Contents

Sistem

Giriş Seçenekleri

initramfs

initramfs dosyası kernel 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. initramfs dosyası üretmek için öncelikle bir dizin oluşturulur. Paketlenen dosya gzip veya lzma ile sıkmıştırılabilir veya sıkıştırılmadan da kullanılabilir. Bu dizine gerekli dosyalar eklenir ve aşağıdaki gibi paketlenir.

```
cd /initrd/dizini/
find . | cpio -o -H newc | gzip -9 > /boot/initrd.img-xxx
```

Kernel initrd dosyasını ram üzerine yükler ve içerisindeki /init dosyasını çalıştırır.

Örneğin aşağıdaki gibi bir C dosyamız olsun.

```
#include <stdio.h>
int main(){
    printf("Hello World!\n");
    while(1); // programın bitmesini engellemek için
    return 0;
}
```

Bu dosyayı static olarak derleyelim ve initramfs dosyasının içine koyup paketleyelim.

```
mkdir /tmp/initrd
cd /tmp/initrd
gcc -o /home/deneme/main.c /tmp/initrd/init -static
find . | cpio -o -H newc > /home/deneme/initrd.img
```

Daha sonra da qemu kullanarak test edelim.

```
qemu-system-x86_86 --enable-kvm -kernel /boot/vmlinuz-5.17 -initrd /home/deneme/initrd.img -append "quiet" -m 512m
```

Eğer tüm adımları doğru yaptıysanız ekranda hello world yazısı ile karşılaşacaksınız. Ayrıca kendi işletim sisteminizi çalıştırmış olacaksınız.

Teorik olarak kernel ve initramfs tek başına bir işletim sistemi sayılabilir. Genel olarak linux dağıtımlarında sistemin kurulu olduğu diskte GNU sistemi bulunur ve kernel ve initramfs kullanılarak bu sistemdeki **/sbin/init** dosyası çalıştırılır. Bu dosya servis yöneticisi dosyamızdır ve sistemin geri kalanının çalışmasını sağlar.

Yukarıdaki anlatımda initramfs nasıl çalıştığından söz edildi. Şimdi ise bir linux dağıtımında kullanılması için gereken işlemler üzerinde durulacaktır. Öncelikle initramfs oluşturma dizinimize gereken modülleri eklemeliniz. Bunun için /lib/modules/xxx içerisindeki dosyaları initramfs içine kopyalayalım.

```
...
mkdir -p /tmp/initrd/lib/modules/
for directory in {crypto,fs,lib} \
    drivers/{block,ata,md,firewire} \
    drivers/{scsi,message,pcmcia,virtio} \
```

```
    drivers/usb/{host,storage}; do
    find /lib/modules/$(uname -r)/kernel/${directory}/ -type f \
        -exec install {} /tmp/initrd/lib/modules/$(uname -r)/ \;
done
depmod --all --basedir=/tmp/initrd
...
```

Yukarıdaki örnekte ilk önce gerekli modülleri kopyaladık ve ardından modüllerin listesini güncellemek için **depmod** komutunu kullandık. Bu modülleri yüklemek için /init dosyamız içinde **modprobe** komutunu kullanabiliriz. Bu dosya genellikle **busybox ash** kullanılarak yazılır. Bunun için öncelikle busybox dosyamız initramfs dizinine kopyalanır. Ve ardından sembolik bağları atılarak komutları kullanılabilir hale getirilir.

```
...
mkdir -p /tmp/initrd/bin
install /bin/busybox /tmp/initrd/busybox
chroot /tmp/initrd /busybox --install -s /bin
...
```

Burada eğer busybox static olarak derlenmemişse çalışmayacağı için **glibc** ve gereken diğer dosyalarımızı da eklememiz gerekmektedir. Bunun için önce **ldd** komutu ile bağımlılıkları öğrenilir ve bağımlılık dosyası initramfs dizininde /lib içine yerleştirilir. Bu işlem tüm alt bağımlılıklarda tekrarlarır. Aşağıdaki örnekte bağımlılıkların bulunması ve kopyalanması için bir fonksiyon oluşturulmuştur.

```
...
function get_lib(){
    ldd $1 | cut -f3 -d" " | while read lib ; do
        if [[ "$lib" == "" ]] ; then
            : empty line
        elif ! echo ${libs[@]} | grep $lib >/dev/null; then
            echo $lib
            get_lib $lib
        fi
    done | sort | uniq
}
function install_binary(){
    get_lib $1 | while read lib ; do
        file=/tmp/initrd/lib/$(basename $lib)
        if [[ ! -f $file ]] ; then
            install $lib $file
        fi
    done
    install $1 /tmp/initrd/bin/$(basename $1)
}
mkdir -p /tmp/initrd/lib/
ln -s lib /tmp/initrd/lib64
install_binary /bin/busybox
...
```

Eğer Bazı dağıtımlarda /lib64 bulunur. Bu sebeple lib64 adında bir sembolik bağ oluşturmamız gerekebilir.

Modülleri yüklemek için elle **modprobe** komutu kullanılabilir. Bu sayede initramfs dosyamıza eklediğimiz modüllerin tamamını yükleyip donanımları tanıması sağlanabilir.

```
...
find /lib/modules/$(uname -r)/ -type f | while read module ; do
    module_name=$(basename "$module"| sed "s/\..*//g")
    if echo ${module_name} | grep "debug" ; then
        : ignore debug module
    else
        modprobe ${module_name}
    fi
done
...
```

Bu işlemin dezavantajı hem yavaş çalışması hem de gerekli olmayan modüllerin de yüklenmesidir. Bu yüzden bu yöntem yerine alternatif olarak **eudev** veya **systemd-udev** kullanılabilir. Bunun için initramfs dizinimize aşağıdaki eklemeler yapılır.

```
...
# eudev için
install_binary /sbin/udevd
# systemd-udev için
install_binary /lib/systemd/systemd-udevd
# Her ikisi için
install_binary /sbin/udevadm
...
```

Daha sonra initramfs içerisindeki /init içinde aşağıdaki komutlar çalıştırılmalıdır.

```
...
# systemd-udev için
systemd-udevd --daemon
# eudev için
udevd --daemon
# Her ikisi için
udevadm trigger -c add
udevadm settle
...
```

Eğer systemd kullanmayan bir dağıtım geliştirecekseniz veya initramfs dosyasının daha az boyutlu olmasını istiyorsanız **eudev** tercih etmelisiniz.

Initramfs dosyasının birinci amacı ana sistemi diske bağlayıp görevi servis yöneticisine devretmektir. Bu sebeple önce disk bağlanır ve ardından içerisine **/dev**, **/sys**, **/proc** dizinleri bağlanır ve **switch_root** kullanılarak ana sisteme geçilir.

```
# Eğer yoksa dev sys proc dizinlerini oluşturalım.
mkdir -p /dev /sys /proc
# dev sys proc bağlayalım
mount -t devtmpfs devtmpfs /dev
mount -t sysfs sysfs /sys
mount -t proc proc /proc
...
# diski bağlayalım
mount $root /new_root
# dev sys proc taşıyalım
mount --move /dev /new_root/dev
```

```
mount --move /sys /new_root/sys
mount --move /proc /new_root/proc
# /dev/root oluşturalım (isteğe bağlı)
ln -s $root /new_root/dev/root
# servis yöneticisini çalıştıralım.
exec switch_root /new_root $init
```

Yukarıdaki örnekte **$root** ve **$init** değişkenleri değerini /proc/cmdline içerisinden okumalısınız. varsayılan init değeri **/sbin/init** olmalıdır.

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.

kmod Nedir? Nasıl Yazılır ve Kullanılır?

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.

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

```
ln -s kmod /bin/depmod
ln -s kmod /bin/insmod
ln -s kmod /initrd/bin/lsmod
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.

hello.c dosyamız

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

Makefile dosyamız

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

modülün derlenmesi ve eklenip kaldırılması

```
make

insmod my_modul.ko // modül kernele eklendi.

lsmod | grep my_modul //modül yüklendi mi kontrol ediliyor.

rmmod my_modul // modül kernelden çıkartılıyor.
```

Not:

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

udev Nedir? Niçin Kullanılır?

systemd-udevd, Linux sistemlerinde donanım aygıtlarının eşleştirilmesi ve yönetimi için kullanılan bir sistem hizmetidir. Bu hizmet, udev adı verilen bir alt sistem üzerinde çalışır ve donanım olaylarını izler, aygıt dosyalarını oluşturur ve aygıtların durumunu günceller.

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.

systemd-udevd, donanım olaylarını izler ve bu olaylara göre belirli eylemler gerçekleştirir. Örneğin, bir USB cihazı takıldığında veya çıkarıldığında, systemd-udevd bu olayı algılar ve ilgili aygıt dosyasını oluşturur veya kaldırır. Ayrıca, donanım aygıtlarının durumunu güncellemek için de kullanılır. Örneğin, bir ağ arabirimi devre dışı bırakıldığında, systemd-udevd bu durumu algılar ve ilgili aygıt dosyasını günceller.

systemd-udevd, Linux sistemlerinde donanım aygıtlarının dinamik olarak yönetilmesini sağlayarak sistem yöneticilerine büyük bir esneklik ve kolaylık sağlar. Bu hizmet, donanım aygıtlarının otomatik olarak algılanmasını ve yapılandırılmasını sağlar, böylece kullanıcılar yeni bir aygıt takıldığında veya çıkarıldığında manuel olarak müdahale etmek zorunda kalmazlar.

Etahta Ayarları

initramfs

initramfs dosyası kernel 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. initramfs dosyası üretmek için öncelikle bir dizin oluşturulur. Paketlenen dosya gzip veya lzma ile sıkmıştırılabilir veya sıkıştırılmadan da kullanılabilir. Bu dizine gerekli dosyalar eklenir ve aşağıdaki gibi paketlenir.

```
cd /initrd/dizini/
find . | cpio -o -H newc | gzip -9 > /boot/initrd.img-xxx
```

Kernel initrd dosyasını ram üzerine yükler ve içerisindeki /init dosyasını çalıştırır.

Örneğin aşağıdaki gibi bir C dosyamız olsun.

```
#include <stdio.h>
int main(){
    printf("Hello World!\n");
    while(1); // programın bitmesini engellemek için
```

```
    return 0;
}
```

Bu dosyayı static olarak derleyelim ve initramfs dosyasının içine koyup paketleyelim.

```
mkdir /tmp/initrd
cd /tmp/initrd
gcc -o /home/deneme/main.c /tmp/initrd/init -static
find . | cpio -o -H newc > /home/deneme/initrd.img
```

Daha sonra da qemu kullanarak test edelim.

```
qemu-system-x86_86 --enable-kvm -kernel /boot/vmlinuz-5.17 -initrd /home/deneme/initrd.img -append "quiet" -m 512m
```

Eğer tüm adımları doğru yaptıysanız ekranda hello world yazısı ile karşılaşacaksınız. Ayrıca kendi işletim sisteminizi çalıştırmış olacaksınız.

Teorik olarak kernel ve initramfs tek başına bir işletim sistemi sayılabilir. Genel olarak linux dağıtımlarında sistemin kurulu olduğu diskte GNU sistemi bulunur ve kernel ve initramfs kullanılarak bu sistemdeki **/sbin/init** dosyası çalıştırılır. Bu dosya servis yöneticisi dosyamızdır ve sistemin geri kalanının çalışmasını sağlar.

Yukarıdaki anlatımda initramfs nasıl çalıştığından söz edildi. Şimdi ise bir linux dağıtımında kullanılması için gereken işlemler üzerinde durulacaktır. Öncelikle initramfs oluşturma dizinimize gereken modülleri eklemeliniz. Bunun için /lib/modules/xxx içerisindeki dosyaları initramfs içine kopyalayalım.

```
...
mkdir -p /tmp/initrd/lib/modules/
for directory in {crypto,fs,lib} \
    drivers/{block,ata,md,firewire} \
    drivers/{scsi,message,pcmcia,virtio} \
    drivers/usb/{host,storage}; do
    find /lib/modules/$(uname -r)/kernel/${directory}/ -type f \
        -exec install {} /tmp/initrd/lib/modules/$(uname -r)/ \;
done
depmod --all --basedir=/tmp/initrd
...
```

Yukarıdaki örnekte ilk önce gerekli modülleri kopyaladık ve ardından modüllerin listesini güncellemek için **depmod** komutunu kullandık. Bu modülleri yüklemek için /init dosyamız içinde **modprobe** komutunu kullanabiliriz. Bu dosya genellikle **busybox ash** kullanılarak yazılır. Bunun için öncelikle busybox dosyamız initramfs dizinine kopyalanır. Ve ardından sembolik bağları atılarak komutları kullanılabilir hale getirilir.

```
...
mkdir -p /tmp/initrd/bin
install /bin/busybox /tmp/initrd/busybox
chroot /tmp/initrd /busybox --install -s /bin
...
```

Burada eğer busybox static olarak derlenmemişse çalışmayacağı için **glibc** ve gereken diğer dosyalarımızı da eklememiz gerekmektedir. Bunun için önce **ldd** komutu ile bağımlılıkları öğrenilir ve bağımlılık dosyası initramfs dizininde /lib içine yerleştirilir. Bu işlem tüm alt

bağımlılıklarda tekrarlarır. Aşağıdaki örnekte bağımlılıkların bulunması ve kopyalanması için bir fonksiyon oluşturulmuştur.

```
...
function get_lib(){
    ldd $1 | cut -f3 -d" " | while read lib ; do
        if [[ "$lib" == "" ]] ; then
            : empty line
        elif ! echo ${libs[@]} | grep $lib >/dev/null; then
            echo $lib
            get_lib $lib
        fi
    done | sort | uniq
}
function install_binary(){
    get_lib $1 | while read lib ; do
        file=/tmp/initrd/lib/$(basename $lib)
        if [[ ! -f $file ]] ; then
            install $lib $file
        fi
    done
    install $1 /tmp/initrd/bin/$(basename $1)
}
mkdir -p /tmp/initrd/lib/
ln -s lib /tmp/initrd/lib64
install_binary /bin/busybox
...
```

Eğer Bazı dağıtımlarda /lib64 bulunur. Bu sebeple lib64 adında bir sembolik bağ oluşturmamız gerekebilir.

Modülleri yüklemek için elle **modprobe** komutu kullanılabilir. Bu sayede initramfs dosyamıza eklediğimiz modüllerin tamamını yükleyip donanımları tanıması sağlanabilir.

```
...
find /lib/modules/$(uname -r)/ -type f | while read module ; do
    module_name=$(basename "$module"| sed "s/\..*//g")
    if echo ${module_name} | grep "debug" ; then
        : ignore debug module
    else
        modprobe ${module_name}
    fi
done
...
```

Bu işlemin dezavantajı hem yavaş çalışması hem de gerekli olmayan modüllerin de yüklenmesidir. Bu yüzden bu yöntem yerine alternatif olarak **eudev** veya **systemd-udev** kullanılabilir. Bunun için initramfs dizinimize aşağıdaki eklemeler yapılır.

```
...
# eudev için
install_binary /sbin/udevd
# systemd-udev için
install_binary /lib/systemd/systemd-udevd
# Her ikisi için
```

```
install_binary /sbin/udevadm
...
```

Daha sonra initramfs içerisindeki /init içinde aşağıdaki komutlar çalıştırılmalıdır.

```
...
# systemd-udev için
systemd-udevd --daemon
# eudev için
udevd --daemon
# Her ikisi için
udevadm trigger -c add
udevadm settle
...
```

Eğer systemd kullanmayan bir dağıtım geliştirecekseniz veya initramfs dosyasının daha az boyutlu olmasını istiyorsanız **eudev** tercih etmelisiniz.

Initramfs dosyasının birinci amacı ana sistemi diske bağlayıp görevi servis yöneticisine devretmektir. Bu sebeple önce disk bağlanır ve ardından içerisine **/dev**, **/sys**, **/proc** dizinleri bağlanır ve **switch_root** kullanılarak ana sisteme geçilir.

```
# Eğer yoksa dev sys proc dizinlerini oluşturalım.
mkdir -p /dev /sys /proc
# dev sys proc bağlayalım
mount -t devtmpfs devtmpfs /dev
mount -t sysfs sysfs /sys
mount -t proc proc /proc
...
# diski bağlayalım
mount $root /new_root
# dev sys proc taşıyalım
mount --move /dev /new_root/dev
mount --move /sys /new_root/sys
mount --move /proc /new_root/proc
# /dev/root oluşturalım (isteğe bağlı)
ln -s $root /new_root/dev/root
# servis yöneticisini çalıştıralım.
exec switch_root /new_root $init
```

Yukarıdaki örnekte **$root** ve **$init** değişkenleri değerini /proc/cmdline içerisinden okumalısınız. varsayılan init değeri **/sbin/init** olmalıdır.

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.

kmod Nedir? Nasıl Yazılır ve Kullanılır?

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.

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

```
ln -s kmod /bin/depmod
ln -s kmod /bin/insmod
ln -s kmod /initrd/bin/lsmod
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.

hello.c dosyamız

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

Makefile dosyamız

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

modülün derlenmesi ve eklenip kaldırılması

```
make

insmod my_modul.ko // modül kernele eklendi.

lsmod | grep my_modul //modül yüklendi mi kontrol ediliyor.

rmmod my_modul // modül kernelden çıkartılıyor.
```

Not:

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

udev Nedir? Niçin Kullanılır?

systemd-udevd, Linux sistemlerinde donanım aygıtlarının eşleştirilmesi ve yönetimi için kullanılan bir sistem hizmetidir. Bu hizmet, udev adı verilen bir alt sistem üzerinde çalışır ve donanım olaylarını izler, aygıt dosyalarını oluşturur ve aygıtların durumunu günceller.

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.

systemd-udevd, donanım olaylarını izler ve bu olaylara göre belirli eylemler gerçekleştirir. Örneğin, bir USB cihazı takıldığında veya çıkarıldığında, systemd-udevd bu olayı algılar ve ilgili aygıt dosyasını oluşturur veya kaldırır. Ayrıca, donanım aygıtlarının durumunu güncellemek için de kullanılır. Örneğin, bir ağ arabirimi devre dışı bırakıldığında, systemd-udevd bu durumu algılar ve ilgili aygıt dosyasını günceller.

systemd-udevd, Linux sistemlerinde donanım aygıtlarının dinamik olarak yönetilmesini sağlayarak sistem yöneticilerine büyük bir esneklik ve kolaylık sağlar. Bu hizmet, donanım aygıtlarının otomatik olarak algılanmasını ve yapılandırılmasını sağlar, böylece kullanıcılar yeni bir aygıt takıldığında veya çıkarıldığında manuel olarak müdahale etmek zorunda kalmazlar.

Wine Ayarları

initramfs

initramfs dosyası kernel 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. initramfs dosyası üretmek için öncelikle bir dizin oluşturulur. Paketlenen dosya gzip veya lzma ile sıkmıştırılabilir veya sıkıştırılmadan da kullanılabilir. Bu dizine gerekli dosyalar eklenir ve aşağıdaki gibi paketlenir.

```
cd /initrd/dizini/
find . | cpio -o -H newc | gzip -9 > /boot/initrd.img-xxx
```

Kernel initrd dosyasını ram üzerine yükler ve içerisindeki /init dosyasını çalıştırır.

Örneğin aşağıdaki gibi bir C dosyamız olsun.

```
#include <stdio.h>
int main(){
    printf("Hello World!\n");
    while(1); // programın bitmesini engellemek için
    return 0;
}
```

Bu dosyayı static olarak derleyelim ve initramfs dosyasının içine koyup paketleyelim.

```
mkdir /tmp/initrd
cd /tmp/initrd
gcc -o /home/deneme/main.c /tmp/initrd/init -static
find . | cpio -o -H newc > /home/deneme/initrd.img
```

Daha sonra da qemu kullanarak test edelim.

```
qemu-system-x86_86 --enable-kvm -kernel /boot/vmlinuz-5.17 -initrd /home/deneme/initrd.img -append "quiet" -m 512m
```

Eğer tüm adımları doğru yaptıysanız ekranda hello world yazısı ile karşılaşacaksınız. Ayrıca kendi işletim sisteminizi çalıştırmış olacaksınız.

Teorik olarak kernel ve initramfs tek başına bir işletim sistemi sayılabilir. Genel olarak linux dağıtımlarında sistemin kurulu olduğu diskte GNU sistemi bulunur ve kernel ve initramfs kullanılarak bu sistemdeki **/sbin/init** dosyası çalıştırılır. Bu dosya servis yöneticisi dosyamızdır ve sistemin geri kalanının çalışmasını sağlar.

Yukarıdaki anlatımda initramfs nasıl çalıştığından söz edildi. Şimdi ise bir linux dağıtımında kullanılması için gereken işlemler üzerinde durulacaktır. Öncelikle initramfs oluşturma

dizinimize gereken modülleri eklemeliniz. Bunun için /lib/modules/xxx içerisindeki dosyaları initramfs içine kopyalayalım.

```
...
mkdir -p /tmp/initrd/lib/modules/
for directory in {crypto,fs,lib} \
    drivers/{block,ata,md,firewire} \
    drivers/{scsi,message,pcmcia,virtio} \
    drivers/usb/{host,storage}; do
    find /lib/modules/$(uname -r)/kernel/${directory}/ -type f \
        -exec install {} /tmp/initrd/lib/modules/$(uname -r)/ \;
done
depmod --all --basedir=/tmp/initrd
...
```

Yukarıdaki örnekte ilk önce gerekli modülleri kopyaladık ve ardından modüllerin listesini güncellemek için **depmod** komutunu kullandık. Bu modülleri yüklemek için /init dosyamız içinde **modprobe** komutunu kullanabiliriz. Bu dosya genellikle **busybox ash** kullanılarak yazılır. Bunun için öncelikle busybox dosyamız initramfs dizinine kopyalanır. Ve ardından sembolik bağları atılarak komutları kullanılabilir hale getirilir.

```
...
mkdir -p /tmp/initrd/bin
install /bin/busybox /tmp/initrd/busybox
chroot /tmp/initrd /busybox --install -s /bin
...
```

Burada eğer busybox static olarak derlenmemişse çalışmayacağı için **glibc** ve gereken diğer dosyalarımızı da eklememiz gerekmektedir. Bunun için önce **ldd** komutu ile bağımlılıkları öğrenilir ve bağımlılık dosyası initramfs dizininde /lib içine yerleştirilir. Bu işlem tüm alt bağımlılıklarda tekrarlarır. Aşağıdaki örnekte bağımlılıkların bulunması ve kopyalanması için bir fonksiyon oluşturulmuştur.

```
...
function get_lib(){
    ldd $1 | cut -f3 -d" " | while read lib ; do
        if [[ "$lib" == "" ]] ; then
            : empty line
        elif ! echo ${libs[@]} | grep $lib >/dev/null; then
            echo $lib
            get_lib $lib
        fi
    done | sort | uniq
}
function install_binary(){
    get_lib $1 | while read lib ; do
        file=/tmp/initrd/lib/$(basename $lib)
        if [[ ! -f $file ]] ; then
            install $lib $file
        fi
    done
    install $1 /tmp/initrd/bin/$(basename $1)
}
mkdir -p /tmp/initrd/lib/
```

```
ln -s lib /tmp/initrd/lib64
install_binary /bin/busybox
...
```

Eğer Bazı dağıtımlarda /lib64 bulunur. Bu sebeple lib64 adında bir sembolik bağ oluşturmamız gerekebilir.

Modülleri yüklemek için elle **modprobe** komutu kullanılabilir. Bu sayede initramfs dosyamıza eklediğimiz modüllerin tamamını yükleyip donanımları tanıması sağlanabilir.

```
...
find /lib/modules/$(uname -r)/ -type f | while read module ; do
    module_name=$(basename "$module"| sed "s/\..*//g")
    if echo ${module_name} | grep "debug" ; then
        : ignore debug module
    else
        modprobe ${module_name}
    fi
done
...
```

Bu işlemin dezavantajı hem yavaş çalışması hem de gerekli olmayan modüllerin de yüklenmesidir. Bu yüzden bu yöntem yerine alternatif olarak **eudev** veya **systemd-udev** kullanılabilir. Bunun için initramfs dizinimize aşağıdaki eklemeler yapılır.

```
...
# eudev için
install_binary /sbin/udevd
# systemd-udev için
install_binary /lib/systemd/systemd-udevd
# Her ikisi için
install_binary /sbin/udevadm
...
```

Daha sonra initramfs içerisindeki /init içinde aşağıdaki komutlar çalıştırılmalıdır.

```
...
# systemd-udev için
systemd-udevd --daemon
# eudev için
udevd --daemon
# Her ikisi için
udevadm trigger -c add
udevadm settle
...
```

Eğer systemd kullanmayan bir dağıtım geliştirecekseniz veya initramfs dosyasının daha az boyutlu olmasını istiyorsanız **eudev** tercih etmelisiniz.

Initramfs dosyasının birinci amacı ana sistemi diske bağlayıp görevi servis yöneticisine devretmektir. Bu sebeple önce disk bağlanır ve ardından içerisine **/dev**, **/sys**, **/proc** dizinleri bağlanır ve **switch_root** kullanılarak ana sisteme geçilir.

```
# Eğer yoksa dev sys proc dizinlerini oluşturalım.
mkdir -p /dev /sys /proc
# dev sys proc bağlayalım
mount -t devtmpfs devtmpfs /dev
mount -t sysfs sysfs /sys
mount -t proc proc /proc
...
# diski bağlayalım
mount $root /new_root
# dev sys proc taşıyalım
mount --move /dev /new_root/dev
mount --move /sys /new_root/sys
mount --move /proc /new_root/proc
# /dev/root oluşturalım (isteğe bağlı)
ln -s $root /new_root/dev/root
# servis yöneticisini çalıştıralım.
exec switch_root /new_root $init
```

Yukarıdaki örnekte **$root** ve **$init** değişkenleri değerini /proc/cmdline içerisinden okumalısınız. varsayılan init değeri **/sbin/init** olmalıdır.

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.

kmod Nedir? Nasıl Yazılır ve Kullanılır?

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.

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

```
ln -s kmod /bin/depmod
ln -s kmod /bin/insmod
ln -s kmod /initrd/bin/lsmod
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.

hello.c dosyamız

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

Makefile dosyamız

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

modülün derlenmesi ve eklenip kaldırılması

```
make

insmod my_modul.ko // modül kernele eklendi.

lsmod | grep my_modul //modül yüklendi mi kontrol ediliyor.

rmmod my_modul // modül kernelden çıkartılıyor.
```

Not:

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

udev Nedir? Niçin Kullanılır?

systemd-udevd, Linux sistemlerinde donanım aygıtlarının eşleştirilmesi ve yönetimi için kullanılan bir sistem hizmetidir. Bu hizmet, udev adı verilen bir alt sistem üzerinde çalışır ve donanım olaylarını izler, aygıt dosyalarını oluşturur ve aygıtların durumunu günceller.

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.

systemd-udevd, donanım olaylarını izler ve bu olaylara göre belirli eylemler gerçekleştirir. Örneğin, bir USB cihazı takıldığında veya çıkarıldığında, systemd-udevd bu olayı algılar ve ilgili aygıt dosyasını oluşturur veya kaldırır. Ayrıca, donanım aygıtlarının durumunu güncellemek için de kullanılır. Örneğin, bir ağ arabirimi devre dışı bırakıldığında, systemd-udevd bu durumu algılar ve ilgili aygıt dosyasını günceller.

systemd-udevd, Linux sistemlerinde donanım aygıtlarının dinamik olarak yönetilmesini sağlayarak sistem yöneticilerine büyük bir esneklik ve kolaylık sağlar. Bu hizmet, donanım aygıtlarının otomatik olarak algılanmasını ve yapılandırılmasını sağlar, böylece kullanıcılar yeni bir aygıt takıldığında veya çıkarıldığında manuel olarak müdahale etmek zorunda kalmazlar.

Yazıcı Ayarları

initramfs

initramfs dosyası kernel 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. initramfs dosyası üretmek için öncelikle bir dizin oluşturulur. Paketlenen dosya gzip veya lzma ile sıkmıştırılabilir veya sıkıştırılmadan da kullanılabilir. Bu dizine gerekli dosyalar eklenir ve aşağıdaki gibi paketlenir.

```
cd /initrd/dizini/
find . | cpio -o -H newc | gzip -9 > /boot/initrd.img-xxx
```

Kernel initrd dosyasını ram üzerine yükler ve içerisindeki /init dosyasını çalıştırır.

Örneğin aşağıdaki gibi bir C dosyamız olsun.

```
#include <stdio.h>
int main(){
    printf("Hello World!\n");
    while(1); // programın bitmesini engellemek için
    return 0;
}
```

Bu dosyayı static olarak derleyelim ve initramfs dosyasının içine koyup paketleyelim.

```
mkdir /tmp/initrd
cd /tmp/initrd
gcc -o /home/deneme/main.c /tmp/initrd/init -static
find . | cpio -o -H newc > /home/deneme/initrd.img
```

Daha sonra da qemu kullanarak test edelim.

```
qemu-system-x86_86 --enable-kvm -kernel /boot/vmlinuz-5.17 -initrd /home/deneme/initrd.img -append "quiet" -m 512m
```

Eğer tüm adımları doğru yaptıysanız ekranda hello world yazısı ile karşılaşacaksınız. Ayrıca kendi işletim sisteminizi çalıştırmış olacaksınız.

Teorik olarak kernel ve initramfs tek başına bir işletim sistemi sayılabilir. Genel olarak linux dağıtımlarında sistemin kurulu olduğu diskte GNU sistemi bulunur ve kernel ve initramfs kullanılarak bu sistemdeki **/sbin/init** dosyası çalıştırılır. Bu dosya servis yöneticisi dosyamızdır ve sistemin geri kalanının çalışmasını sağlar.

Yukarıdaki anlatımda initramfs nasıl çalıştığından söz edildi. Şimdi ise bir linux dağıtımında kullanılması için gereken işlemler üzerinde durulacaktır. Öncelikle initramfs oluşturma dizinimize gereken modülleri eklemeliniz. Bunun için /lib/modules/xxx içerisindeki dosyaları initramfs içine kopyalayalım.

```
...
mkdir -p /tmp/initrd/lib/modules/
for directory in {crypto,fs,lib} \
    drivers/{block,ata,md,firewire} \
    drivers/{scsi,message,pcmcia,virtio} \
    drivers/usb/{host,storage}; do
    find /lib/modules/$(uname -r)/kernel/${directory}/ -type f \
        -exec install {} /tmp/initrd/lib/modules/$(uname -r)/ \;
done
depmod --all --basedir=/tmp/initrd
...
```

Yukarıdaki örnekte ilk önce gerekli modülleri kopyaladık ve ardından modüllerin listesini güncellemek için **depmod** komutunu kullandık. Bu modülleri yüklemek için /init dosyamız içinde **modprobe** komutunu kullanabiliriz. Bu dosya genellikle **busybox ash** kullanılarak yazılır. Bunun için öncelikle busybox dosyamız initramfs dizinine kopyalanır. Ve ardından sembolik bağları atılarak komutları kullanılabilir hale getirilir.

```
...
mkdir -p /tmp/initrd/bin
install /bin/busybox /tmp/initrd/busybox
chroot /tmp/initrd /busybox --install -s /bin
...
```

Burada eğer busybox static olarak derlenmemişse çalışmayacağı için **glibc** ve gereken diğer dosyalarımızı da eklememiz gerekmektedir. Bunun için önce **ldd** komutu ile bağımlılıkları öğrenilir ve bağımlılık dosyası initramfs dizininde /lib içine yerleştirilir. Bu işlem tüm alt bağımlılıklarda tekrarlarır. Aşağıdaki örnekte bağımlılıkların bulunması ve kopyalanması için bir fonksiyon oluşturulmuştur.

```
...
function get_lib(){
    ldd $1 | cut -f3 -d" " | while read lib ; do
        if [[ "$lib" == "" ]] ; then
            : empty line
        elif ! echo ${libs[@]} | grep $lib >/dev/null; then
            echo $lib
            get_lib $lib
        fi
    done | sort | uniq
}
function install_binary(){
    get_lib $1 | while read lib ; do
        file=/tmp/initrd/lib/$(basename $lib)
        if [[ ! -f $file ]] ; then
            install $lib $file
        fi
    done
    install $1 /tmp/initrd/bin/$(basename $1)
}
mkdir -p /tmp/initrd/lib/
ln -s lib /tmp/initrd/lib64
install_binary /bin/busybox
...
```

Eğer Bazı dağıtımlarda /lib64 bulunur. Bu sebeple lib64 adında bir sembolik bağ oluşturmamız gerekebilir.

Modülleri yüklemek için elle **modprobe** komutu kullanılabilir. Bu sayede initramfs dosyamıza eklediğimiz modüllerin tamamını yükleyip donanımları tanıması sağlanabilir.

```
...
find /lib/modules/$(uname -r)/ -type f | while read module ; do
    module_name=$(basename "$module"| sed "s/\..*//g")
    if echo ${module_name} | grep "debug" ; then
        : ignore debug module
    else
        modprobe ${module_name}
    fi
done
...
```

Bu işlemin dezavantajı hem yavaş çalışması hem de gerekli olmayan modüllerin de yüklenmesidir. Bu yüzden bu yöntem yerine alternatif olarak **eudev** veya **systemd-udev** kullanılabilir. Bunun için initramfs dizinimize aşağıdaki eklemeler yapılır.

```
...
# eudev için
install_binary /sbin/udevd
# systemd-udev için
install_binary /lib/systemd/systemd-udevd
# Her ikisi için
install_binary /sbin/udevadm
...
```

Daha sonra initramfs içerisindeki /init içinde aşağıdaki komutlar çalıştırılmalıdır.

```
...
# systemd-udev için
systemd-udevd --daemon
# eudev için
udevd --daemon
# Her ikisi için
udevadm trigger -c add
udevadm settle
...
```

Eğer systemd kullanmayan bir dağıtım geliştirecekseniz veya initramfs dosyasının daha az boyutlu olmasını istiyorsanız **eudev** tercih etmelisiniz.

Initramfs dosyasının birinci amacı ana sistemi diske bağlayıp görevi servis yöneticisine devretmektir. Bu sebeple önce disk bağlanır ve ardından içerisine **/dev**, **/sys**, **/proc** dizinleri bağlanır ve **switch_root** kullanılarak ana sisteme geçilir.

```
# Eğer yoksa dev sys proc dizinlerini oluşturalım.
mkdir -p /dev /sys /proc
# dev sys proc bağlayalım
mount -t devtmpfs devtmpfs /dev
mount -t sysfs sysfs /sys
mount -t proc proc /proc
...
# diski bağlayalım
mount $root /new_root
# dev sys proc taşıyalım
mount --move /dev /new_root/dev
mount --move /sys /new_root/sys
mount --move /proc /new_root/proc
# /dev/root oluşturalım (isteğe bağlı)
ln -s $root /new_root/dev/root
# servis yöneticisini çalıştıralım.
exec switch_root /new_root $init
```

Yukarıdaki örnekte **$root** ve **$init** değişkenleri değerini /proc/cmdline içerisinden okumalısınız. varsayılan init değeri **/sbin/init** olmalıdır.

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.

kmod Nedir? Nasıl Yazılır ve Kullanılır?

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.

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

```
ln -s kmod /bin/depmod
ln -s kmod /bin/insmod
ln -s kmod /initrd/bin/lsmod
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.

hello.c dosyamız

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

Makefile dosyamız

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

modülün derlenmesi ve eklenip kaldırılması

```
make

insmod my_modul.ko // modül kernele eklendi.

lsmod | grep my_modul //modül yüklendi mi kontrol ediliyor.

rmmod my_modul // modül kernelden çıkartılıyor.
```

Not:

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