distro Dokümanı

version

distro Linux

Aralık 23, 2023

Contents

Dağıtım

Dağıtım Hazırlama

Dağıtım Nedir?

> Linux kullanmaya başlayan kişilerin en çok karşılaştığı terimlerden birisi **dağıtım** kelimesidir.

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

> Linux dağıtımları genelde **kernel** ve uygulamalardan oluşur. Kernel ve uygulamaların kodları github, gitlab vb. ortamlarda paylaşıldığı için sıfırdan bir dağıtımda oluşturmak mümkündür. Bunu oluştururken yasal olmayan hiçbir işlem yapmamış oluruz. Çünkü genel felsefe ***"Al, kullan, değiştir ve kimseden izin almadan dağıt"*** şeklinde olduğunu hatırlayalım.
>
> Bu doküman basit seviveyede bir dağıtım oluşturmak ve kurulabilir bir medya dosyası(iso dosya) nasıl hazırlanacağını anlatan bir rehber olacaktır.

Bu dokümanı hazırlanmasında ve anlatılanları tecrübe ederek öğrenmeme katkısı olan **Turkman Linux** dağıtım ekibiden @sulincix(Ali Rıza KESKİN) ve Celaleddin AKARSU'ya teşekkür ederim.

Dağıtım Nasıl Hazırlanmalı?

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

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

Burada sıralanan maddeler konu başlıkları olarak anlatılacaktır.

Ortam Hazırlama

Dağıtım İçin Ortamın Hazırlanması

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

Bir dağıtım hazırlamak için çeşitli paketler lazım. Bu paketler;

- debootstrap : Dağıtım hazırlaken 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 içiçn kullanılan uygulama.
- zstd : Yüksek sıkıştırma yapan sıkıştırma uygulaması
- grub-mkrescue : Hazırladığımız iso dizinini iso yapmak için kullanılan uygulama

qemu-system-x86 : iso dosyalarını test etmek ve kullanmak için sanal makina emülatörü uygulaması.

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

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

Derleme Türleri

Derleme türleri

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

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

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

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

Bu dosyayı derleyip çalıştırılabilir dosya elde edelim.

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

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

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

Dynamic derleme

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

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

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

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

Burada **libc.so.6** ve **ld-linux-x86_64.so.2** dosyaları tamamında ortaktır ve **glibc** tarafından sağlanır. Dynamic derlenmiş bir dosyanın derlenmesi veya çalıştırılabilmesi için tüm bağımlılıklarının sistemde bulunması gereklidir.

Static derleme

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

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

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

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

Static ve Dynamic derlemenin kıyaslanması

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

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

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

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

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

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

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

Kütüphane oluşturma

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

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

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

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

Kütüphane derlemek için aşağıdaki iki komutu arka arkaya kullanmalıyız.

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

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

Kütüphane aşağıdaki gibi kullanılabilir.

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

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

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

Kütüphaneyi Sisteme Dahil Etme ve Kullanma

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

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

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

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

```
$ ldd ./main
linux-vdso.so.1 (0x00007fffab5e8000)
libdeneme.so => not found
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ff73002c000)
/lib64/ld-linux-x86-64.so.2 (0x00007ff73020d000)
```

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

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

Kütüphane Dosyasının Konumunu İsteğe Göre Belirleme(rpath)

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

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

Bunun için;

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

Kütüphaneyi Uygulama İçine Gömme(Static Derleme)

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

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

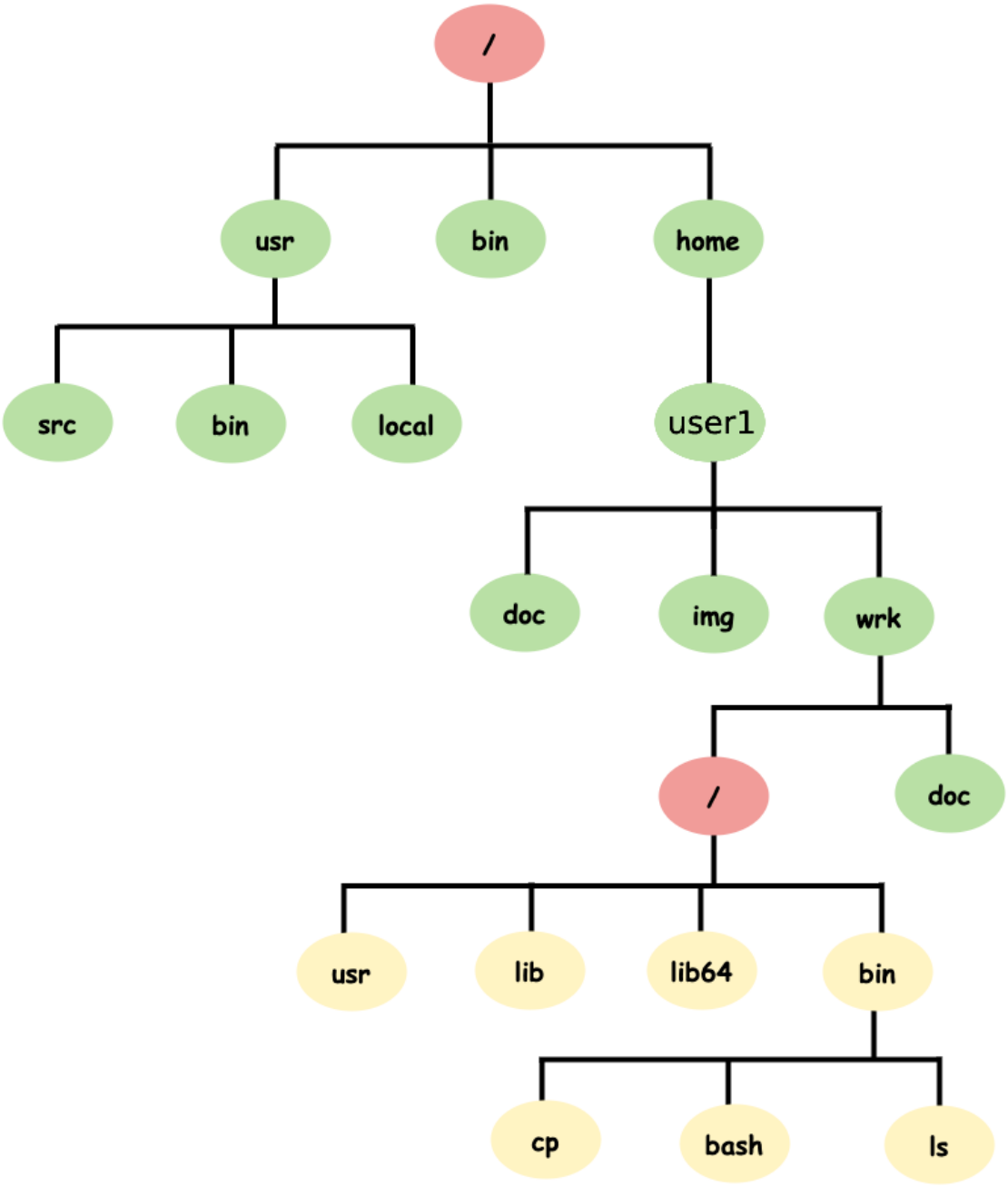
```
$ gcc main.o ./deneme.o -o main -static
$ ldd ./main
özdevimli bir çalıştırılabilir değil
$ ./main
Merhaba Dünya
```

Chroot Kullanımı

Chroot Nedir?

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

- Sistem tasarlama
- Sitem üzerinde yeni dağıtımlara müdahale etme ve sorun çözme
- Kullanıcının yetkilerini sınırlandırma.
- Kullanıcıyı sistemden yalıtma.
- Güvenlik.
- 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. Bu işlem için küçük bir örnek yapalım. çalıştığımız dizin wrk dizin.

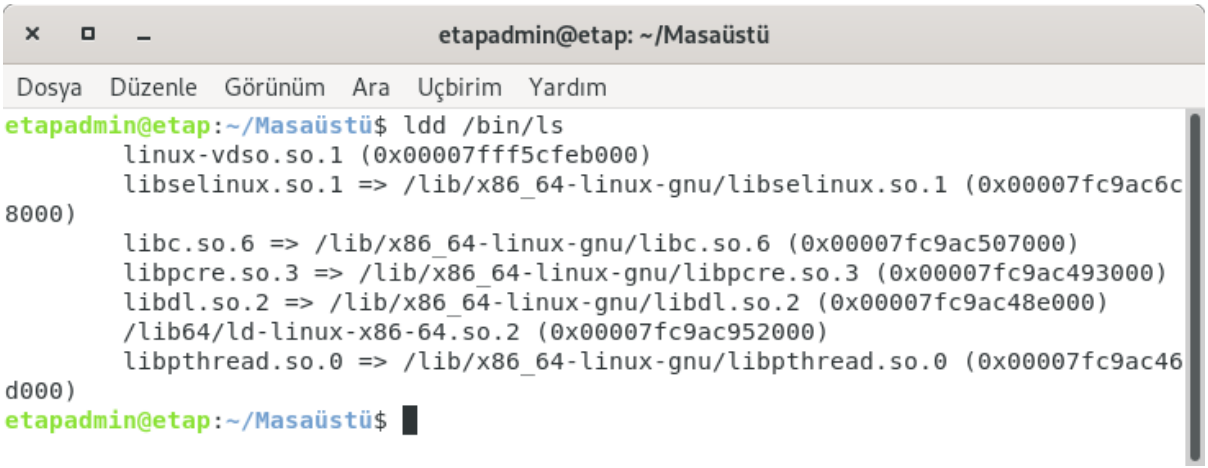
Kök Dizin Oluşturma;

```
sudo chroot /home/user1/wrk seklinde yapabiliriz.
```

Sistemi Kaldırmak;

```
sudo rm -rf /home/user1/wrk komutuyla kaldırabiliriz.
```

Bu yapının oluşturulması 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.



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

Bağımlılık Scripti

lldscript.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

# 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
```

Basit Sistem Oluşturma

Bu örnekte masaüstünde test dizini oluşturup ve işlemleryapıldı. ls, rmdir, mkdir ve bash komutlarından oluşan sistem hazırlama.

ls Komutu

```
bash lldscript.sh /bin/ls $PWD/test/ #komutunu kullanmalıyız.
```

Bu işlemi diğer komutlar içinde sırasıyla yapmamız gerekmektedir. rmdir Komutu ----------

```
bash lldscript.sh /bin/rmdir $PWD/test/ #komutunu kullanmalıyız.
```

mkdir Komutu

```
bash lldscript.sh /bin/mkdir $PWD/test/ #komutunu kullanmalıyız.
```

Chroot Kullanımı

bash Komutu

```
bash lldscript.sh /bin/bash $PWD/test/ #komutunu kullanmalıyız.
```

chroot sistemde Çalışma

```
sudo chroot $PWD/test komutunu kullanmalıyız.
```

çıkış için ise ***exit*** komutunu kullanmalıyız.

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

Temel Paketleri Derleme

Temel Paketler

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

Bunlar;

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

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

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

Sonuç olarak bash paketini derlerken paketin;

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

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

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

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

Listede bulunan paketler sırasıyla nasıl derleneceği ayrı başlıklar altında anlatılacaktır.

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

```
make install DESTDIR=$HOME/rootf # 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 bir hata alacağız.

```
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.

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 doayası
#include<stdio.h>
#include<readline/readline.h>
void main()
{
char* msg=readline("Adını 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.

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 işlevlere 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 libtinfow.so.6
ln -s libncursesw.so.6 libtinfo.so.6
ln -s libncursesw.so.6 libncurses.so.6
```

kmod

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 çıkarttığımzda kernel derlemek ciddi bir iş yükü ve karmaşa yaratacaktır.

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

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 hazırlayın:

```
./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.

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/lsmod
```

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/lsmod
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

dmesg ile log kısmında eklendiğinde veya modul komutlarının kullanılmasının sonuçlarını görebiliriz.

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. 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 seçine sahip olan bir programdır. Bu araçlar initramfs sisteminde ve sistem genelinde sıkça kullanılabilir. Busybox aşağıdaki gibi kullanılır. Örneğin, dosya listelemek için ls komutunu kullanmak isterseniz:

```
$ busybox ls
```

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

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

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

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

Busybox derlemek için öncelikle **make defconfig** kullanılarak veya önceden oluşturduğumuz yapılandırma dosyasını atarak yapılandırma işlemi yapılır. Ardından eğer static derleme 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 milimal kök dizin oluşturabiliriz. Burada static yapı kallanılmayacaktır. 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 kullabilirsiniz.

```
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
```

Paket Sistemi Hazırlama

initrd Hazırlama

Linux Tabanlı Sistem 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 doyamız 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 yapılma 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 yapalacaktır. Örneğin S1 ifadesi satır 1 anlamında anlatımı kolaylaştımak 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/lsmod                                #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-udevd                       #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 dizin 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/lsmod      #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 rmmode 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 modullerimizin 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-udevd

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

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

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 kurralar 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 dizinler 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/sda1 diskini bağlamak için dizin oluşturuluyor
        modprobe ext4           #ext4 modülü yükleniyor harici olarak yüklememiz gerekiyor
        mount /dev/sda1 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 sda1 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 ..
```

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 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 kullanılabililir.

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
```

Sistem Hazırlama

İso Kurulumu

Qemu Kullanımı

Qemu Nedir?

Açık kaynaklı sanallaştırma aracıdır.

Kaynaktan 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 belilemek 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

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

qemu Host Erişimi:

kendi ipsi:10.0.2.15

ana bilgisayar 10.0.0.2 olarak ayarlıyor.

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

İki Bölüm Kurulum

Bu bölümde **Ext4** dosya sistemine grub kullanarak kurulum anlatılacaktır. Anlatım boyunca **/dev/sda** diski üzerinden örnekleme 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 modulleri **modprobe** komutuyla yüklenmeli.

Disk Hazırlanmalı

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

1. **cfdisk komutuyla disk bölümlendirilmeli.**

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

2. gpt seçilmeli
3. 512 MB type vfat alan(sda1)
4. geri kalanı type linux system(sda2)
5. write
6. quit

6. Bu işlem sonucunda sadece sda1 sda2 olur 6. 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.

```
$ 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 paketinini 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). 3. 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.

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 modulleri **modprobe** komutuyla yüklenmeli.

Disk Hazırlanmalı(legacy)

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

1. **cfdisk komutuyla disk bölümlendirilmeli.**

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

2. dos seçilmeli
3. type linux system
4. write
5. quit
6. Bu işlem sonucunda sadece sda1 olur
7. 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.

```
$ sync
```

Bootloader kurulumu

grub kurulumu yapmak için grub paketinini 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). 3. dev/sda1 diskimizim 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
```

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

1. **cfdisk komutuyla disk bölümlendirilmeli.**

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

2. gpt seçilmeli
3. 512 MB type uefi alan(sda1)
4. geri kalanı type linux system(sda2)
5. write
6. quit

6. Bu işlem sonucunda sadece sda1 sda2 olur 6. 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.

```
$ 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 paketinini 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 değise **/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.

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

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). 3. 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.