiniz.

Dosyayı kaydedin ve düzenlediğiniz kuralların etkili olması için Polkit servisini yeniden başlatın. Bu işlem için aşağıdaki komutu kullanabilirsiniz:

```
sudo systemctl restart polkit
```

Artık polkit kurallarınız etkinleştirilmiş olmalı ve belirlediğiniz yetkilendirmeler uygulanmalıdır.

Polkit kuralları, sistem yöneticilerinin belirli işlemleri kontrol etmesine ve kullanıcıların erişimini yönetmesine olanak tanır. Bu sayede sistem güvenliği ve yetkilendirme düzeyi daha iyi kontrol edilebilir.

Tüm Uygulamalarda İzin Verme

```
polkit.addRule(function(action, subject) {  
    return polkit.Result.YES;  
});
```

Bir Gruba İzin Verme

```
polkit.addRule(function(action, subject) {  
    if (subject.isInGroup("test")) {  
        return polkit.Result.YES;  
    }  
});
```

Yardımcı Konular

Bir Grup-User-Uygulamaya İzin Verme

```
polkit.addRule(  
  function(action,subject)  
  {  
    if ( (action.id == "org.freedesktop.policykit.exec") &&  
        (action.lookup("user") == "root") &&  
        (action.lookup("program") == "/path/to/script") &&  
        (subject.isInGroup("someGroup") ) )  
      return polkit.Result.YES;  
  
    return polkit.Result.NOT_HANDLED;  
  }  
);
```

user-dirs

user-dirs, Linux işletim sistemlerinde kullanıcıların özel klasörlerini içeren bir sistemdir. Bu klasörler genellikle "Masaüstü", "Belgeler", "İndirilenler" gibi isimlerle tanımlanır ve kullanıcıların dosyalarını düzenlemelerini kolaylaştırır.

user-dirs klasörünü kaldırmak için aşağıdaki adımları izleyebilirsiniz:

Terminali açın ve aşağıdaki komutu girin:

```
nano ~/.config/user-dirs.dirs
```

Bu komut, user-dirs.dirs dosyasını açacaktır. Bu dosya, kullanıcı klasörlerinin yolunu ve adını içerir.

Dosyayı düzenlemek için, kaldırmak istediğiniz klasörün satırını bulun ve **"#"** işaretiyle başlayan bir yorum satırı yapın. Örneğin, **"Masaüstü"** klasörünü kaldırmak istiyorsanız, satırı aşağıdaki gibi düzenleyin:

```
#XDG_DESKTOP_DIR="$HOME/Masaüstü"
```

Dosyayı kaydetmek ve kapatmak için **"Ctrl + X"** tuşlarına basın, ardından **"Y"** tuşuna basın ve Enter tuşuna basın.

Değişikliklerin etkili olması için oturumu kapatıp tekrar açın veya aşağıdaki komutu girin:

```
xdg-user-dirs-update
```

Bu adımları takip ederek user-dirs klasörünü Linux sisteminizden kaldırabilirsiniz. Ancak, dikkatli olun ve yanlışlıkla önemli dosyaları silmemeye özen gösterin.

Yeni oluşturulacak kullanıcılarda oluşturulmamasını istiyorsak **/etc/skel/.config/user-dirs.dirs** dosyasını silmeliyiz.

Ek Konular

openrc Servis Yönetimi

Servisi Başlangıca Ekleme

```
rc-update add servis default
```

Servisi Başlangıçtan Kaldırma

```
rc-update del servis default
```

Servislerin Çalışma Sırası

Servislerin çalışma sırasını **rc-update -v** ile öğrenebiliriz. Bazen servisin başka servisin çalışmasını beklemesi gerekebilir. Aşağıda **sshd** çalıştıktan sonra çalışacaktır.

```
depend() {  
    after sshd  
}
```

Bazen servisin başka başka bir servise ihtiyacı olabilir. Aşağıda bu servisin çalışması için **net** servisine ihtiyacı vardır.

```
depend() {  
    need net  
}
```

Servislerin çalışma sırasını alfabetik olarak yapmaktadır. adlandırma yaparken buna dikkat etmek gerekir. Yani **axxxx** ile başlayan bir dosyayı **bxxxx** dosyasına göre daha önce çalıştıracaktır.

Basit Servis Komutlarını Çalıştırma

/etc/local.d/ dizini içine basit betikler konabilir.

```
#!/sbin/openrc-run description="load modules runsystemd"  
name="runsystemda"          command="/etc/local.d/runsystemd"          command_args=""  
pidfile="/run/runsystemd.pid" command_background=true depend() {  
    after lightdm  
}
```

Yukarıdaki **runsystemda** servisi **/etc/local.d/runsystemd** dosyasını çalıştırmaktadır.

```
#/etc/local.d/runsystemd #!/bin/sh chmod 755 -R /run/systemd
```

Sistem Dili Değiştirme

Dil ayarlama

Sistem dilini ayarlamak için öncelikle **/etc/locale.gen** dosyamızı aşağıdaki gibi düzenleyelim.

Dil kodlarına **/usr/share/i18n/locales** içerisinden ulaşabilirsiniz.

Karakter kodlamalara **/usr/share/i18n/charmaps** içinden ulaşabilirsiniz.

```
tr_TR.UTF-8 UTF-8
```

Not: En altta boş bir satır bulunmalıdır.

Ardından **/lib64/locale** dizini yoksa oluşturalım.

Ek Konular

```
mkdir -p /lib64/locale/
```

Şimdi de çevresel değişkenlerimizi ayarlamak için /etc/profile.d/locale.sh dosyamızı düzenleyelim.

```
#!/bin/sh # Language settings export LANG="tr_TR.UTF-8" export LC_ALL="tr_TR.UTF-8"
```

Not: Türkçe büyük küçük harf dönüşümü (i -> İ ve ı -> I) ascii standartına uyumsuz olduğu için LC_ALL kısmını türkçe ayarlamayı önermiyoruz. Bunun yerine C.UTF-8 veya en_US.UTF-8 olarak ayarlayabilirsiniz.

Son olarak locale-gen komutunu çalıştıralım.

```
locale-gen
```

Eğer /lib64/locale/ dizine okuma izniniz yoksa verelim.

```
chmod 755 -R /lib64/locale/
```

1. Yöntem

/etc/default/locale dosyasını root olarak bir metin editörü ile açın.

Türkçe için : LANG=tr_TR.UTF-8 İngilizce için : LANG=en_US.UTF-8

Sistemi yeniden başlattığınızda seçtiğiniz dil aktif olacaktır.

2. Yöntem

/etc/profile.d/locale.sh dosyanı oluşturun içeriğini aşağıdaki gibi ayarlayın.

```
# Language settings export LANG="tr_TR.UTF-8" export LC_ALL="tr_TR.UTF-8"
```

/etc/profile.d/ dizin erişim iznini 755 yapın.

profile

/etc/profile dosya içeriğini aşağıdaki gibi bir betik bulunmalıdır.

/etc/profile dosyanının içerisinde aşağıdaki betik olmalıdır. Bu betik **/etc/profile.d** içerisinde betikler varsa tüm kullanıcılar için çalıştırılmasını sağlar.

```
if [ -d /etc/profile.d ]; then
    for i in /etc/profile.d/*.sh; do
        if [ -r $i ]; then
            . $i
        fi
    done unset i
```

```
fi
```

3.Yöntem

ayarlarını değiştirmek istediğimiz kullanıcı dizinindeki ~/.profile dosyasının içeriğine aşağıdaki kod satırını eklemeliyiz.

```
export LANG="tr_TR.UTF-8" export LC_ALL="tr_TR.UTF-8"
```

Kullanıcı Sistem Ayarları

Profile Dosyası

/etc/profile dosyası, Linux sistemlerinde kullanıcıların oturum açtıklarında çalıştırılan bir betik dosyasıdır. Bu dosya, tüm kullanıcılar için ortak kabuk ayarlarını ve ortam değişkenlerini tanımlar. Kullanıcı oturumu başlatıldığında, /etc/profile dosyası sistem genelindeki ayarları yükler ve kullanıcı oturumuna uygular. Bu dosya, kullanıcıların kabuk ortamlarını yapılandırmak ve özelleştirmek için kullanılır. Örneğin, PATH değişkenini tanımlamak veya diğer kabuk ayarlarını yapılandırmak için /etc/profile dosyası düzenlenebilir. Bu dosya, sistem genelinde tutarlı bir kabuk ortamı sağlamak için önemlidir.

```
/etc/profile /etc/profile.d/* /etc/environment ~/.profile, veya ~/.bash_profile veya
~/.login veya ~/.zprofile giriş kabuğunuza bağlı olarak ~/.pam_environment (yalnızca
terminalde çalışan kabuklar için) /etc/bash.bashrc, /etc/zshrc, ~/.bashrc, ~/.zshrc,
vesaire.
```

Not: /etc/profile.d/ dizinine root dışında kullanıcılar erişim sağlaması için 755 yapmalısınız.

**** profile****

/etc/profile dosyasının içerisinde aşağıdaki betik olmalıdır. Bu betik **/etc/profile.d** içerisinde betikler varsa tüm kullanıcılar için çalıştırılmasını sağlar.

```
if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done unset i
fi
```

fi

adduser ve useradd Kullanımı

adduser ve useradd komutları, Linux işletim sisteminde kullanıcı hesapları oluşturmak için kullanılan iki farklı komuttur. Bu iki komut arasındaki temel farklar şunlardır:

1. Kullanım Kolaylığı:

adduser, interaktif bir arayüz sağlar ve kullanıcı oluşturma işlemi sırasında bir dizi soru sorarak kullanıcı dostu bir yaklaşım sunar. useradd ise daha düşük seviyeli bir komuttur ve tüm parametreleri elle belirtmenizi gerektirir. Bu nedenle, kullanımı daha teknik ve karmaşıktır.

Örnek kullanım:

language-bash

```
# adduser komutuyla kullanıcı oluşturma sudo adduser yeni_kullanici
```

```
# useradd komutuyla kullanıcı oluşturma sudo useradd -m -s /bin/bash yeni_kullanici
```

2. Varsayılan Davranışlar:

adduser, varsayılan olarak kullanıcı için ev dizini oluşturur ve gerekli ayarları yapar. useradd ise temel bir kullanıcı hesabı oluşturur ancak ek parametreler belirtilmediği sürece ek işlemleri gerçekleştirmez.

3. Güvenlik ve Uyumluluk:

Ek Konular

adduser, kullanıcı oluřtururken bazı gvenlik kontrolleri yapar ve sistem uyumluluęunu saęlamak iin ek adımlar atar. useradd ise daha esnek bir yapıya sahiptir ve kullanıcı oluřturma iřlemini daha zelleřtirilmiř bir řekilde gerekleřtirmenize olanak tanır.

Sonu olarak, genel olarak adduser komutu, kullanıcı dostu bir arayz sunar ve standart kullanıcı oluřturma iřlemleri iin tercih edilirken, useradd komutu daha teknik detaylara hakim olan kullanıcılar tarafından tercih edilebilir. Her iki komut da kullanıcı ynetimi iin nemli aralardır ve ihtiyaa gre tercih edilmelidir.