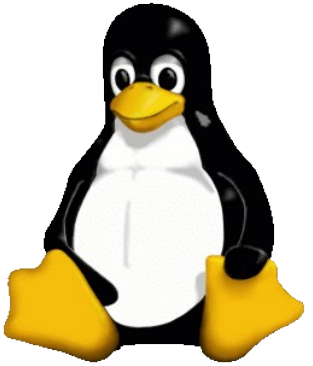
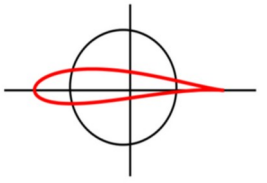
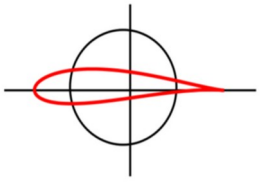
Gömülü Linux Sistemleri Lab. Çalışması



Nâzım KOÇ
https://UCanLinux.com
training@ucanlinux.com



Telif Hakları

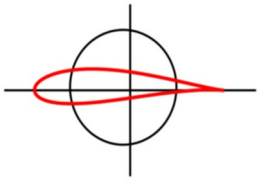


Amaçlar

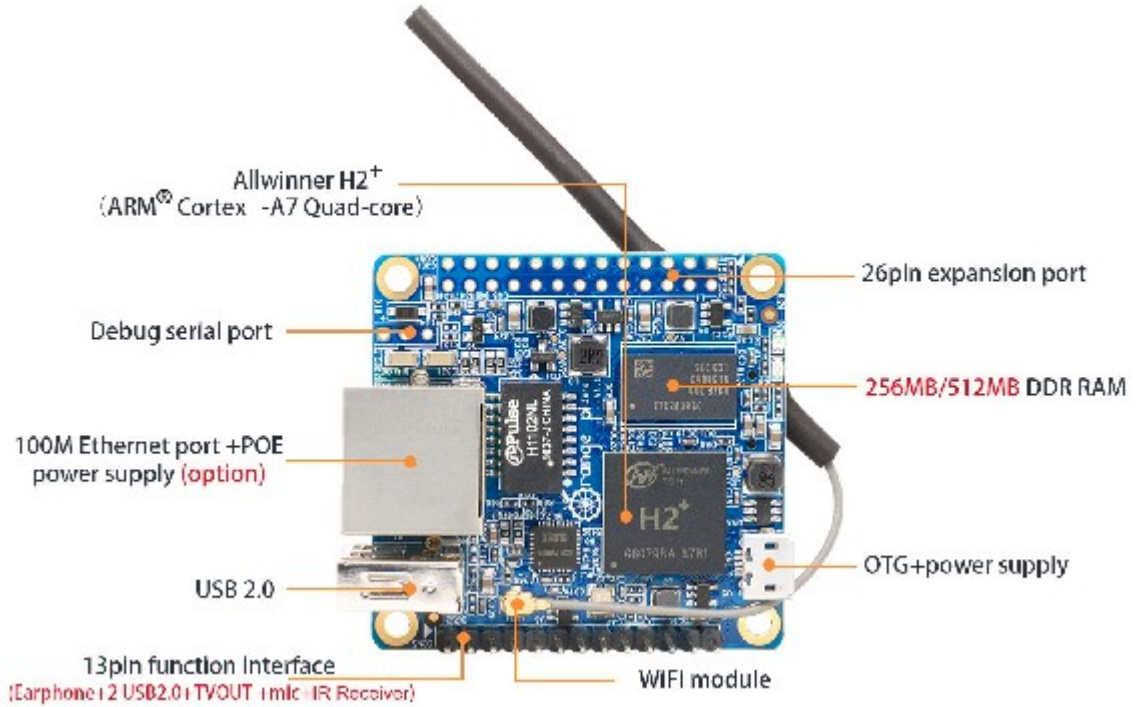
PowerOn ile Login arasındaki bütün adımları
FARKINDA OLARAK yapmak.

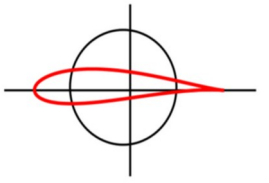
Projeye uygun gömülü sistemin seçilmesini sağlamak.

Sistematik olarak bir gömülü sistemin kurulmasını sağlamak.



Test Cihazı, Orange Pi Zero





Test Cihazı

Pi0 sadece test amacı ile kullanılacaktır.

Eğitimin Pi0 ile doğrudan ilgisi yoktur. Anlatılacak konular makineden bağımsızdır.

Donanım ile ilgili konulardan bahsedilmeyecektir. Eğitim tamamen yazılım tabanlıdır.

Bahsedilecek bütün konular qemu emülatörü ile de test edilebilir.

Temeller

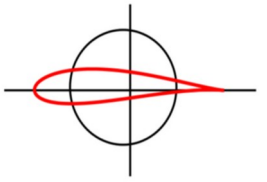
Linux sistemleri 4 temel üzerine kurulur.

Boot Loader:
   u-boot

Kernel:
   Linux

Root File System:
   Busybox, buildroot

Boot Scripts:
   handwritten, buildroot



Yardımcı Yazılımlar

Hiç bir yardımcı betik, program veya tool kullanılmayacak,

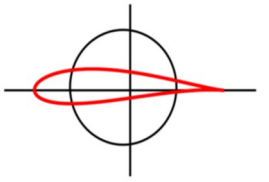
Bütün kuruluşlar standard Linux komutları ile yapılacak,

Hiç bir Linux dağıtımına bağlı kalınmayacaktır.

64 Bit Ubuntu 22.04 LTS sürümü, emülatör altında kullanılacaktır.

Aşağıdaki paketler yüklü olmalıdır.

```
$ sudo apt install -y gcc g++ tree make libncurses-dev
$ sudo apt install -y net-tools nfs-kernel-server
$ sudo apt install -y minicom tftpd tftp xinetd unzip vim
$ sudo apt install -y file bison flex swig python3-dev
$ sudo apt install -y libssl-dev gnutls-dev man-db
```

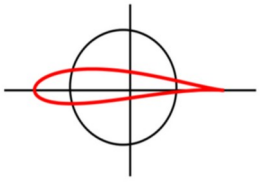
Gömülü Sistemin Karakteristiği

Çekirdeğin ve Kök Dosya Sisteminin oturduğu yer sistemin karakteristiğini verir.

Çekirdek:
- Network
- SD/MMC
- eMMC

RootFS:
- RAM (çekirdeğe gömülü)
- SD/MMC
- eMMC
- NFS

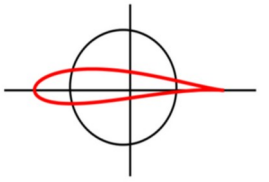
Not:
eMMC hariç bütün sistemlerden birer örnek yapılacaktır.
eMMC üzerinde kuruluş, teorik olarak verilecektir.



Ön Hazırlık

Terminalden girilen komutlar ve ekran görüntüleri otomatik olarak aşağıdaki gibi saklanabilir. Her açılan terminal için bu komut bir kez girilmelidir.

```
$ script -a -f ilk_gun

# -a : append
# -f : flush after each write, optional
#
# Gün sonuna kadar çalışma yapılır.
# Sonra exit komutu ile script kapatılır.

$ exit

# Bütün ekran görüntüleri artık ilk_gun dosyası içindedir.
# Bu dosya metin tabanlıdır, cat/more/vi/less gibi komutlarla incelenebilir.
# Sıkıştırılıp saklanabilir.

$ more ilk_gun

# sudo komutlarında, sürekli şifre girmemek için, şifreyi sil.

$ passwd -d can

# tekrar şifre vermek için

$ passwd can
```



Dizin Yapısının Kurulması

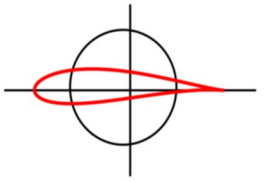
```
# eğitim için gerekli araç gereçler aşağıdaki gibi /opt altına kurulur
#

$ cd /opt

$ sudo tar xvf ~/orangepi0_ucanlinux.tgz

$ id
uid=1000(student) gid=1000(student) ...

$ sudo chown -R student:student gomsis  # gerekliyse

$ cd gomsis  # bu adımdan sonra asla root ile çalışma!

$ ls -l
```

Paketler

gomsis/ftp/oku dosyasında, her paketin nasıl indirileceği mevcuttur, vakitten kazanmak için gerekli paketler önceden indirilmiştir.

Dizin Yapısı

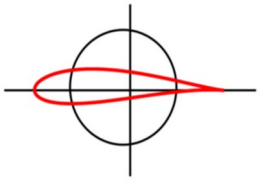
Her proje için, sonraki yansıda verildiği gibi, bir çalışma dizini kurulmalıdır. Bu dizin eksiksiz bütün projeyi kapsamalıdır.



Dizin Yapısı ve Çalışma Ortamı

```
$ cd /opt
$ tree -L 2 gomsis
```

```
gomsis
├── RootFS.skel
│   ├── bin
│   ├── dev
│   ├── etc
│   ├── init -> /sbin/init
│   ├── lib
│   ├── mnt -> /tmp
│   ├── opt
│   ├── proc
│   ├── root
│   ├── sbin
│   ├── sys
│   ├── tmp
│   ├── usr
│   └── var
├── configs
│   ├── 1
│   ├── 2
│   ├── 3
│   ├── 4
│   └── buildroot
├── ftp
│   ├── br.tgz
│   ├── busybox.bz2
│   ├── linux.zip
│   ├── oku
│   ├── toolchain.zip
│   ├── u-boot.zip
│   └── zlib.tar.gz
...
```



Günlük Çalışma Planı

1. gün
   Çekirdek ve kök dosya sistemi SD kartta olan sistemle açılış.

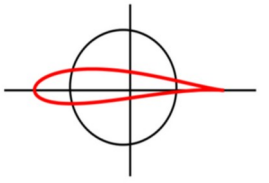
2. gün
   Çekirdek SD kart içinde, kök dosya sistemi çekirdeğin içinde olan açılış.

   Çekirdek SD kart içinde, kök dosya sistemi SD içinde ve arşivlenmiş açılış.

   Çekirdek ağ üzerinde, kök dosya sistemi NFS üzerinde iken açılış.

3. gün
   Build Root ile kök dosya sistemi kuruluşu, açılış betiklerinin incelenmesi.

   Sorular ve kapanış.



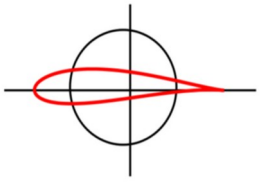
Çapraz Derleyiciler

Kuruluş

```
$ cd /opt/gomsis

$ unzip ftp/toolchain.zip

$ ln -s toolchain-master/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabihf arm

$ ls -l

$ ls -l arm/bin
```

PATH adlı çevre değişkenine gcc'nin bulunduğu dizin eklenir.
Başka derleyiciler ile karışmaması için, ilgili dizin PATH'ın önüne eklenmelidir.
Editor olarak pico, nano, gedit de kullanılabilir.

```
$ vi ~/.bashrc  # dosyanın en dibine ekle
export PATH=/opt/gomsis/arm/bin:$PATH

kaydedip çıkmak için "ESC :wq"   # en çok sorulan soru!

Mevcut terminalden çık, tekrar terminal aç ki .bashrc'deki atamalar işlesin.
$ exit
```

Test

```
$ arm-linux-  <TAB> <TAB>
```

Bütün toolchain komutları listelenmeldir.
Listelenmezse PATH hatalıdır ya da yeni terminal açılmamış olabilir.

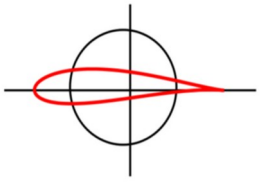
Aynı ön-eke sahip, birden fazla toolchain varsa, bunlar birbirlerine karışabilir.
Emin olmak için which ile kurulan toolcahin'in kullanımda olup olmadığı mutlaka kontrol edilmelidir.

```
$ which arm-linux-gnueabihf-gcc
/opt/gomsis/toolchain/arm/bin/arm-linux-gnueabihf-gcc
```

Toolchain'in, apt komutları ile kurulması tavsiye edilmez.
Yukarıda verilen yöntemle aynı anda aynı toolchain'in farklı sürümleri sorunsuzca kurulup kullanılabilir.

Sadece PATH değişkeninin güncellenmesi yeterli olacaktır

Bu toolchain ile, x86 üzerinde, ARM makine için açılış yükleyicisi, kernel, busybox gibi programları derleyebiliriz.



Çalışma

/opt/gomsis/test/hello.c programı üzerinde çalış.

native derle.

boy ve lib bağımlılıklarını incele, file ile incele.

ufalt.

statik derle.

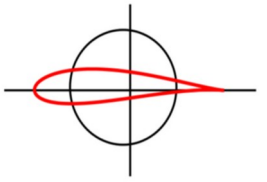
boy ve lib bağımlılıklarına bak.

arm- <tab> <tab> ile kuruluşu incele.

which ile tam yerini bul.

yukarıdaki bütün adımları cross derleme ile yap.

CROSS_COMPILE ile derle.



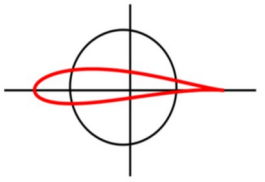
Açılış Yükleyicisi

Kuruluş

```
$ cd /opt/gomsis

$ unzip ftp/u-boot.zip

$ ln -s u-boot-master u-boot

$ ls -l u-boot
lrwxrwxrwx 1 student student 13 Dec 29 07:20 u-boot -> u-boot-master

$ cd u-boot

$ make orangepi_zero_defconfig

$ more .config

$ make menuconfig
```

Yapılandırma ekranı geldiğinde, şimdilik aşağıda verilen temel güncellemeler yapılmalıdır.

Sonra ustalaştıkça u-boot parameterleri üzerinde oynama yapılabilir.



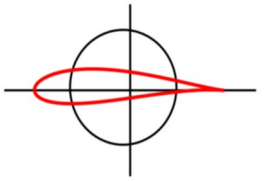
```
(5) delay in seconds before automatically booting


Command line interface --->
   (UCanLinux > ) Shell prompt
```

Shell prompt kısmına herkes kendi adını yazabilir.

```
Environment  --->
   [*] Environment is in a FAT filesystem
   (mmc) Name of the block device for the environment
   (0:auto) Device and partition for where to store the environemt in FAT
   (uboot.env) Name of the FAT file to use for the environment
```

<exit> ile çıkılır.

Sonraki adımda U-Boot çapraz derlenecektir.



Derleyici düzgün yerde mi?

```
$ which arm-linux-gnueabihf-gcc
/opt/gomsis/arm/bin/arm-linux-gnueabihf-gcc


$ make CROSS_COMPILE=arm-linux-gnueabihf- -j$(nproc)
$ echo $?      # başarılı mı bitmiş?
```

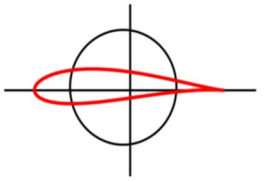
Üretilen bazı dosyaları incele.

```
$ ls -l spl/sunxi-spl.bin u-boot.img u-boot-sunxi-with-spl.bin

-rw-rw-r-- 1 student student  24576 Dec 29 07:24 spl/sunxi-spl.bin
-rw-rw-r-- 1 student student 535872 Dec 29 07:24 u-boot-sunxi-with-spl.bin
-rw-rw-r-- 1 student student 503104 Dec 29 07:24 u-boot.img
```

Buraya gelindiğinde açılış yükleyicileri artık elde edilmiştir.

sunxi-spl.bin: 1. açılış yükleyicisi (first stage boot loader)

u-boot.img   : 2. açılış yükleyicisi (second stage boot loader)



Açılış Sırası

ROM Boot Loader --> sunxi-spl.bin --> u-boot.img --> kernel

Sistemin açılması için spl ve u-boot kodlarının arka arkaya, 8K'dan sonraya kopyalanması yeterlidir (orangepi için).

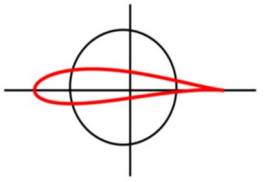
Bu özellik sadece OrangePI ailesine aittir.
Genelde her bordun farklı bir yükleme tekniği vardır.

Açılış Yükleyicilerinin Kopyalanması

Açılış yükleyicileri aşağıdaki gibi açılış sektörlerine kopyalanır

SD kart notebook'a takılır.
df ile bakılır.
Eğer kart otomatik olarak mount edilmiş ise *mutlaka unmount* yapılmalıdır.
$ dmesg komutu ile kartın cihaz adına bakılır.

```
$ dmesg | tail
  mmc0: new high speed SDHC card at address 1234
  mmcblk0: mmc0:1234 SA08G 7.21 GiB
  mmcblk0: p1 p2
```

Verilen örnekte mmcblk0 bizim SD kartımızın adıdır.
Kullanılan donanıma göre farklı isimler gelebilir.
/dev/sdb veya /dev/sdc gibi cihaz ismleri de gelebilir.

Aşağıdaki komut ile ilk 1023 sektör sıfırlanır.

```
$ df  # mount edilmiş olmamalı!

$ sudo dd if=/dev/zero of=/dev/mmcblk0 bs=1k count=1023 seek=1
```

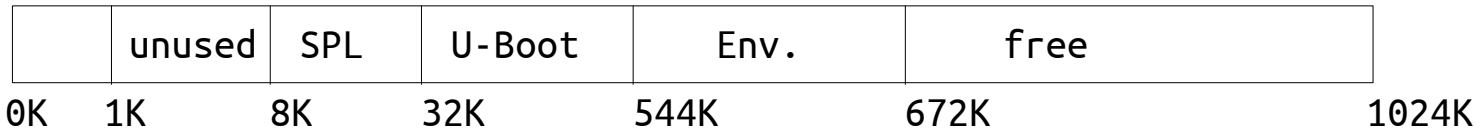
seek=1 ile MBR + devamındaki 512 bayta, yani toplamda 1K'lık alana dokunulmaz.

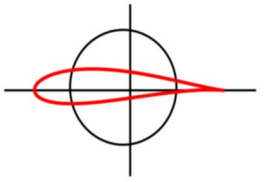
!!! DİKKAT !!!

Cihaz ismi hatalı ise mevcut makinenizi kaybedebilirsiniz.

dmesg komutu ile veya başka bir yol ile takılan cihazın ismi eksiksiz olarak tespit edilmelidir

```
$ cd /opt/gomsis/u-boot
$ sudo dd if=u-boot-sunxi-with-spl.bin of=/dev/mmcblk0 bs=1024 seek=8
$ sync  # MUTLAKA!!!
```





Seri Kablo ile Bağlantı

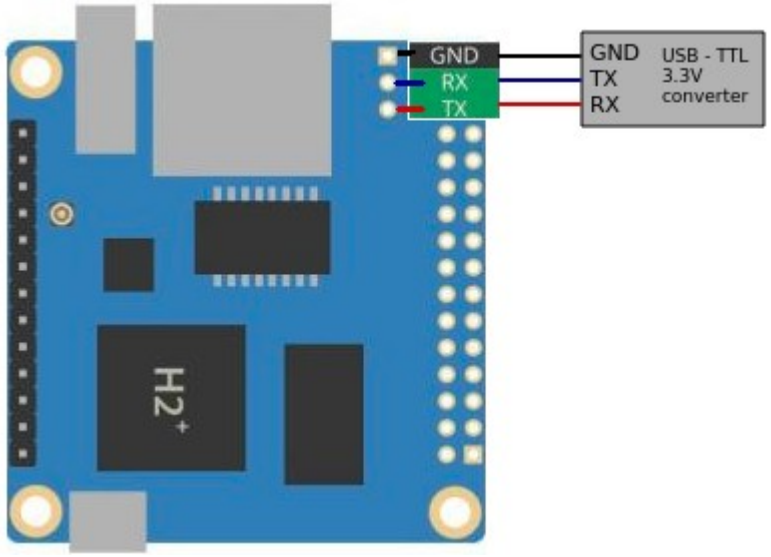
Debug Kablosu Bağlantısı.

Kablonun bir ucu boşta iken notebook'a takılır.
Seri/USB çeviricinin çekirdek tarafındaki adı öğrenilir.

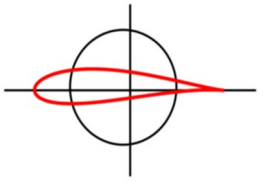
```
$ dmesg|tail
...
usb 3-2: pl2303 converter now attached to ttyUSB0

$ sudo minicom -c on -w -D /dev/ttyUSB0 -b 115200
```

Kablonun diğer uçları sonraki şekildeki gibi borda takılır.
Power kablosu asla takılmamalıdır.

OrangePi makinesi USB ile beslenir.



beyaz TX
yeşil RX
siyah GND



U-Boot Açılış Testi

SD kart borda takılır.

Debug kablosunun her iki ucu da boştadır.
Önce, debug kablosunun bir ucu, bord'a takılır, GND, TX RX.

Sonra diğer ucu, notebook'a takılır.
minicom açılır, kablonun diğer ucu şu anda boştadır.

USB C koblosu borda taklır.
En sona USB kablosunun diğer ucu PC'ye takılır.

Sistem açılır ve mesajlar akmaya başlar.
5 sn içinde boşluğa basılırsa, açılış durur ve u-boot içine düşülür.

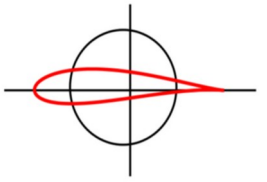
Bu ortam basit bir kabuk programıdır.

Gömülü sistem projelerinde en kritik aşama u-boot promptuna gelebilmektir.

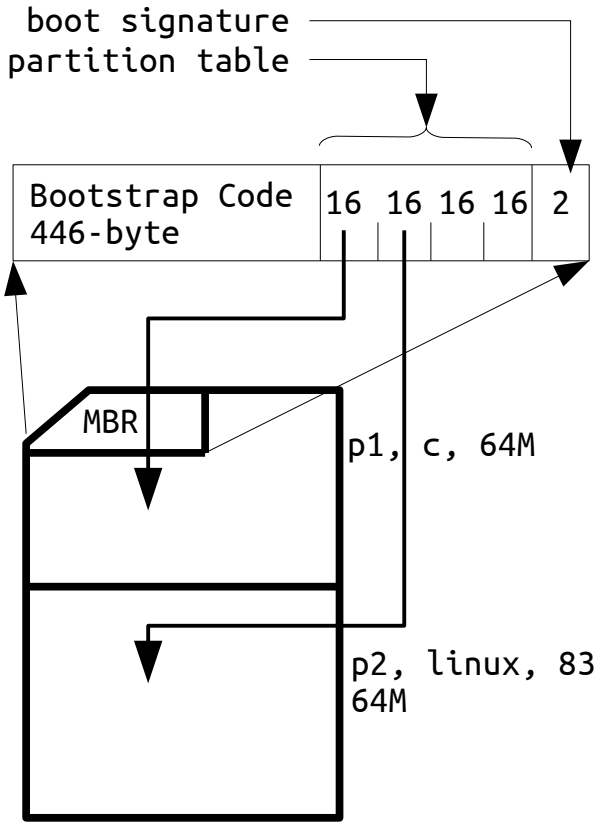
Yedekleme

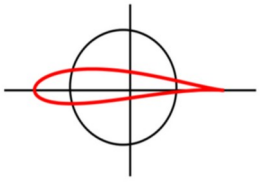
.config yedeği mutlaka alınmalıdır.

```
$ cd /opt/gomsis
$ cp u-boot/.config configs/1/u-boot.config
```

SD Kartın Bölümlere Ayrılması, DOS Part.





SD/MMC kart ile çalışmak

SD kart takılır, df ile bakılır, bağlı ise umount ile koparılır, sonra...

```
$ dmesg|tail
...
[  367.695871] mmcblk0: mmc0:1234 SA04G 3.64 GiB
[  367.719918]  mmcblk0: p1 p2

$ sudo fdisk /dev/mmcblk0
```

64M vfat ve 64M'lik ext2 bölümleri yaratılır, sonuç aşağıdaki gibi olmalıdır.

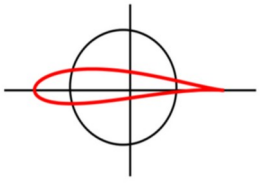
```
$ sudo fdisk /dev/mmcblk0 -l

Device         Boot  Start    End Sectors Size Id Type
/dev/mmcblk0p1        2048 133119  131072  64M  c W95 FAT32 (LBA)
/dev/mmcblk0p2      133120 264191  131072  64M 83 Linux
```

fdisk sadece MBR kısmını günceller.
İlk 1M'lık alana boot programları içindir.
Daha önce buraya 1. ve 2. ön yükleyiciler kurulmuştu.

Dosya Sistemlerinin Kurulması

```
$ sudo mkfs.vfat /dev/mmcblk0p1 -n BOOT
$ sudo mkfs.ext2 /dev/mmcblk0p2 -L ROOT
$ sync
```

-n ve -L seçeneği ile name ve volume label verilebilir, zorunlu değildir.



Linux veya Çekirdek

Bord üreticisinin önerdiği çekirdek sürümü kullanılmalıdır.
ARM çekirdekleri genelde ana çekirdek sürümünü takip etmez, geriden gelirler.

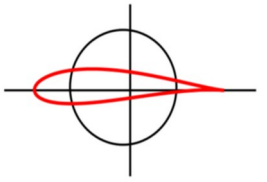
```
$ cd /opt/gomsis

$ unzip ftp/linux.zip

$ ln -s linux-orangepi-orange-pi-5.4 linux

$ cd linux
```

Altın kural
Bütün Makefile işlemleri her zaman kaynak kodun kökünde yapılır.
Burada kaynak kodu kökü yani başlangıç dizini linux/'tur.

```
$ make ARCH=arm help
```

defconfig: default config file
Üreticiler tarafından kurulan, hazır yapılandırma dosyasıdır.

```
$ make ARCH=arm help | grep orange

$ ls arch/arm/configs/

$ more arch/arm/configs/orangepi_defconfig
```



```
$ which arm-linux-gnueabihf-gcc
```

.config dosyasını köke kopyala

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- orangepi_defconfig

$ more .config
```

y: yes, always installed
m: loadable module

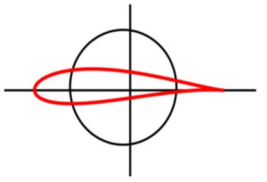
CONFIG_AD=y
AD özelliği derlenecek ve çekirdeğin çalışan kodu içine eklenecektir

CONFIG_AD=m
AD özelliği derlenecek ve derlenmiş kod, çekirdeğin dışında bir yerde, örneğin bir dosya sisteminde, /lib/modules/ altında tek bir dosya da tutulacaktır.
İhtiyaç olduğunda çekirdeğe yükelenecektir veya işi bittikten sonra çekirdekten atılabilecektir.

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

Her zaman LTS sürümleri seçilmelidir.

Örnek çekirdek 5.4.x
long term support, 6 years,
released: 2019-11-24
Projected End Of Life: Dec, 2025



Her zaman 2 config dosyası üretilmesi tavsiye edilir.

Biri development/test sırasında kullanmak, diğeri ise production ortamı içindir.

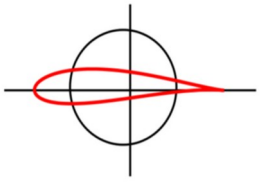
Seçimler

```
[y/n]       yes/no, in kernel
<y/n/m>     yes/no/module, in kernel or loadable module
-*-         selected as in kernel by other options
{M}         selected as loadable module by other options
```

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage -j$(nproc)
...
Kernel: arch/arm/boot/zImage is ready

$ ls -l arch/arm/boot/zImage
-rwxrwxr-x 1 student student 8311384 Dec 29 08:01 arch/arm/boot/zImage

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- modules -j2
```

Örnek olması için 2 modül eklenmişti.

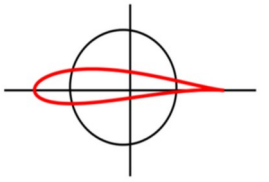
```
$ find . -name "*.ko"
./kernel/configs.ko
./fs/minix/minix.ko
```

DTB

DTB, cihaz bilgilerinin tutulduğu, hiyerarşik veri yapısıdır. Prensip olarak XML gösterimi ile aynı mantığa sahiptir. Cihaz verileri ağaç şeklinde gösterilir. Çekirdek ihtiyacı olan veriyi, ağaç yapısı üzerinden bulur.
Örnek, Pi0 için kullanılan DTS dosyasından,

```
leds{
     compatible = "gpio-leds";

      pwr_led {
            label = "orangepi:green:pwr";
            gpios = <&r_pio 0 10 GPIO_ACTIVE_HIGH>;
            default-state = "on";
      };

      status_led {
            label = "orangepi:red:status";
            gpios = <&pio 0 17 GPIO_ACTIVE_HIGH>;
      };
};

<leds>  # sezgisel bir XML gösterimi, gerçek değildir, sadece fikir vermesi içindir
     <compatible>gpio-leds</compatiable>

      <pwr_led>
            <label orangepi:green:pwr/>
            <gpios &r_pio 0 10 GPIO_ACTIVE_HIGH/>
            <default-state on/>;
      </pwr_led>

      <status_led>
            <label orangepi:red:status/>
            <gpios gpio 0 17 GPIO_ACTIVE_HIGH/>
      </staus_led>
</leds>
```

Çekirdek, hiyerarşik veriyi of_*() fonksiyonları ile elde eder.

Çekirdek içindeki bir uygulama örneği,

```
struct device_node *led_node;

led_node = of_parse_phandle(np, "leds", 0);

if (!led_node) {
        pr_err("No led node found\n");
        return -EINVAL;
}
```

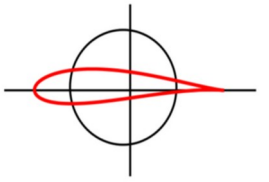
of_*(): open firmware functions

Device Tree Kernel API
https://www.kernel.org/doc/html/latest/devicetree/kernel-api.html

---

DTS, DTB, DTC

hello.c --> gcc --> hello
foo.dts --> dtc --> foo.dtb



Çekirdek derlenmesi ile zImage dosyası elde edilmişti. Ayrıca cihaz bilgilerini barındıran DTS dosyasının da, çekirdeğin anlayabileceği DTS dosyasına çevrilmesi gerekir.

Örnek dtb'yi incele.

```
$ more arch/arm/boot/dts/sun8i-h2-plus-orangepi-zero.dts
```

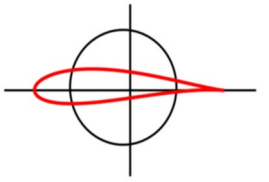
Bütün dts'leri, ayrım yapmadan derle.

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- dtbs
```

```
$ file arch/arm/boot/dts/sun8i-h2-plus-orangepi-zero.dts
arch/arm/boot/dts/sun8i-h2-plus-orangepi-zero.dts: C source, ASCII text
```

```
$ file arch/arm/boot/dts/sun8i-h2-plus-orangepi-zero.dtb
arch/arm/boot/dts/sun8i-h2-plus-orangepi-zero.dtb: Device Tree Blob version 17,
size=30537, boot CPU=0, string block size=2521, DT structure block size=27960
```

```
$ ls -l arch/arm/boot/dts/sun8i-h2-plus-orangepi-zero.dtb
-rw-rw-r-- 1 nazim nazim 30537 Mar 24 21:12 arch/arm/boot/dts/sun8i-h2-plus-
orangepi-zero.dtb
```

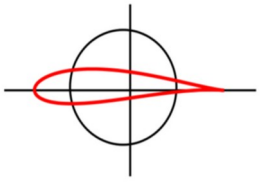



Üretilen Dosyalar

zImage	Linux çekirdeği
*.ko	Yüklenebilir modüller
dtb	Cihaz tanımları

Çekirdeği test etmek için zImage ve dtb dosyasları “/” veya “/boot” veya tamamen keyfi bir dizin ve keyfi bir disk bölümüne kopyalanabilir. Bizler birinci bölümde “/” altına kopyalayacağız. SD kart host makineye takılır.

```
$ sudo mkdir /mnt/boot

$ sudo mount /dev/mmcblk0p1 /mnt/boot

$ df /mnt/boot

$ cd /opt/gomsis/linux

$ sudo cp arch/arm/boot/zImage /mnt/boot

$ sudo cp arch/arm/boot/dts/sun8i-h2-plus-orangepi-zero.dtb /mnt/boot/dtb

$ ls -l /mnt/boot
total 8148
-rwxr-xr-x 1 root root   30537 Dec 29 08:22 dtb
-rwxr-xr-x 1 root root 8311384 Dec 29 08:22 zImage

$ sudo umount /mnt/boot
```

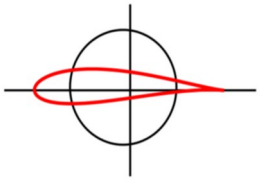
Çekirdeğin Test Edilmesi

SD kart borda takılır, bard açılır ve u-boot seviyesinde açılış durdurlur.

```
$ minicom -c on -w -D /dev/ttyUSB0 -b 115200
```

```
> help
> mmc list
> mmc info
> mmc part

> help ls
> ls mmc 0
> ls mmc 0:1
> ls mmc 0:2
```

```
mmc 0 : birinci mmc
mmc 1 : ikinci mmc

mmc 0:1 -> /dev/mmcblk0p1
mmc 0:2 -> /dev/mmcblk0p2

":" den sonra bölüm numarası yazılır.
Yazılmazsa 1 kabul edilir
```

```
> bdinfo
...
boot_params = 0x40000100   # start+ 256
-> start    = 0x40000000
-> size     = 0x10000000   # 256M
```

bdinfo çıkışındaki start adresi, fiziksel RAM adresidir ve çok önemlidir.
Bu adres genelde her bordda farklıdır ve çekirdek, initramfs ile dtb'nin yükleneceği adreslerin tespit edilmesinde kullanılır.

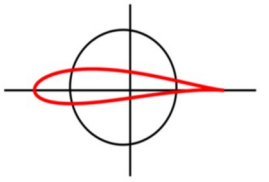
Çekirdeğin Yüklenmesi

Disk bölümü fat formatlandığı için fatload kullanılacaktır.
ext4 formatlansaydı ext4load kullanılacaktı, vs.

```
$ fatload mmc 0 0x46000000 zImage
```

Görüleceği üzere, çekirdeğin oturduğu dizin ve çekirdeğin adının bir önemi yoktur.
Dizin ve çekirdeğin fatload komutunda verilmesi yeterlidir.

Pek çok sistemde boot/ dizini kullanılır.
Basit olması için biz kullanmadık.



0x46000000 adresinin de pek bir önemi yoktur.

Bu adresine, bdinfo'daki "-> start" adresinden sonra, ortalarda bir yerde olması ve dosyanın tamamının fiziksel adres uzayı içine bulunması yeterlidir.

fatload ile verilen adresler başlangıç noktasına yakın olmamalıdır.
Çünkü açılış sırasında başlangıç adresine yakın alanlar, kernel tarafından işgal edilecektir.

DTB'nin Yüklenmesi

```
> fatload mmc 0 0x49000000 dtb
```

Buradaki 0x49000000 adresinin de bir önemi yoktur.
Daha önce yüklenen zImage ile bu dosyanın, birbirlerini ezmemeleri yeterlidir.
Tabii ki dtb de, verilen adres uzayı içinde kalmalıdır.

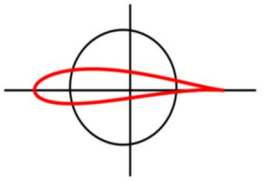
Çekirdeğe Parametre Aktarımı

```
> setenv bootargs console=ttyS0,115200
```

Genel olarak a=b ataması yapmak için

```
> setenv a b
```

ifadesi kullanılır.



u-boot, bootargs değişkeninin sağ tarafını çekirdeğe aktarır.
Bunun için de bdinfo çıkışında verilen boot params = 0x40000100 adresini kullanır.

Bu adrese, bootargs'ın sağında olan ifadeleri bağlı liste şeklinde yazar.
Çekirdek de bu adresi bildiği için kendine gelen parametreleri açılış sırasında bu adresten okur.

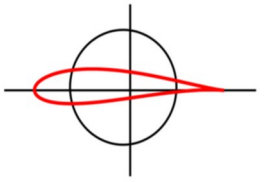
Genelde bu adres start adresinden sonraki 0x100 baytı kabul edilir ve bdinfo çıkışında boot_params değişkeni ile verilir.

Çekirdeği Yükle ve Başlat

bootz kernelAddr initrdAddr dtbAddr
boot Linux zImage stored in memory

Örnekte initrd yoktur ve adres yerine "-" girilmiştir.

```
> bootz 0x46000000 - 0x49000000

Starting kernel ...
...
Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
```



bootz komutu önce bootargs ile verilmiş parametreleri boot_params adresine yazar.

Sonra dtb'nin adresini çekirdeğe bildirir ve yürütmeyi çekirdeğe verir.

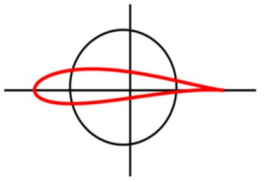
Çekirdek kök dosya sistemini bulamadığı için panikler.

Bu aşamaya kadar gömülü Linux sisteminin, açılış yükleyicisi ve çekirdek ayakları tamamlanmıştır. Geriye kök dosya sistemi ve açılış betiği ayakları kalmıştır.

Yedekleme

Çekirdeğin config dosyası mutlaka saklanmalıdır.

```
$ cd /opt/gomsis/linux
$ cp .config ../configs/1/kernel.config
```

Kök Dosya Sistemi

Dosya sistemleri hiyerarşisinde, en tepede olan dosya sistemine kök dosya sistemi denir.

Diğer bir deyişle, çekirdeğin açılışta bağladığı dosya sistemidir.

Çekirdek bu dosya sistemini bağladıktan sonra sıra ile aşağıdaki programları çalıştırmayı dener.

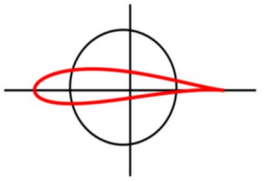
```
/sbin/init
/etc/init
/bin/init
/bin/sh
```

Kuruluşu

Kök dosya sisteminin kuruluşu iki aşamada gerçeklenir.

Önce içi hemen hemen boş bir dizin hiyerarşisi yaratılır.
Bu dizin hiyeraşisine **iskelet dosya sistemi** denir.

Hemen hemen boş olan bu dizinler, busybox, toolchain ve kernel'dan alınan dosyalar ile doldurulur.



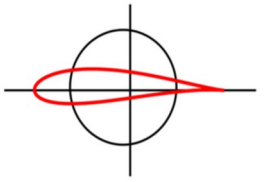
```
$ cd /opt/gomsis

$ cp -a RootFS.skel RootFS

$ du -ks RootFS
148   RootFS

$ tree RootFS
```

Ya da el ile teker teker kuruluş yapılır.

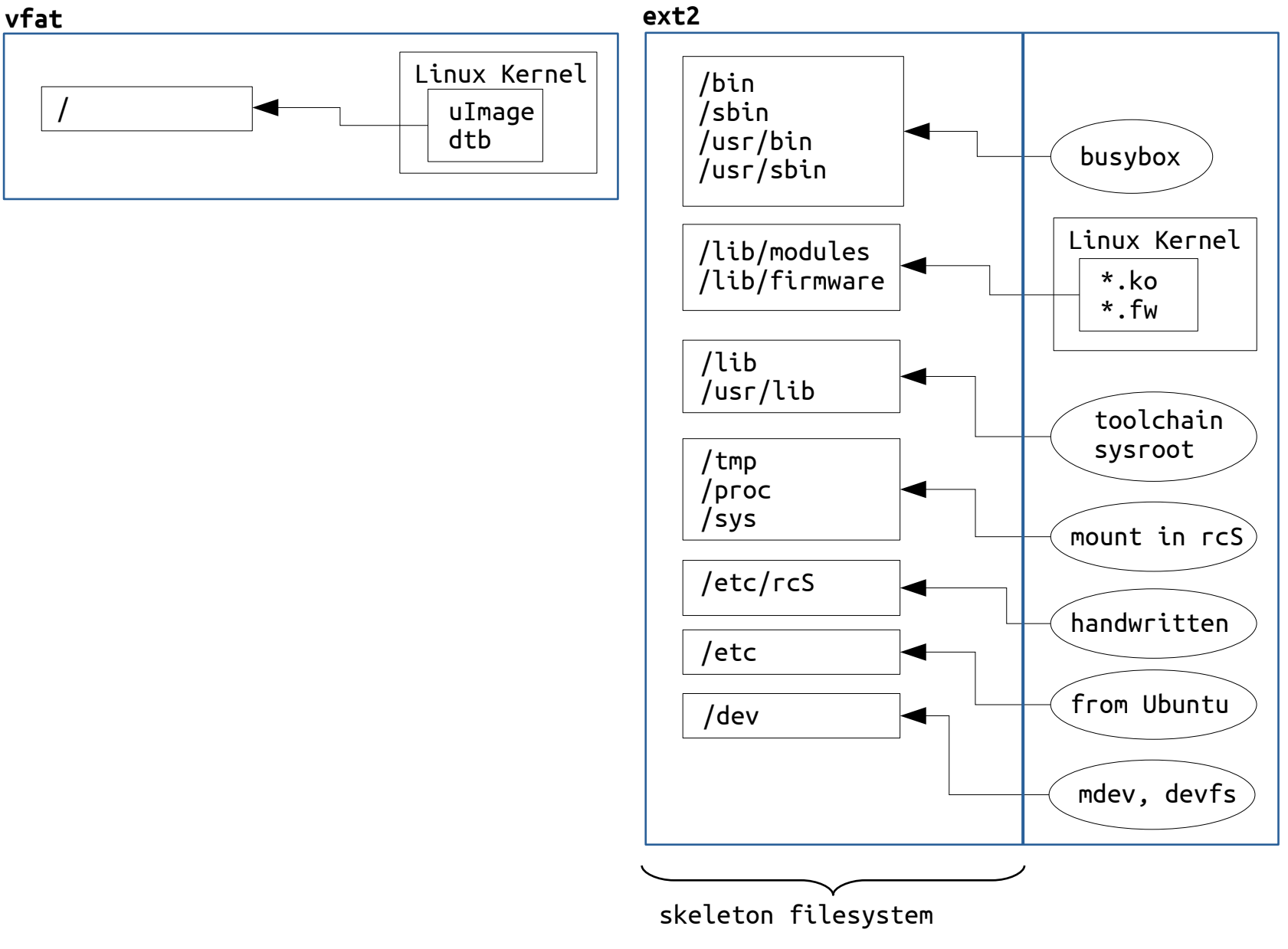
```
$ cd /opt/gomsis

$ mkdir RootFS

$ cd RootFS

$ mkdir bin boot proc root dev etc sbin sys lib tmp mnt usr opt var
```

Ayrıca etc/ altına da birkaç örnek dosya kopyalanmalıdır.
Bunlar herhangi bir Linux dağıtımından kopyalanabilir.

Bu dosyaların hepsi zorunlu değildir.
Örnek dosyalar aşağıda verilmiştir.



/opt/gomsis/RootFS/



/etc

fstab
mount edilecek dosya sistemleri hakkında bilgi barındırır. İçi boştur.

group
Sistemdeki grup tanımları.

hostname
Sistemin adı. Keyfidir, biz UCanLinux yazdık.

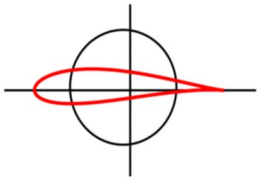
hosts
IP ve hostname bilgileri. Sadece localhost tanımı vardır.

inittab
Açılış yönlendiren dosya, /sbin/init içindir.

issue
Login öncesi reklam içindir.

mtab
./proc/self/mounts için sembolik linktir.

nsswitch.conf
Çeşitli hizmetlerin kaynağının adresi ve sırasını barındırır.



passwd
Kullanıcı bilgilerini barındırır.

profile
Kabuğa girmeden evvel çalışacak dosya.

protocols
İnternet protokollerinin isimleri ve port numaraları.

rcS
Açılış betiği. /etc/inittab tarafından tetiklenir.

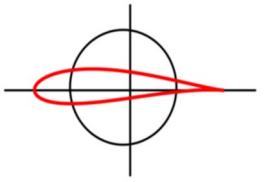
resolv.conf
DNS içindir, ../tmp/resolv.conf için sembolik linktir.

services
İnternet servislerinin isimleri ve numaralarını tutar.

shadow
Kullanıcı şifrelerini ve sürelerini saklar.

udhcpc.script
Otomatik IP alındıktan sonra çalışan betiktir.

```
$ man 5 services
```

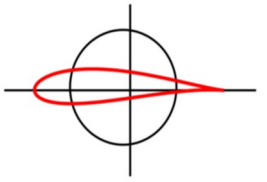



Busybox

/bin, /sbin, /usr/bin ve /usr/sbin dizinleri altında Linux komutları bulunur. Bu komutlar pek çok yerden temin edilebilir.Örnek sistemimizde bu komutlar busybox sistemi yardımı ile üretilecektir.

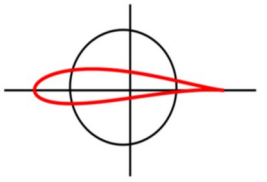
Busybox projesinde pek çok komut seçenekleri azaltılarak yeniden yazılmıştır. Yeniden yazılan bu komutlara applet denir.

```
int mkdir_main(int argc, char *argv[]){
   // mkdir komutunun işlevi burada gerçeklenir
   return res;
}

int ls_main(int argc, char *argv[]){
   // ls komutunun işlevi burada gerçeklenir
   return res;
}

int cd_main(int argc, char *argv[]){
   // cd komutunun işlevi burada gerçeklenir
   return res;
}
...
```

Busybox

```
// busybox sisteminin ana girişi.

int main(int argc, char *argv[]){
   // sembolik link ile kullanım
   // ls -> busybox
   //  ^
   //  argv[0]

   if ( argv[0] == "ls")     return ls_main(argc, argv);
   if ( argv[0] == "mkdir" ) return mkdir_main(argc, argv);
   if ( argv[0] == "cd")     return cd_main(argc, argv);
   ...
   if ( argv[0] == "busybox"){
      // doğrudan kullanım
      // argv[1]'e göre apletler seçilir
      // busybox ls -l gibi
      //         ^
      //        argv[1]
      }
   ...
   err("applet not found");
}
```

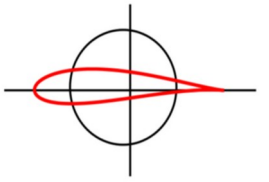



Zorunlu olmamakla birlikte, kullanım kolaylığı için sembolik linkler kullanılır.

```
$ ln -s busybox mkdir

$ ln -s busybox ls

$ ln -s busybox cd

$ ls -l
... mkdir -> busybox
... ls -> busybox
... cd -> busybox
```

Kuruluş

```
$ cd /opt/gomsis

$ tar -xvf ftp/busybox.bz2

$ ln -s busybox-1.36.1 busybox

$ ls -l busybox
```

Applet'lerin Seçimi

```
$ cd /opt/gomsis/busybox

$ make menuconfig
```

İhtiyaç olan applet'ler menuconfig ile seçilir.

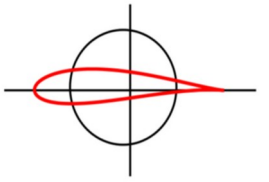
Test/Development ortamları için bütün applet'lerin seçilmesi yararlı olacaktır. Testlerin daha konforlu bir alanda yapılmasını sağlar.

```
$ more .config
```

Production ortamı için, sadece gerekli appletler seçilmelidir.

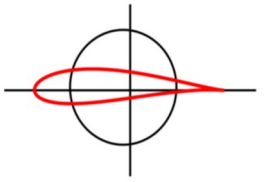
Gerekli applet'ler nasıl tespit edilir?

- /etc betiğindeki komutlar
- init appleti
- exec, system veya popen gibi sistem çağrıları ile kullanılan komutlar
- ...



Derleme

```
$ cd /opt/gomsis/busybox

$ make CROSS_COMPILE=arm-linux-gnueabihf-

$ ls -l busybox
-rwxrwxr-x 1 student student 760968 Dec 29 13:57 busybox


$ ls -l /usr/bin/vim.basic  /usr/bin/top

-rwxr-xr-x 1 root root  133184 Oct 31  2023 /usr/bin/top
-rwxr-xr-x 1 root root 3787824 Nov  7 13:17 /usr/bin/vim.basic



$ file busybox
busybox: ELF 32-bit LSB shared object,
         ARM,
         EABI5 version 1 (SYSV),
         dynamically linked,
         interpreter /lib/ld-linux-armhf.so.3,
         BuildID[sha1]=3579a7c6ce867f38feddea20aebf4d750fe1673e,
         for GNU/Linux 3.2.0,
         stripped
```

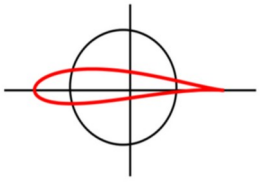
Kurulum

```
$ make CROSS_COMPILE=arm-linux-gnueabihf- install

$ cd _install

$ ls -l

$ cd bin

$ ls
$ ls -l
$ ls -l | wc -l
```

Benzer şekilde sbin ve diğer dizinler incelenmelidir.
İskelet dosya sisteminde /bin, /sbin, /usr/bin, /usr/sbin dizinleri busybox tarafıdan doldurulacaktır.

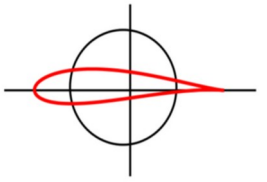
Üretilen applet sayısı

```
$ cd /opt/gomsis/busybox/_install

$ find . -type l | wc -l
402

$ tree .
```



/bin, /sbin, /usr/bin, /usr/sbin

```
$ cd /opt/gomsis
$ cp -a busybox/_install/*   RootFS/
$ rm RootFS/linuxrc
$ tree RootFS
```

/lib

Kütüphaneler toolcahin'den elde edilir.

Bütün toolchain paketleri içinde sysroot denilen özel bir dizin yapısı vardır.

"include dosyaları" ve "kütüphaneleri" barındıran dizine **sysroot** denir.

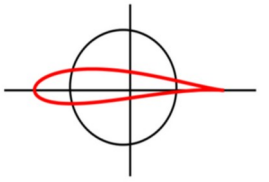
Kütüphaneler sysroot'tan alınacaktır.
sysroot'un yerini öğrenmek için,

```
$ arm-linux-gnueabihf-gcc --print-sysroot
/opt/gomsis/toolchain-master/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-
gnueabihf/bin/../arm-linux-gnueabihf/libc
```

Bu çok uzun ad, kolay kullanım için **SYSROOT** değişkenine atanır.
Bu değişken adı keyfidir.

```
$ SYSROOT=$(arm-linux-gnueabihf-gcc --print-sysroot)
```

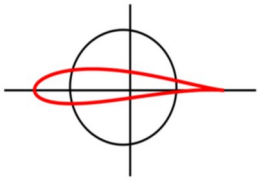
Bu değişkenin adı keyfidir.



```
$ cd /opt/gomsis

# Zorunlu kütüphaneler.
$ cp $SYSROOT/lib/libc.so.6             RootFS/lib/
$ cp $SYSROOT/lib/libdl.so.2            RootFS/lib/
$ cp $SYSROOT/lib/ld-linux-armhf.so.3   RootFS/lib/
# Seçimlik kütüphaneler.
$ cp $SYSROOT/lib/libm.so.6             RootFS/lib/
$ cp $SYSROOT/lib/libnss_files.so.2     RootFS/lib/
$ cp $SYSROOT/lib/libnss_dns.so.2       RootFS/lib/
$ cp $SYSROOT/lib/libresolv.so.2        RootFS/lib/
```

Zorunlu kütüphaneler eksikse init programı çalışmaz ve login gelmez.

Bazen busybox için, seçilen apletlere göre libm.so kütüphanesi de gerekli olabilir.



libc.so
kütüphanesi standard C kütüphanesidir.

libdl.so
kütüphanesi dinamik kütüphanelerin yürütme zamanında yüklenmesini sağlayan kütüphanedir.

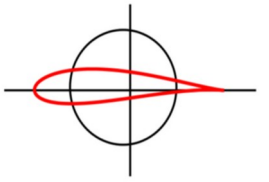
ld-linux-armhf.so
kütüphanesine yorumlayıcı da denir.
Bu kütüphanenin görevi, yürütme zamanında dinamik kütüphaneleri yüklemek, tablo ve adresleri atamaktır.
Derlenmiş busybox kodu **$ file busybox** komutu ile incelenirse "interpreter /lib/ld-linux-armhf.so.3" şeklinde bir çıkış görülebilir.

Eğer busybox, statik derlenirse bu kütüphanelerin hiç birine gerek kalmaz. lib/ dizinine dahi gerek yoktur. Çok özel çalışamalar hariç bu tür bir uygulama tavsiye edilmez.

Busybox'da seçilen apletlere veya uygulama programlarının ihtiyaçlarına göre gerekli kütüphaneler kopyalanmalıdır.

Altın Kural

Kütüphane kopyalama yaparken, kütüphane toolchain'de nerede oturuyorsa, RootFS'de de tam aynı yerde oturması daha sonra işleri çok kolaylaştırır.



/lib/modules

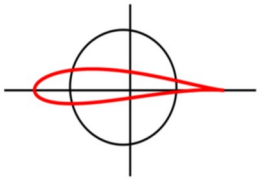
```
$ cd /opt/gomsis/linux

$ find . -name "*.ko"
```

Kopyalama

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- \
       INSTALL_MOD_PATH=../RootFS                  \
       modules_install
```

İnceleme

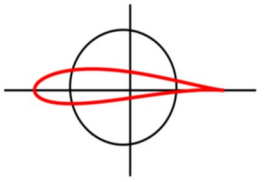
```
$ cd /opt/gomsis/RootFS/lib && ls

$ cd modules && ls

$ cd 5.4.65 && ls

$ more modules.dep

$ cd kernel/fs/minix && ls

$ file minix.ko
```



```
$ modinfo minix.ko
filename:       /opt/gomsis/RootFS/lib/modules/5.4.65/kernel/fs/minix/minix.ko
license:        GPL
alias:          fs-minix
depends:
intree:         Y
name:           minix
vermagic:       5.4.65 SMP mod_unload ARMv7 p2v8
```

Örnek modules.dep satırı

```
kernel/drivers/net/wireless/admtek/adm8211.ko:
     kernel/net/mac80211/mac80211.ko
     kernel/net/wireless/cfg80211.ko
     kernel/drivers/misc/eeprom/eeprom_93cx6.ko
     kernel/lib/crypto/libarc4.ko
```

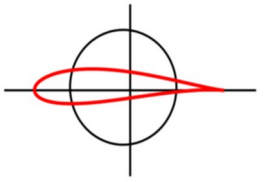
Örnek modules.aliases satırları

alias fs-minix minix
alias usb:v*p*d*dc*dsc*dp*ic08isc04ip00in* usb_storage

make modules install komutu bütün modülleri, modül bağımlılıklarını vs.
bulur ve RootFS/lib/modules/<Çekirdek Sürümü>/ dizini altına kopyalar.

Dizin adı olarak çekirdek sürümünün kullanımı zorunludur.
Böylece sistemde birden fazla çekirdek varsa, modüller karışmaz.

Eğer aynı çekirdeğin farklı derlenmiş modülleri varsa, çekirdekte,
“make menuconfig” işlemi sırasında, çekirdeğe son-ek adı verilir,
-test1, -test2 gibi. Böylece ilgili dizin,
RootFS/lib/modules/<Çekirdek Sürümü-test1>/ şeklinde yaratılacaktır.



/lib/firmware

Cihaza ait donanım yazılımları, yani firmware varsa, aşağıdaki gibi bu firmware dosyaları /lib/firwmare altında kopyalanmalıdır.

```
$ cd /opt/gomsis/linux

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- \
            INSTALL_FW_PATH=../RootFS/lib/firmware
            firmware_install
```

Firmware dosyaları binary olarak çekirdek kodunda otururlar, ayrıca derlenmezler. Bu dosyalar doğrudan /lib/firmware altına kopyalanır.

Modüllerde olduğu gibi, dizin adı olarak ayrıca çekirdek sürümü kullanılmaz. Bu dosyalar kernel tarafından doğrudan cihazlara gönderilirler.

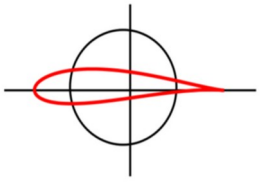
Örnek borda ait firmware yoktur.

Ubuntu x86_64 için

```
$ ls -l /lib/firmware
```

file ile incelenirse "data" dosyaları olduğu görülebilir.

Bu adımdan sonra eksik kalan dizinler çalışma zamanında, otomatik olarak doldurulacaktır.



/dev, /proc, ...

/dev, /proc, /sys, /tmp ve /var dizinleri açılış sırasında, /etc/rcS betiği tarafından veya çekirdek tarafından mount edilirler.

/dev dizininde, cihazların düğüm adları ve diğer bilgiler saklanır.

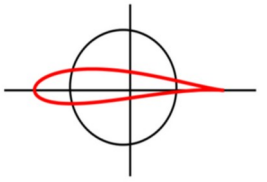
Bu dizini kurmanın çok yolu vardır.
Şu anda en yaygın olanı çekirdeğe kurdurmaktır.

Bunun için çekirdek derlemesi aşamasında aşağıdaki seçimler yapılmalıdır.

```
Device Drivers --->

 Generic Driver Options --->
    [*] Maintain a devtmpfs filesystem to mount at /dev
    [*] Automount devtmpfs at /dev, after the kernel mounted the rootfs
```

"Maintain a devtmpfs filesystem to mount at /dev" seçeneği ile çekirdek, düğüm isimlerini, tmpfs dosya sistemi içine, ram bellekte kurar.

Diğer seçenek ise bu dosya sistemini /dev altına bağlayarak, kullanıma açar.



/dev dizini için özel bir durum:

```
$ make menuconfig
```

sırasında

"[*] Automount devtmpfs at /dev, after the kernel mounted the rootfs"

seçimi initramfs tabanlı açılışlar için geçerli değildir.

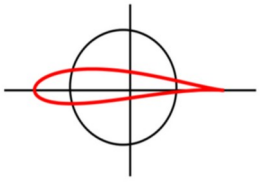
Mount işleminin /etc/rcS içinde açıkça yapılması gerekir.

Çünkü, çekirdek initramfs açılışından sonra gerçek dosya sistemine geçileceğini kabul eder ve mount işlemini yapmaz.

/proc, /sys

Açılış sırasında açıkça bağlanırlar.

```
$ mount -t proc  proc  /proc
$ mount -t sysfs sysfs /sys
```

/tmp

tmpfs tipinde bir dosya sistemidir.
Geçici dosyaların yazıldığı yerdir.
Güç kesilince içindeki bilgiler kaybolur.

```
$ mount -t tmpfs -o mode=1777 tmpfs /tmp
```

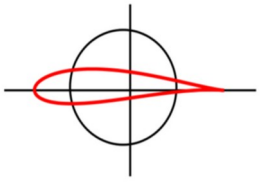
sticky bit: owner or root can delete the file.

/var

Aynı /tmp dizini gibidir.
Geçici dosyaların yazıldığı yerdir.
Güç kesilince içindeki bilgiler kaybolur.

```
$ mount -t tmpfs -o mode=0755,nosuid,nodev tmpfs /var
```

Pek çok gömülü sistemde /var dizini doğrudan /tmp dizinine sembolik olarak
Bağlanmıştır. Yani ayrıca mount edilmemiştir.



/dev/pts

devpts dosya sistemi, sözde terminal numarası üretir.
Diğer bir deyişle ssh, telnet gibi uzaktan bağlantılarda ya da GUI'lerde açılan programların kullanacağı terminallerin cihaz isimlerini üretir.

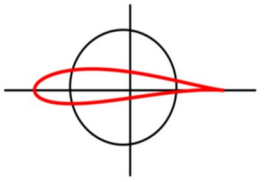
Eğer uzaktan bağlanırken login ekranı gelir gelmez program düşüyorsa, muhtemelen devpts dosya sistemi bağlı değildir.

```
$ mkdir /dev/pts
```

```
$ mount -t devpts -o gid=5,mode=620 devpts /dev/pts
```

/dev/shm

/tmp, /var gibi geçicidir.
Bazı programlar veya sistem çağrılar illa ki bu adı ararlar.
/tmp'ye link de olabilir.

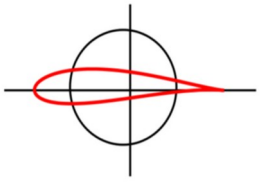


Kök dosya sistemi, notebook'da RootFS/ dizini altına artık kurulmuştur.

Açılış sırası

u-boot çekirdeği yükler,
çekirdek çalışmaya başlar,
kök dosya sistemini mount eder,
çekirdek tarafından /dev bağlanır,
/sbin/init başlatılır,
/etc/inittab'a göre açılış yönlendirilir,
login gelir.

Ön-yükleyici, Çekirdek ve Kök dosya sistemi tamamlandı.
Son ayak, açılış betikleridir.



Açılış Betikleri

/sbin/init kodunun çalıştırılabilir olması yeterlidir, ELF veya bash.
Çekirdek bu kodu gözü kapalı başlatır.

Açılışta ilk hangi programın çalışacağı, istenirse

init=/falan/hede

şeklinde, çekirdek parametresi olarak, açılışta verilebilir. Çekirdek otomatik olarak **/falan/hede** kodunu yürütecektir.

Çekirdek ilk çalışan kodun pid numarasını 1 olarak atar.

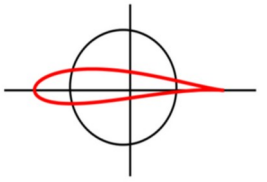
/etc/inittab

/sbin/init programı, /etc/inittab içindeki satırlara göre açılışı yönetir.

```
::sysinit:/etc/rcS
::respawn:/sbin/getty -L ttyS0 115200 vt100
::shutdown:/bin/sync
::shutdown:/bin/umount -a -r
```

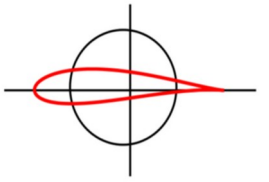
Genel yapı
id:runlevels:action:process

Gömümlü sistemlerde id ve runlevel son derece gereksizdir.
Budan dolayı, busybox'ın inittab dosyasından kaldırılmıştır.

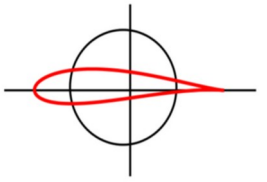


/etc/rcS

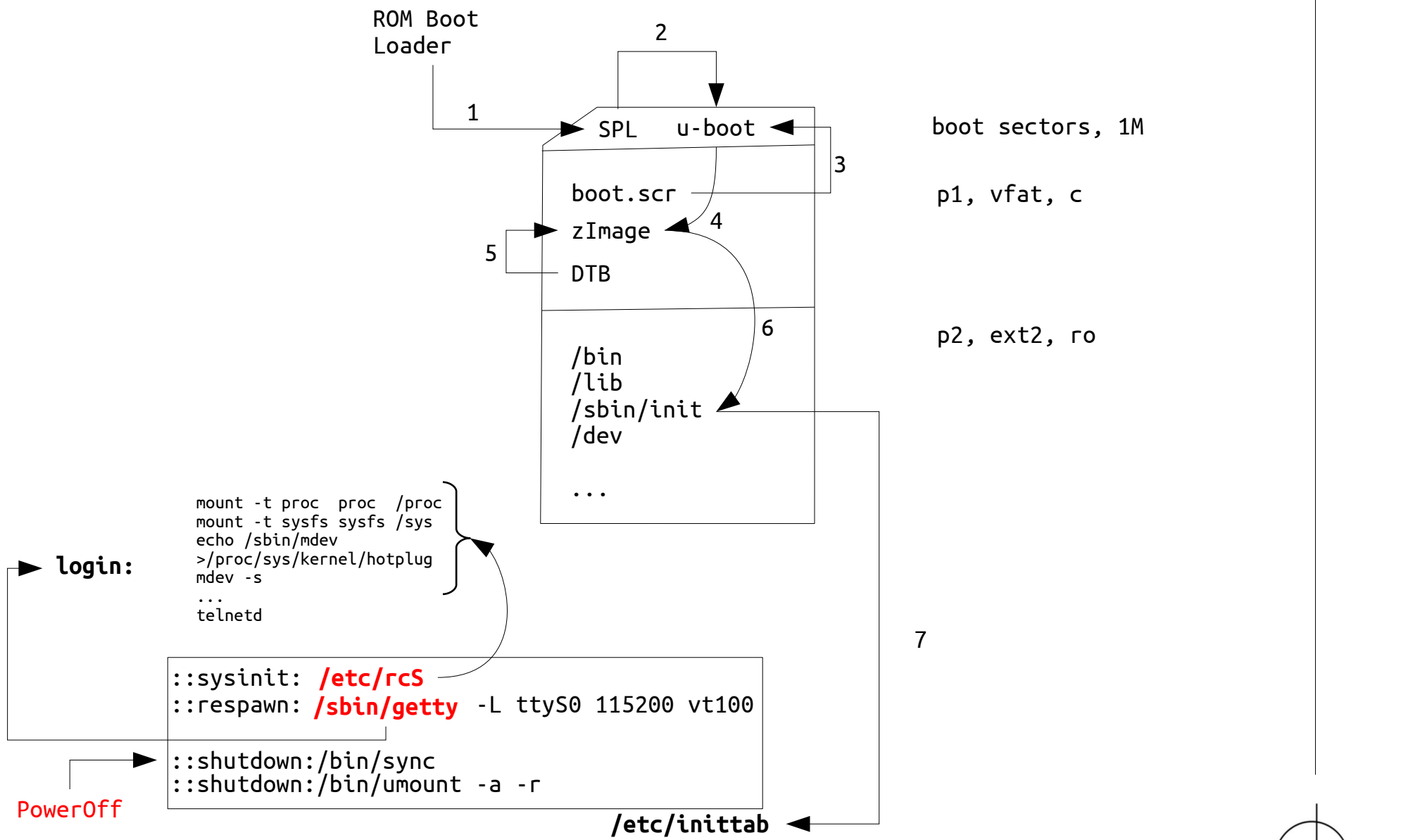
```
01 #!/bin/sh -x

02 export PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin

03 mount -t proc proc /proc
04 mount -t sysfs sysfs /sys
05 mount -t tmpfs -o mode=1777 tmpfs /tmp

06 mkdir /dev/pts
07 mount -t devpts -o gid=5,mode=620 devpts /dev/pts

08 mkdir /dev/shm
09 mount -t tmpfs -o mode=0777 tmpfs /dev/shm

10 mount -t tmpfs -o mode=0755,nosuid,nodev tmpfs /var

11 mkdir /var/cache
12 mkdir /var/lock
13 mkdir /var/log
14 mkdir /var/run
15 mkdir /var/spool
16 mkdir /var/tmp
```

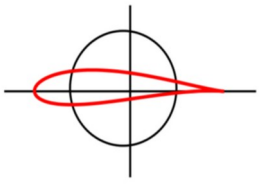



```
17 echo "nameserver 8.8.8.8" > /etc/resolv.conf

18 echo /sbin/mdev > /proc/sys/kernel/hotplug

19 hostname -F /etc/hostname

20 syslogd
21 klogd

22 ifconfig lo 127.0.0.1 up
23 route add -net 127.0.0.0 netmask 255.0.0.0 gw 127.0.0.1 lo

24 # static ip
25 ifconfig eth0 192.168.1.100 up
26 route add default gw 192.168.1.1

27 # dynamic ip
28 # udhcpc -s /etc/udhcpc.script

29 telnetd
```

1. Sistem, SD kart ile Açılış





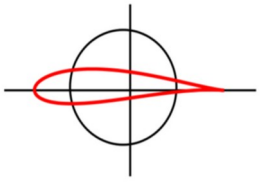
Bu örnek sistemde kök çekirdek ve kök dosya sistemi SD kart içindedir.
En genel açılış tekniğidir.

Test için uygundur.
Profesyonel sistemlerde kök dosya sistemi ASLA SD/MMC içinde olmamalıdır.

Şu ana kadar, kök dosya sisteminin **/opt/gomsis/RootFS** dizini altına kuruluşu tamamlandı.

Bu dizin aşağıdaki gibi SD karta kopyalanır.
İşlemler PC tarafında yapılacaktır.

```
$ sudo mkdir /mnt/root

$ sudo dmesg|tail

$ sudo mount /dev/mmcblk0p1 /mnt/root

$ df

$ sudo cp -a /opt/gomsis/RootFS/* /mnt/root/

$ df
$ ls /mnt/root

$ sudo chown -R root:root /mnt/root/*

$ sudo umount /dev/mmcblk0p1
```



Kart çıkarılıp borda takılır ve borda güç verilir.

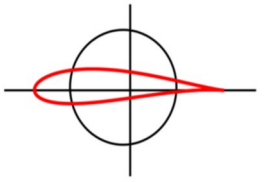
u-boot seviyesinde açılış durdurulur ve aşağıdaki komutlar girilerek login'e kadar gelinir.

```
> ls mmc 0:1
> ls mmc 0
> ls mmc 0:2 bin

> fatload mmc 0 0x46000000 zImage

> fatload mmc 0 0x49000000 dtb

> setenv bootargs console=ttyS0,115200 earlyprintk \
        root=/dev/mmcblk0p2 ro rootwait

> print bootargs

> bootz 0x46000000 - 0x49000000
```

Sistem açılır, root ile giriş yapılır. Şifre yoktur.

Dosya sistemi ro bağlandığı için doğrudan güncellenemez.
Önce rw bağlanır, sonra güncellenir.

```
$ mount -o remount -o rw /

# Bu arada güncellemeleri yap.

$ mount -o remount -o ro /
```

Otomatik Açılış

Sistemi açmak için u-boot seviyesinde açılış durdurulup, el yordamı ile açılış yapılmıştı.

Dışarıdan müdahale olmadan yapılan açılışa, **otomatik açılış** denir.

Bütün bordlarda otomatik açılış hemen hemen standard hale getirilmiştir.

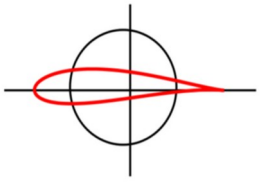
Genelde çalışacak u-boot komutları bir dosyaya yazılır ve bu dosya betik imajı haline getirilir.
Borda ait **bootcmd** komutu bu betik imajını başlatır.

OrangePi için, başlatma betiğinin adı boot.scr'dir. Her bord için bu farklı olabilir. Ayrıca kullanıcı isterse tamamen farklı adlar da verebilir. Bizler boot.scr betiğini kurararak devam edeceğiz.

boot.scr betiği, var olan bir betiğe, mkimage ile header eklenerek üretilir.

```
$ vi kuantek.scr
```

```
setenv bootargs console=ttyS0,115200 earlyprintk root=/dev/mmcblk0p2 ro rootwait
fatload mmc 0 0x46000000 zImage
fatload mmc 0 0x49000000 dtb
bootz 0x46000000 - 0x49000000
```



Görüleceği gibi, daha önce, el ile girilen bütün komutlar, olduğu gibi bir dosyaya yazılmıştır.

Dosya adının bir önemi yoktur.

U-Boot çalıştıracağı betiklerin **u-boot imajı** olmasını bekler.

U-boot imajı üretmek

Bir betik veya herhangi bir dosya, aşağıdaki şekilde u-boot imajı haline getirilir.

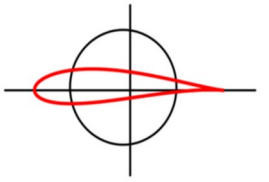
```
$ mkimage --help
```

mkimage komut PC'de mevcut değilse, u-boot/tools altından /usr/bin altına kopyalanabilir.

```
$ mkimage -C none -A arm -T script -d kuantek.scr -n "ilk sistem" boot.scr

$ file boot.scr

$ mkimage -l boot.scr
```

boot.scr dosyası artık bir u-boot imajıdır ve u-boot tarafından doğrudan yürütülebilir.

U-Boot imajı = <u-boot header, data>



boot.scr imajı vfat bölünde oturmalıdır. Bu bilgiler u-boot derlemesi sırasında verilmişti.

Her bordun varsayılan betik adları farklıdır. Default env dosyasından, varsayılan başlangıç imaj adları, distro_bootcmd tanımından incelenebilir.

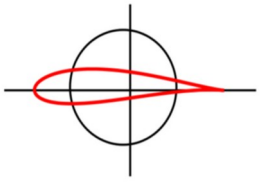
boot.scr dosyası, SD kartın fat bölümüne kopyalanır, SD kart borda takılır ve boot edilir.

Açılış u-boot seviyesinde durdurulur.

```
> saveenv   # save default env.
```

ile varsayılan env dosyası fat bölümüne yazılır.
Fat bölümü artık aşağıdaki gibi gözükecektir.

```
> ls mmc 0

  8854624   zImage
      244   boot.scr
    30537   dtb
   131072   kuantek.env

4 file(s), 0 dir(s)
```

kuantek.env adını, çeşni olsun diye, u-boot derlemesi sırasında,
$ make menuconfig aşamasında vermiştik.

u-boot açılırken önce **kuantek.env** dosyasını yükler.

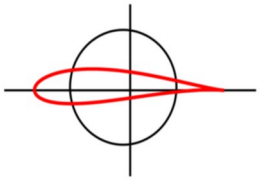
Bu dosya olmazsa varsayılan env bilgileri ile devam eder ki zaten env bilgilerini hiç güncellemedik.

bootcmd komutu, fat bölümü içinde boot.scr dosyasını arar, bulursa hemen işletir.

> reset komutu girilise bord login'e kadar, dışardan müdahale olmadan gelecektir. login gelir, minicom'dan root ile girilir.

Testleri kolaylaştırmak ve hep aynı host IP değeri ile çalışmak için PC'nin bütün ağ erişimi kapatılır, PC ile bord, doğrudan eternet kablosu ile bağlanır ve aşağıdaki gibi bütün PC'lere 192.168.1.22/24 ip değeri atanır.

```
$ ifconfig -a    # eternet kartının adını öğren
$ ifconfig enp5s7 192.168.1.22 netmask 255.255.255.0 up
$ ifconfig   # ip'yi kontrol et
```



Ubuntu 20 LTS'de bu yöntem ile IP ataması karasız olabilir.

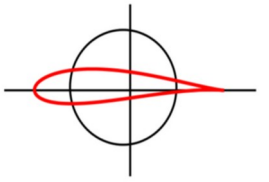
Doğrudan sağ üst köşedeki "Wired Connected" sekmesinden, ipv4, manual girişi ile de aynı IP verilebilir.

PC'den telnet ile bağlan

PC'den **$ telnet 192.168.1.100**

Bord içindeyiz...

```
$ df
$ ps
$ top
$ cd /proc
$ cd /sys
$ pstree -p
$ more /etc/inittab
$ cat /proc/cmdline  # kernel'dan userspace'e keyfi parametre aktarılabilir
...
$ poweroff
```



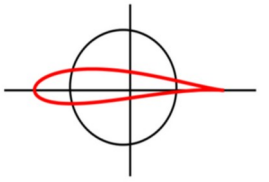
FAT bölümüne Erişim

SD'deki FAT bölümü açılış sırasında bağlanmaz, gerek de yoktur. Çünkü gerekli dosyalar açılış sırasında u-boot veya çekirdek tarafından doğrudan okunur.

Yine de 1. bölüm aşağıdaki gibi bağlanıp, güncelleme yapılabilir.

Aşağıdaki komutlar bord tarafında girilecektir.
Telnet ile bağlan ve dene.

```
$ mkdir /mnt/boot

$ cat /proc/partitions

$ mount /dev/mmcblk0p1 /mnt/boot

$ df

$ cd /mnt/boot

$ ls -l

$ cd

$ sudo umount /mnt/boot
```

Diğer AutoBoot Yöntemleri

Açılış u-boot ortamında durdurlur ve boot scr içindeki bilgiler bootcmd değişkenine atanabilir.

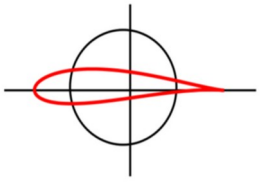
Örnek

```
> setenv bootargs console=ttyS0,115200 earlyprintk root=/dev/mmcblk0p2 ro rootwait

> setenv bootcmd="fatload mmc 0 0x46000000 zImage ; \
                  fatload mmc 0 0x49000000 dtb ;  \
                  bootz 0x46000000 - 0x49000000"

> savenev

> reset
```

Bu durumda bootargs ve bootcmd değişkenleri doğrudan global env tanımları içine girecektir.

saveenv ile, env bilgileri SD kartın FAT bölümüne yazılır.

Açılışta **bootcmd** otomatik olarak işleyecek ve sistem açılacaktır.



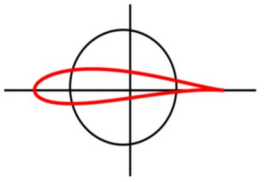
Bir diğer yöntem ise, boot.scr'yi açılışta doğrudan çağırmaktır.

Bunun için, u-boot derlemesinde, make menuconfig sırasında aşağıdaki gibi tanımlar yapılır.

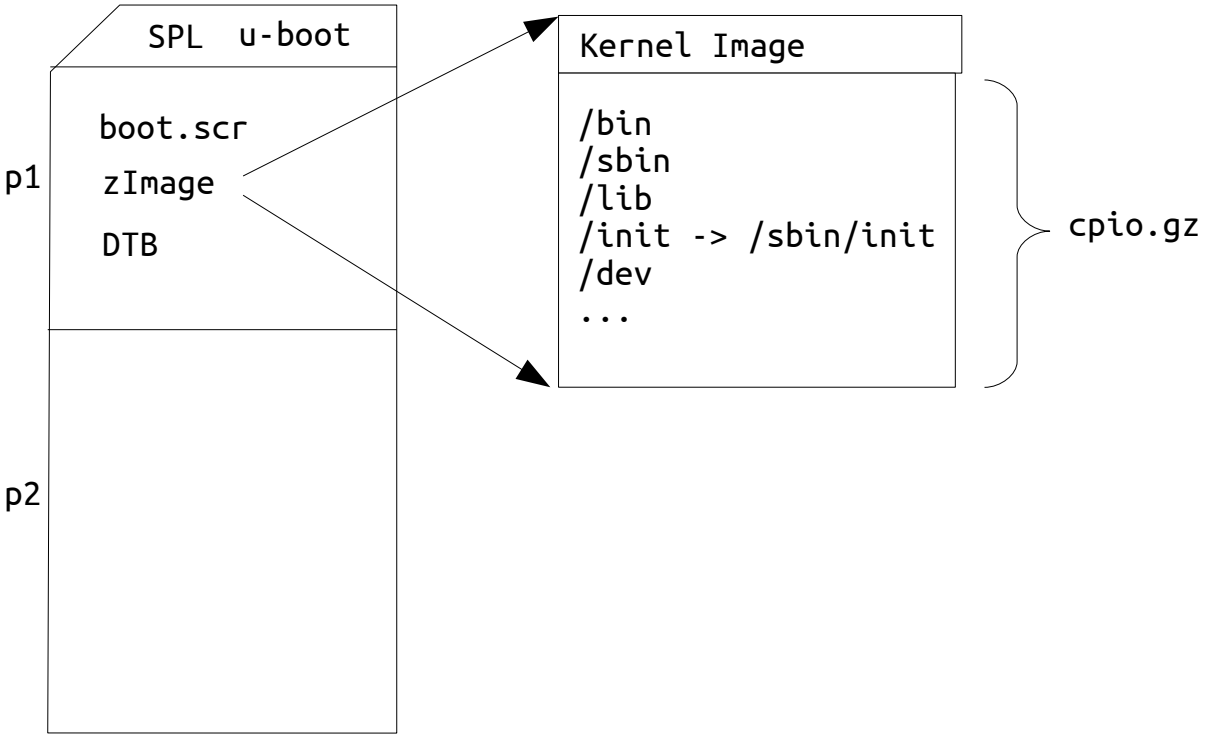
```
[*] Enable a default value for bootcmd
     (fatload mmc 0 43100000 boot.scr && source 43100000)
```

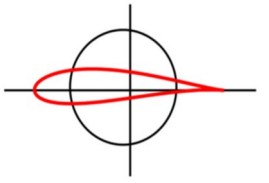
Bu iki yöntemde, env dosyası içindeki açılış komutlarının hiç birisi işlemez. Sistem doğrudan açılır.

Tamamı SD kart içinde olan sistemin kuruluşu ve testi tamamlanmıştır.



2. Sistem, InitRamFS Açılışı





Birinci gömülü sistemde, çekirdek ve kök dosya sistemi SD kart üzerindeydi.

Çekirdek ve kök dosya sistemi fiziksel olarak farklı yerlerde olabilir.

Linux çekirdeği kök dosya sistemini pek çok yerden mount edebilir.

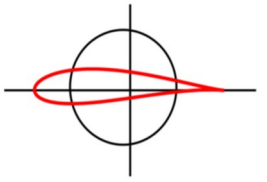
Şimdiki örnek çalışmada kök dosya sistemi doğrudan çekirdek içine gömülecektir. Projede ayrı bir kök dosya sistemi olmayacaktır ve dolayısı ile SD kartta ikinci bir dosya sistemine gerek kalmayacaktır.

Kök dosya sisteminin doğrudan RAM bellekte kurulmasına ve mount edilmesine **initial RAM file system** veya kısaca **initramfs** denir.

Initramfs dosya sistemine "öncü kök dosya sistemi" de denir.

Standard dağıtımlarda öncü kök dosya sisteminin asıl görevi,
esas kök dosya sistemi için ön hazırlık yapmak ve gerekli çekirdek modüllerini yüklemektir.

Bizler initramfs dosya sistemini öncü değil de esas kök dosya sistemi olarak kullanacağız.



Yararları

Sistem çok basit görünür, ayrı bir kök dosya sistemine ve onun oturacağı bir disk bölümüne gerek yoktur.

Sistem SD karttan bağımsız çalışır.
Sistem açıldıktan sonra SD kart yerinden çıkarılabilir.

Kök dosya sistemi her açılışta sıfırdan kurulur.
Donanım hataları hariç, kök dosya sisteminin bozulma ihtimali yoktur.

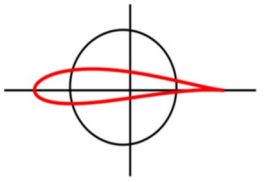
Kök dosya sistemi her zaman r/w modundadır. Böyle olmasına rağmen, ani kapanma sorun olmaz. Çünkü her açılışta kök dosya sistemi baştan kurulacaktır.

Acil açılış sistemleri için çok uygundur.

Sistem initramfs ile açıldıktan sonra, esas kök dosya sistemini kontrol edebilir. Gerekli eksiklikleri tamamlar ve esas kök dosya sistemini mount ederek kendisi aradan çekilebilir.

Initramfs, esas kök dosya sistemi için de çok uygundur.

...



Sakıncaları

Çekirdeğe bir ek yapıldığı için, kök dosya sisteminde bulunan bütün programların, uygulama programları dahil, kaynak kodları açılmalıdır.

Kural: GPL lisanlı programa yapılan her güncelleme GPL olmalı ve yayınlanmalıdır.

Bu sıkıntının etrafından dolanmak için, özel lisanslı kodlar ayrı bir disk bölümüne konup, kök dosya sisteminden çağrılabilir.

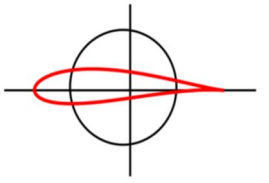
Kök dosya sistemi her güncellenediğinde, çekirdek yeniden derlenmelidir.

Açılış, doğrudan SD karta açılışa göre biraz daha yavaştır.
Çünkü sıkıştırılmış kök dosya sistemi önce ram'e unzip edilir ve sonra ram'deki kök dosya sistemi mount edilir. SD kartta ise sadece mount işlemi vardır, unzip işlemi yoktur.

Kök dosya sistemi büyük olmamalıdır.
Çünkü dosya sistemi doğrudan ram'e yerleştiği için aşırı bellek tüketmektedir.

Güç kesilince bütün bilgiler kaybolur.
Kalıcı olması gereken bilgiler, uçucu olmayan bir bellekte ayrıca saklanmalıdır.

...



RootFS'in Kuruluşu

Birinci sitemde kurulan kök dosya sistemi olduğu gibi kullanılacaktır.

Birinci sistemde, çekirdek, öncelikle **/sbin/init** programı çalıştırıldı.
Bu yöntemde ilk çalışacak program **/init** programıdır.

Acil veya öncül açılış gibi özel bir iş yapmadığımız için /init programı sembolik link ile /sbin/init'e yönlendirilmiştir.

Acil açılış sistemi veya öncü açılış yapıyor olsaydık, **/init** kodu genelde kabuk program olacaktı, dosya sistemlerinin denetimi, modüllerin önceden yüklenmesi gibi işlemler yapılabilirdi.

Bu açılışa uygun rcS betiği ilk sistemle tamamen aynıdır.
Farklı olarak, /dev dosya sistemi çekirdek tarafıdan yaratılır ama sadece esas kök dosya sistemi mount edildikten sonra, mount edilir.

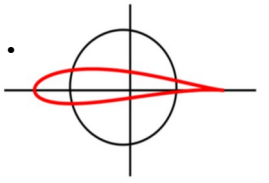
Bu örnek esas kök dosya sistemi yoktur ve çekirdek /dev sistemini mount edemeyecektir. Bundan dolayı **/dev** sistemi **/etc/rcS** içinde açıkça mount edilir.

mount -t devtmpfs devtmpfs /dev

Çekirdeğe gömülü Initramfs yönteminde dikkat edilecek noktalar,

- Açılışta, çekirdek tarafından çalıştırılacak ilk program **/init**'dir.
- devtmpfs dosya sistemi rcS içinde açıkça mount edilmelidir.
- Kök dosya sistemi çekirdeğe gömülüdür.

Şimdi kök dosya sistemi, çekirdek derlemesi ile çekirdeğin içine eklenecektir.



Çekirdeğin Yeniden Derlenmesi

```
$ cd /opt/gomsis/RootFS/etc
$ rm rcS && ln -s rcS.2 rcS  # açılış betiğini değiş
$ diff -w rcS.1 rcS.2
$ cd /opt/gomsis/linux
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig

General setup
  [*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
     (/opt/gomsis/RootFS) Initramfs source file(s)
     (1000)   User ID to map to 0 (user root) (NEW)
     (1000)   Group ID to map to 0 (group root) (NEW)
```

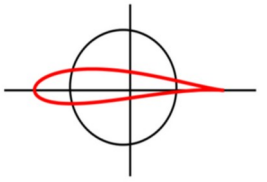
Katılımcı 1000 yerine kendi userId:group değerini yazmalıdır.
Bu değerler `$ id` komutu ile bulunabilir.

Çekirdek aşağıdaki gibi derlenir.
Derleme sırasında, RootFS/ dizini olduğu gibi çekirdeğin içine eklenecektir.

```
$ cd /opt/gomsis/linux

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage
```

Derleme sırasında aşağıdaki gibi satır geçecektir.
GEN usr/initramfs_data.cpio.gz

Kök dosya sistemi önce cpio arşivi haline getirilir, sonra sıkıştırılır
sonra da çekirdeğin içine eklenir.



Derleme bittikten sonra, zImage'nin boyu kontrol edilmelidir.

```
$ cd /opt/gomsis/linux

$ ls -l arch/arm/boot/zImage
-rwxrwxr-x 1 student student 15750488 Dec 30 08:27 arch/arm/boot/zImage
```

8M civarında olan çekirdek, 16M civarında gözükmektedir.
Çünkü RootFS çekirdeğe eklenmiş, ve çekirdek şişmiştir.

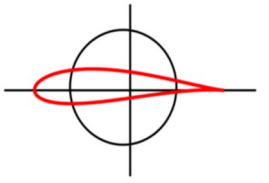
Şişme miktarı kök dosya sisteminin, diskteki boyutundan her zaman daha azdır. Çünkü kök dosya sistemi sıkıştırılarak çekirdek içine alınır.

Ayrıca modül derlemesine ve diğer işlemlere gerek yoktur.
Çünkü çekirdeğe sadece RootFS eklenmiştir, bunun dışında seçim yapılmamıştır.

Initramfs kurarken çekirdek seçimlerinde değişiklik yapılmamalıdır.

Şimdi yeni çekirdek SD kartın birinci bölümüne kopyalanır.

```
$ cd /opt/gomsis/linux

$ sudo mount /dev/mmcblk0p1 /mnt/boot

$ cp arch/arm/boot/zImage /mnt/boot
```



U-Boot Betiği

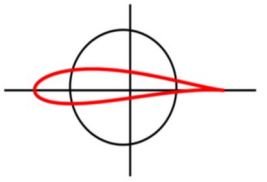
Açılış u-boot seviyesinde durdurulur ve aşağıdaki komutlar girilerek el yordamı ile test yapılabilir.

Ya da otomatik açılış için bu komutlar aşağıdaki gibi bir dosyaya yazılır ve imaj haline getirilerek vfat bölümüne kopyalanır.

```
$ vi kuantek.scr
setenv bootargs console=ttyS0,115200 earlyprintk
fatload mmc 0 0x46000000 zImage
fatload mmc 0 0x49000000 dtb
bootz 0x46000000 - 0x49000000
```

Bu betiğin bir önceki betikten tek farkı vardır.
Sadece bootargs içindeki “kök dosya sistemi tanımı” kırpılmıştır.

Çünkü kök dosya sistemi artık çekirdek içindedir ve ayrıca bir cihaz ismi belirtmeye gerek yoktur.

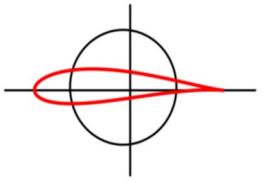


kuantek.scr betiğinin u-boot tarafından tanınabilmesi için imaj haline getirilmesi gerekir.

```
$ mkimage -C none -A arm -T script -d kuantek.scr boot.scr

$ file boot.scr

$ mkimage -l boot.scr

$ sudo cp boot.scr /mnt/boot

$ ls -l /mnt/boot
total 15414
-rwxr-xr-x 1 root root      213 Dec 30 08:38 boot.scr
-rwxr-xr-x 1 root root    30537 Dec 29 08:22 dtb
-rwxr-xr-x 1 root root 15750488 Dec 30 08:37 zImage

$ sudo umount /mnt/boot

$ df
```

Bütün sistemin açılış için bu 3 dosya yeterlidir.
Kök dosya sistemi zImage içinde oturmaktadır.

Sistem açıldıktan sonra SD kart çıkarılabilir.



Açılış Testi

SD kart borda takılır ve sistem login'e kadar açılır.

starting kernel aşamasında, normalden daha fazla bekleme yapılır.
Çünkü burada hem çekirdek kodu hem kök dosya sistemi unzip edilmektedir.

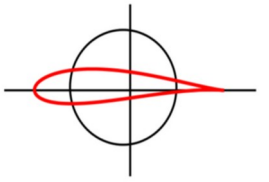
SD kart yerinden çıkarılırsa bile, sistem çalışmaya devam eder.

bord tarafında, free komutu ile boş bellek miktarı tespit edilebilir.
Bu makinede RAM bellek 256M'dir.

```
$ free
              total        used        free      shared  buff/cache   available
Mem:         171140       14756      119932       33828       36452      118888
Swap:             0           0           0
```

df'de kök dosya sistemi **/dev/root** olarak ayrıca gözükmez.

```
$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
devtmpfs                 77376         0     77376   0% /dev
tmpfs                    85568         4     85564   0% /tmp
tmpfs                    85568         0     85568   0% /dev/shm
tmpfs                    85568        48     85520   0% /var
```

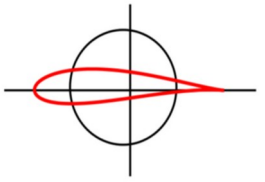


Kök dosya sistemi r/w bağlıdır.

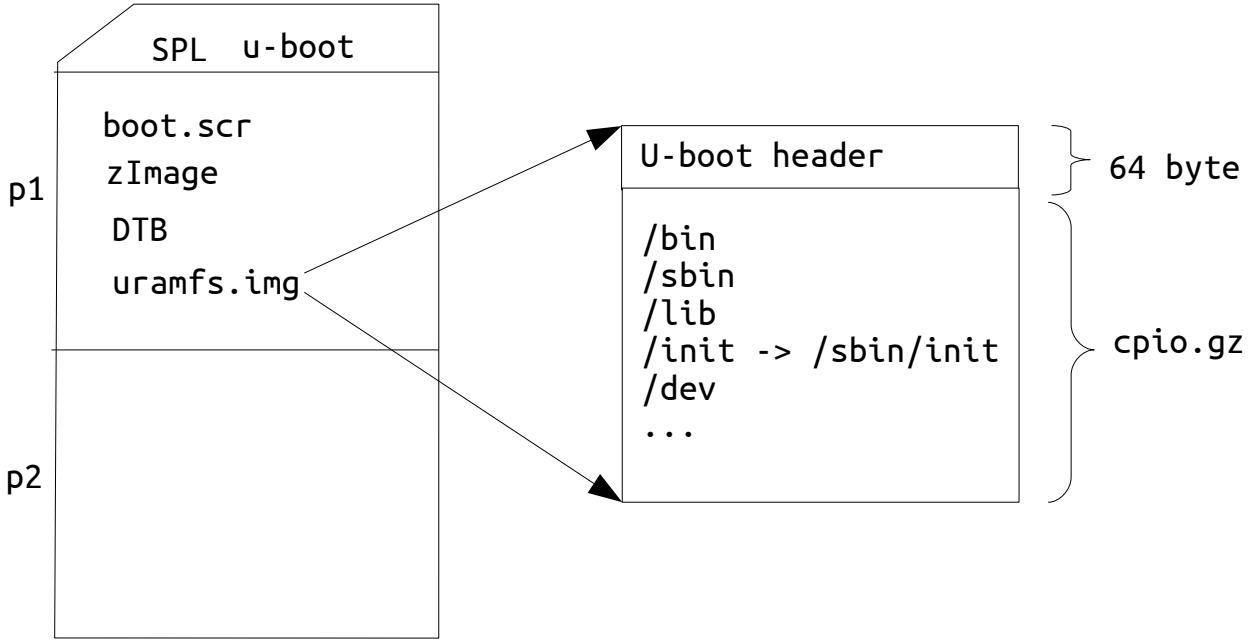
```
$ touch foo
```

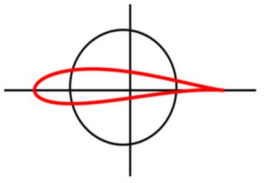
Boş mount komutu ile bağlı bölümlerin özelliklerine bakılabilir.

```
$ mount
```



3. Sistem, InitRamFS Açılışı, Ayrık İmaj



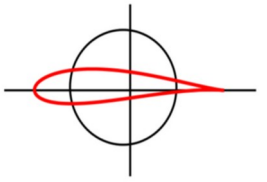


Bir önceki initramfs destekli açılış ile tamamen aynıdır.

Farklı olarak kök dosya sistemi çekirdeğin içinde değil, SD kart üzerinde ve sıkıştırılmış olarak tutulacaktır.

Böylece kök dosya sistemi, ilk sistemi göre daha kolay güncellenir, çekirdeği yeniden derlemeye gerek yoktur.

Ayrıca kök dosya sistemi çekideğe eklenmeyeceği için çalışan programların GPL olmasına gerek yoktur, özel lisanslı programlar çalıştırılabilir.



Çekirdeğin Derlenmesi

1. sistemdeki çekirdek olduğu gibi kullanılabilir.

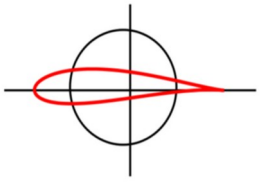
Ya da 2. sistemdeki çekirdek derlemesinde bulunan

(/opt/gomsis/RootFS) Initramfs source file(s)

ifadesindeki kök dosya dizini olan **/opt/gomsis/RootFS** silinip, tekrar çekirdek derlemesi yapılır.

```
$ cd /opt/gomsis/linux

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage

$ ls -l arch/arm/boot/zImage
-rwxrwxr-x 1 student student 8311384 Dec 30 08:50 arch/arm/boot/zImage
```

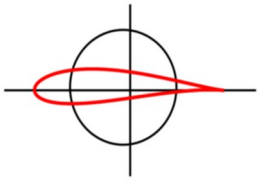
Çekirdek boyunun küçüldüğü görülür.



RootFS'in Hazırlanması

rcS betiği, 2. sistemdeki ile tamamen aynı kalacaktır.

```
$ cd /opt/gomsis

$ sudo chown -R -h root:root RootFS

$ cd RootFS

$ find . | cpio -o -H newc | gzip > ../rootfs.cpio.gz

$ cd ..

$ ls -l rootfs.cpio.gz
-rw-rw-r-- 1 student student 7857515 Dec 30 08:55 rootfs.cpio.gz


$ mkimage -A arm -T ramdisk -C none -n "test3" -d rootfs.cpio.gz uramfs.img
Image Name:   test3
Created:      Mon Dec 30 08:57:11 2024
Image Type:   ARM Linux RAMDisk Image (uncompressed)
Data Size:    7857515 Bytes = 7673.35 KiB = 7.49 MiB
Load Address: 00000000
Entry Point:  00000000
```

U-Boot header 64-byte yer kaplar

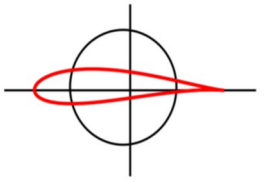
```
$  ls -l rootfs.cpio.gz
-rw-rw-r-- 1 student student 7857515 Dec 30 08:55 rootfs.cpio.gz

$ ls -l uramfs.img
-rw-rw-r-- 1 student student 7857579 Dec 30 08:57 uramfs.img


$ file uramfs.img
$ mkimage -l uramfs.img
```

U-Boot Betiğinin Hazırlanması

```
$ vi kuantek.scr
setenv bootargs console=ttyS0,115200 earlyprintk
fatload mmc 0 0x46000000 zImage
fatload mmc 0 0x43300000 uramfs.img
fatload mmc 0 0x49000000 dtb
bootz 0x46000000 0x43300000 0x49000000
```

kuantek.scr betiğinin u-boot tarafından tanınabilmesi için u-boot imajı haline getirilmesi gerekir.

```
$ mkimage -C none -A arm -T script -d kuantek.scr boot.scr

$ file boot.scr
```

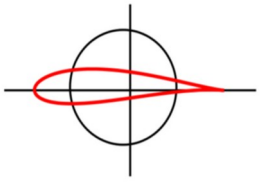


Açılış Testi

Çekirdek, dtb, ramdisk ve boot.scr vfat bölümüne kopyalanır ve sistem açılır.

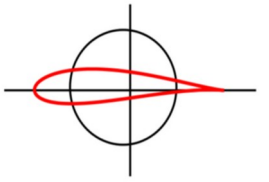
```
$ sudo mount /dev/mmcblk0p1 /mnt/boot

$ cd /opt/gomsis

$ sudo cp linux/arch/arm/boot/zImage /mnt/boot
$ sudo cp linux/arch/arm/boot/dts/sun8i-h2-plus-orangepi-zero.dtb /mnt/boot/dtb
$ sudo cp uramfs.img                  /mnt/boot
$ sudo cp boot.scr                    /mnt/boot

$ ls -l /mnt/boot
total 15824
-rwxr-xr-x 1 root root     258 Dec 30 09:23 boot.scr
-rwxr-xr-x 1 root root   30537 Dec 30 09:22 dtb
-rwxr-xr-x 1 root root 7857579 Dec 30 09:23 uramfs.img
-rwxr-xr-x 1 root root 8311384 Dec 30 09:22 zImage


$ sudo umount /mnt/boot
```

SD kart borda takılır ve güç verilince, login'e kadar gelinir.

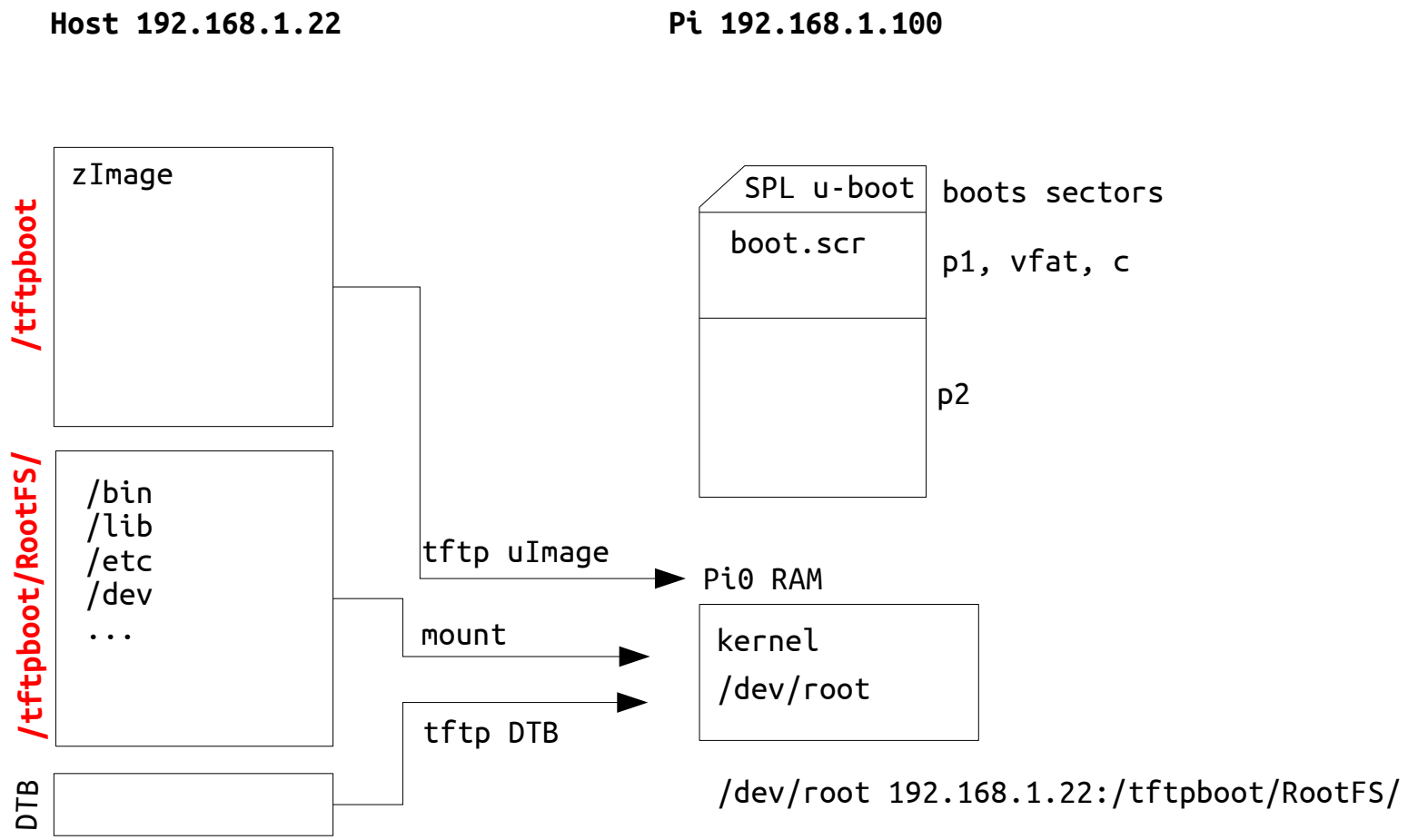


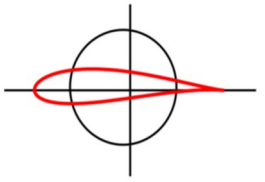
Açılışın İncelenmesi

```
...
Scanning mmc 0:1...
Found U-Boot script /boot.scr
259 bytes read in 2 ms (126 KiB/s)              <-- boot.scr

## Executing script at 43100000                 <-- boot.scr

8836512 bytes read in 663 ms (12.7 MiB/s)       <-- kernel
15107893 bytes read in 1134 ms (12.7 MiB/s)     <-- rootfs
30537 bytes read in 6 ms (4.9 MiB/s)            <-- dtb

## Loading init Ramdisk from Legacy Image at 43300000 ...
   Image Name:   test3
   Image Type:   ARM Linux RAMDisk Image (uncompressed)
   Data Size:    15107829 Bytes = 14.4 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
...

Starting kernel ...
```

4. Sistem, Ağ Üzerinden Açılış





Bu yöntemde SD kart üzerinde sadece spl ve u-boot bulunacaktır.

zImage ve dtb, tftp ile uzaktan yüklenecektir.

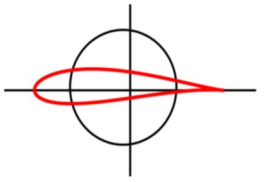
Kök dosya sistemi ise NFS ile ağ üzerinden bağlanacaktır.

Ağ ortamları için çok uygudur.

Ayrıca olabilecek en uygun test ortamıdır.

Hem RootFS hem de boot tarafında güncellemeler anında test edilebilir.

Production ortamında ağ sistemi olmasa bile, her gömülü sistem projesinin bu yapılandırma ile başlatılması, projeye çok büyük hız kazandıracaktır.

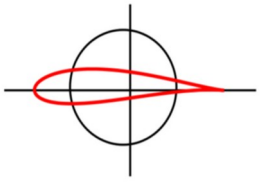


tftp server

Kuruluş

```
$ sudo vi /etc/xinetd.d/tftp
```

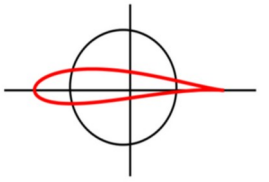
```
service tftp
{
protocol = udp
port = 69
socket_type = dgram
wait = yes
user = nobody
server = /usr/sbin/in.tftpd
server_args = /tftpboot
disable = no
}
```

```
$ sudo mkdir /tftpboot
$ sudo chmod 777 /tftpboot
$ sudo chown nobody:nogroup /tftpboot

$ ls -ld /tftpboot
drwxrwxrwx 2 nobody nogroup 4096 Mar 31 21:18 /tftpboot

$ ls -l /usr/sbin/in.tftpd

$ sudo service xinetd restart
```

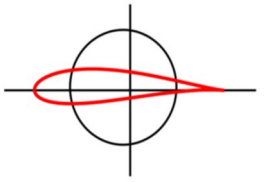
basit yerel test

```
$ cd /tftpboot
$ vi foo
$ ls -l

$ cd /opt/gomsis
$ ls foo   # foo yok

$ tftp localhost
> get foo
> q

$ ls -l   # foo var
$ rm /tftpboot/foo  foo
```

NFS server

```
$ sudo service nfs-kernel-server restart
$ pidof nfsd
$ cat /proc/fs/nfsd/versions
```

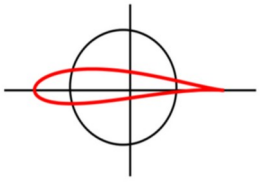



Çekirdeğin derlenmesi

Çekirdeğe, ağ üzerinden RootFS ile açılabilmesi için NFS desteği eklenir.

```
$ cd /opt/gomsis/linux

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig

File systems  --->
   [*] Network File Systems  --->
       <*>   NFS client support
       <*>     NFS client support for NFS version 2
       <*>     NFS client support for NFS version 3
       [*]   Root file system on NFS

   [*] Networking support  --->
        Networking options  --->
         [*]   IP: kernel level autoconfiguration
         [*]     IP: DHCP support
         [*]     IP: BOOTP support
         [*]     IP: RARP support


$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage
```

Aslında ilk derlemeden beri burası seçilmiştir.
Ayrıca derlemeye gerek yoktur.

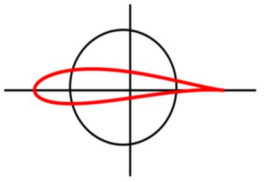


Kök Dosya Sistemi

Öncelikle rcS aşağıdaki gibi güncellenmelidir.

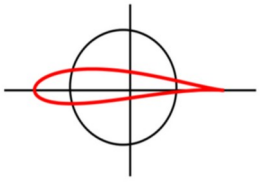
```
$ cd /opt/gomsis/RootFS

$ vi etc/rcS
```

```
#!/bin/sh +x
#
export PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin

mount -t proc   proc   /proc
mount -t sysfs  sysfs  /sys
mount -t tmpfs -o mode=1777 tmpfs  /tmp

mkdir /dev/pts
mount -t devpts -o gid=5,mode=620 devpts /dev/pts

mkdir /dev/shm
mount -t tmpfs -o mode=0777 tmpfs /dev/shm
```



```
mount -t tmpfs -o mode=0755,nosuid,nodev tmpfs /var
mkdir /var/cache
mkdir /var/lock
mkdir /var/log
mkdir /var/run
mkdir /var/spool
mkdir /var/tmp

hostname UCanLinux

syslogd
klogd

ifconfig lo 127.0.0.1 up
route add -net 127.0.0.0 netmask 255.0.0.0 gw 127.0.0.1 lo

telnetd
```

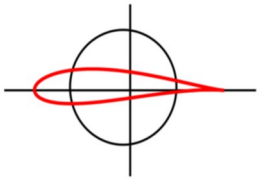
Dikkat edilirse lo hariç, ağ yapılandırması ile ilgili komutlar kaldırılmıştır.



Kök dosya sistemi, olduğu gibi /tftpboot altına kopyalanır.

```
$ cd /opt/gomsis

$ sudo cp -a RootFS /tftpboot/

$ ls -l /tftpboot

$ tree -L 2 /tftpboot
```

```
/tftpboot
├── dtb
├── RootFS
│   ├── bin
│   ├── boot
│   ├── dev
│   ├── etc
│   ├── init -> /sbin/init
│   ├── lib
│   ├── mnt
│   ├── opt
│   ├── proc
│   ├── root
│   ├── sbin
│   ├── sys
│   ├── tmp
│   ├── usr
│   └── var
└── zImage
```



Aslında RootFS, her yerde olabililir.
/tftpboot altında bulunması zorunda değildir.

Hatta hiç kopyalamadan, doğrudan /opt/gomsis/RootFS olarak da kullanılabilir.
Anlatımı kolaylaştırmak için /tftpboot altına kopyalanmıştır.

RootFS, host tarafında, aşağıdaki gibi paylaşıma açılır.

```
$ sudo vi /etc/exports
/tftpboot 192.168.1.0/24(rw,insecure,no_subtree_check,no_root_squash)
/tftpboot 127.0.0.1(no_subtree_check)
```

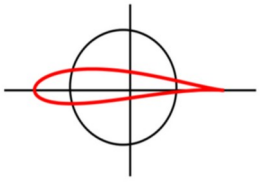
Bordun IP değeri 192.168.1.100'dir.
192.168.1.0/24 tanımı ile 192.168.1.XXX'e sahip bütün IP'ler bu dizini uzaktan bağlayabilirler.

Buraya tek bir IP değeri girilebilir. Ya da * girilerek herkese paylaştırılabilir.

/etc/exports güncellemesi ile yapılan bu paylaşım, NFS'e haber verilmelidir.

```
$ sudo exportfs -avr
exporting 192.168.1.100:/tftpboot
exporting 127.0.0.1:/tftpboot
```

Aslında sadece RootFS'i, /tftpboot/RootFS olarak export edebilirdik.
Ama bu şekilde /tftpboot altında diğer dosyalara da erişim yapılabilir.
Bu da testlerimizi kolaylaştıracaktır.



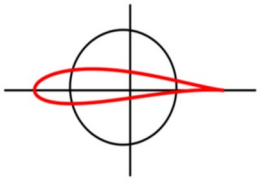
NFS'in yerelde testi

```
$ sudo mount -t nfs 127.0.0.1:/tftpboot /mnt/boot

$ df /mnt/boot
Filesystem          1K-blocks     Used Available Use% Mounted on
127.0.0.1:/tftpboot  57410048 35862528  18601984  66% /mnt/boot

$ sudo umount /mnt/boot
```

Açılış Dosyalarının Kopyalanması

zImage ve dtb, /tftpboot altına kopyalanır.

```
$ cd /opt/gomsis

$ cp linux/arch/arm/boot/zImage  /tftpboot

$ cp linux/arch/arm/boot/dts/sun8i-h2-plus-orangepi-zero.dtb  /tftpboot/dtb

$ ls -l /tftpboot
total 8156
drwxrwxr-x 14 root    root       4096 Dec 29 14:00 RootFS
-rw-rw-r--  1 student student   30537 Dec 31 13:42 dtb
-rwxrwxr-x  1 student student 8311384 Dec 31 13:42 zImage
```

Static IP ataması

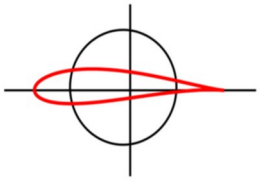
NULL modem kablosu ile PC ve bord, birbirlerine bağlanacaktır.

Bordun dışarı ile ilgili bütün ağ iletişimi kesilmelidir.

Ayrıca bütün bord'lar aynı IP değeri ile çalışmalıdırlar ki testler standard bir biçimde uyugulanabilsin.

```
$ sudo ifconfig eth0 192.168.1.22 netmask 255.255.255.0 up
$ sudo route add default gw 192.168.1.22

# sonucu kontrol et
$ ifconfig
```

Bütün katılımcı PC'lerinin IP değeri 192.168.1.22 olmalıdır ki testleri kolaylıkla yapabilelim.



Açılışın el ile test edilmesi

Açılış tekniğini kavramak için, boot.scr kurulmadan, el ile açılış yapılacaktır.

SD kartın 1. bölümünde bulunan bütün dosyalar silinecektir.

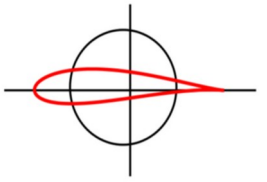
vfat içindeki hiç bir bilgiye ihtiyacımız yoktur.

SD kartın boot sektöründe bulunan "SPL+UBoot" açılış için yeterli olacaktır.

SD kart notebook'a takılır ve bütün dosyalar silinir

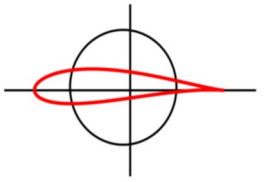
```
$ sudo  mount /dev/mmcblk0p1 /mnt/boot

$ sudo rm -f /mnt/boot/*

$ ls -l /mnt/boot
total 0

$ sudo umount /mnt/boot
```

SD kart borda takılır, minicom başlatılır ve açılış u-boot seviyesinde, boşluğa basılarak durdurulur.



```
> print ipaddr
## Error: "ipaddr" not defined

> setenv ipaddr 192.168.1.100  # bord ip
> print ipaddr
ipaddr=192.168.1.100
> setenv serverip 192.168.1.22  # tftp server ip or host ip
> ping 192.168.1.22             # check the network connection

> tftp 0x46000000 zImage
TFTP from server 192.168.1.22; our IP address is 192.168.1.100
Filename 'zImage'.
Load address: 0x46000000
Loading: ####################
...

> tftp 0x49000000 dtb

> setenv bootargs console=ttyS0,115200n8

> bootz 0x46000000 - 0x49000000

Starting kernel ...

Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
```

Çünkü kök dosya sistemi henüz mount edilmemiş.

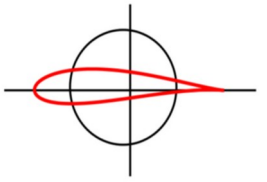


Kök Dosya Sisteminin Uzaktan mount edilmesi

Çekirdeğe **"şu makinede bulunan kök dosya sistemini NFS üzerinden bağla"** anlamında bir parametre geçirirsek, çekirdek, RootFS'i SD kart üzerinde veya başka bir yerde aramayacak, NFS üzerinden mount edecek ve standard açılışını yapacaktır.

Açılış tekrar u-boot seviyesinde durdurulur ve gerekli NFS Root parameterleri çekirdeğe bootargs ile aktarılır.

```
> setenv ipaddr 192.168.1.100

> setenv serverip 192.168.1.22
> ping 192.168.1.22

> setenv bootargs console=ttyS0,115200n8 \
     ip=192.168.1.100:192.168.1.22:192.168.1.100:255.255.255.0:test:eth0:off \
     root=/dev/nfs \
     rw \
     nfsroot=192.168.1.22:/tftpboot/RootFS,nfsvers=4,tcp

> print bootargs

> tftp 0x46000000 zImage

> tftp 0x49000000 dtb

> bootz 0x46000000 - 0x49000000
```



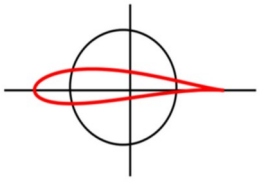
kernel level autoconfiguration

ip=client_ip:server_ip:gw:mask:hostname:device:autoconf:dns1:dns2
nfsroot=server-ip:root-dir,nfs-options

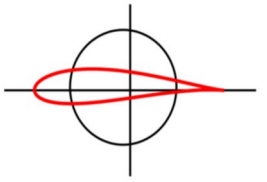
```
autoconf= {on|off|dhcp|bootp|rarp|both}


 off or none: don't use autoconfiguration
               (do static IP assignment instead)
 on or any:   use any protocol available in the kernel
              (default)
 dhcp:        use DHCP
 bootp:       use BOOTP
 rarp:        use RARP
 both:        use both BOOTP and RARP but not DHCP
              (old option kept for backwards compatibility)
```

2. deneme (ağ üzerinde dhcp etkin ise)

```
# otomatik IP almak

> setenv autoload no
```



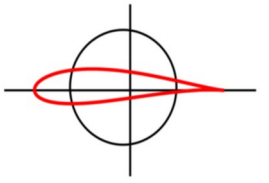
```
> dhcp
BOOTP broadcast 1
DHCP client bound to address 192.168.1.101 (179 ms)

> print ipaddr
ipaddr=192.168.1.101

# el ile IP vermek
# ipaddr değeri el ile, doğrudan da verilebilir.

> setenv ipaddr 192.168.1.101

# server ip

> setenv serverip 192.168.1.22

> setenv bootargs console=ttyS0,115200n8 ip=dhcp \
     root=/dev/nfs rw nfsroot=${serverip}:/tftpboot/RootFS,nfsvers=4,tcp

> print bootargs  # doğruluğu kontrol et

> tftp  0x46000000 zImage

> tftp  0x49000000 dtb

> bootz 0x46000000 - 0x49000000
```

Sonuçların İncelenmesi

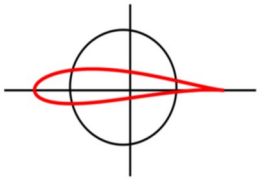
Kernel Level IP Configuration çıkışları:

```
...
Sending DHCP requests ., OK
IP-Config: Got DHCP answer from 192.168.1.1, my address is 192.168.1.101
IP-Config: Complete:
     device=eth0, hwaddr=02:42:50:e9:8f:9b, ipaddr=192.168.1.101,
        mask=255.255.255.0, gw=192.168.1.1
     host=192.168.1.101, domain=home, nis-domain=(none)
     bootserver=0.0.0.0, rootserver=192.168.1.33, rootpath=
     nameserver0=192.168.1.1, nameserver1=192.168.1.1
...
VFS: Mounted root (nfs4 filesystem) on device 0:21.
devtmpfs: mounted
Freeing unused kernel memory: 2048K
Run /sbin/init as init process
...

$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
192.168.1.33:/tftpboot/RootFS
                      57409940  35867216  18596752  66% /
devtmpfs                 18880         0     18880   0% /dev
tmpfs                    85440         0     85440   0% /tmp
tmpfs                    85440         0     85440   0% /dev/shm
tmpfs                    85440        48     85392   0% /var
```



Otomatik açılış

```
$ cd /opt/gomsis

$ vi kuantek.scr
setenv ipaddr 192.168.1.100
setenv serverip 192.168.1.22
setenv bootargs console=ttyS0,115200n8 \
     ip=192.168.1.100:192.168.1.22:192.168.1.22:255.255.255.0:test:eth0:off \
     root=/dev/nfs \
     rw \
     nfsroot=192.168.1.22:/tftpboot/RootFS,nfsvers=4,tcp

tftp 0x46000000 zImage
tftp 0x49000000 dtb
bootz 0x46000000 - 0x49000000

veya

setenv autoload no
dhcp
setenv serverip 192.168.1.22
setenv bootargs console=ttyS0,115200n8 ip=dhcp \
   root=/dev/nfs rw nfsroot=${serverip}:/tftpboot/RootFS,nfsvers=4,tcp
tftp  0x46000000 zImage
tftp  0x49000000 dtb
bootz 0x46000000 - 0x49000000
```

```
$ mkimage -C none -A arm -T script -d kuantek.scr boot.scr
```

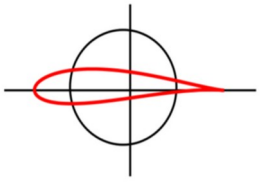
vfat'e kopyala

```
$ sudo mount /dev/mmcblk0p1 /mnt/boot
$ cp boot.scr /mnt/boot
$ ls -l /mnt/boot  # sadece boot.scr mevcut
$ umount /mnt/boot
```

SD kartı tak ve bordu otomatik aç.

örnek çalışma

hello.c kodunu cross derle /tftpboot/RootFS/bin altına at, ve NFS üzerinden çalıştır.



BR destekli RootFS

Kök dosya sistemi RootFS ile kurulur.

Açılış betikleri RootFS ile kurulur.

U-boot, kernel ve toolchain desteği olmasına rağmen, bu destekler çok da kullanışlı değildir.

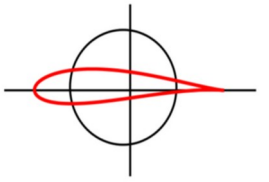
Nihayetinde örneğimizdeki gibi bir RootFS/ elde edilir.

Test için, örnek sistemlerin herhangi birinde, BR ile kurulan RootFS/ doğrudan kullanılabilir.

Genel mantık silsilesi

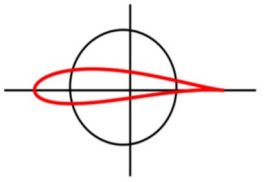
[*] php seçilirse...

- git/wget/rsync/... ile paket indirilir.
- dl/ altında depolanır.
- configure ile Makefile kurulur.
- make ile derleme yapılır.
- make install ile iskelet'e kuruluş yapılır.
- ...
- iskelet, rootfs.tar haline getirilir.
- br/output/images/ altına atılır.



Rootfs.tar'ın üretilmesi

```
# ağı başlatmayı unutma
$ sudo ifdown eth0 --force
$ sudo service network-manager start

$ cd /opt/gomsis
$ tar xvf ftp/br.tgz
$ ln -s buildroot-2024.11 br

$ cd /opt/gomsis/br

$ make clean    # !!!

$ make list-defconfigs  # Bordun burada gözükmemesi hiç önemli değildir :)

$ make orangepi_zero_defconfig
$ make menuconfig

# toolchain'in external seçilmesi tavsiye edilir.

# opencv4 ve gstreamer için,
   Target packages -->
      Libraries -->
        Graphics -->
            [*] opencv4 -->
                istenen programlar seçilir
$ make -j2

# Nihayetinde rootfs.tar elde edilir.
```

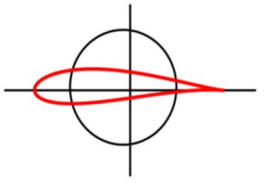



NFS ile test

Üretilen rootfs.tar arşivi, önceki 4 örnek sistemde de test edilebilir. En kolay test NFS ile yapılabilir.

```
$ cd /tftpboot
$ mv RootFS RootFS.old  # eski RootFS'i sakla
$ mkdir RootFS          # yeni roofts için boş dizin kur
$ cd RootFS
$ sudo chown -h root:root *
$ tar xvf /opt/gomsis/br/output/images/rootfs.tar # br ile kurulan sistemi kopyala
$ sudo chown root:root *
$ ls                    # incele
$ ls /usr/bin

$ sudo ifup eth0        # PC'nin ip değerini 192.168.1.22 yap
```

SD kartı borda tak ve br ile kurulan rootfs'in açıldığını gör.
opencv ile ilgili seçilen her kütüphane ve program kök dosya içinde teste hazırdır.



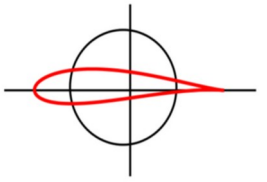
Kernel tarafında elde edilen firmware ve modules ayrıca install edilmelidir. Çünkü çekirdeği br ile derlemediğimiz için, br sistemi modülleri ve firmware bilgilerini nereden alacağını bilemez.

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-  \
       INSTALL_MOD_PATH=/tftpboot/RootFS             \
       modules_install
```

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-  \
       INSTALL_MOD_PATH=/tftpboot/RootFS             \
       firmware_install
```

/tftpboot/RootFS, sadece NFS üzerinden değil, diğer bütün örnek sistemlerde de test edilebilir.

Örnek sistemde modules ve firmware kullanılmadığı için bu adımlar gözardı edilmiştir.



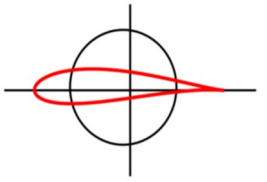
BR'deki gcc'nin Kernel Sürümü

toolchain'de libc/usr/linux/version.h içine bak.

```
$ cd /opt/gomsis/toolchain/arm/arm-linux-gnueabihf/libc/usr/include/linux

$ more version.h
#define LINUX_VERSION_CODE 264707
#define KERNEL_VERSION(a,b,c) (((a) << 16) + ((b) << 8) + (c))
```

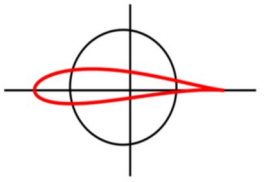
Örnek 264707'nin decode edilmesi.

```
$ printf "%06X\n" 264707
040A03
```

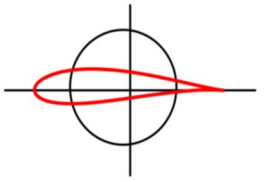
/etc/inittab

```
# This inittab is a basic inittab sample for sysvinit, which mimics
# Buildroot's default inittab for BusyBox.
id:3:initdefault:

si0::sysinit:/bin/mount -t proc proc /proc
#si1::sysinit:/bin/mount -o remount,rw /
si2::sysinit:/bin/mkdir -p /dev/pts
si3::sysinit:/bin/mkdir -p /dev/shm
si4::sysinit:/bin/mount -a
si5::sysinit:/bin/hostname -F /etc/hostname
si6::sysinit:/etc/init.d/rcS

sole::respawn:/sbin/getty -L  console ttyS0 vt100 # GENERIC_SERIAL

# Stuff to do for the 3-finger salute
ca::ctrlaltdel:/sbin/reboot

# Stuff to do before rebooting
shd0:06:wait:/etc/init.d/rcK
shd1:06:wait:/sbin/swapoff -a
shd2:06:wait:/bin/umount -a -r

# The usual halt or reboot actions
hlt0:0:wait:/sbin/halt -dhp
reb0:6:wait:/sbin/reboot
```

/etc/init.d

```
$ ls -l  /etc/init.d

total 52
-rwxr-xr-x 1 root root  423 Nis 28 23:58 rcK
-rwxr-xr-x 1 root root  408 Nis 28 23:58 rcS

-rwxr-xr-x 1 root root  404 Nis 29 00:32 S01logging
-rwxr-xr-x 1 root root 1321 Nis 28 23:58 S20urandom
-rwxr-xr-x 1 root root 1767 Nis 28 23:43 S30dbus
-rwxr-xr-x 1 root root  359 Nis 28 23:58 S40network
-rwxr-xr-x 1 root root  675 Nis 29 00:09 S49ntp
-rwxr-xr-x 1 root root  476 Nis 29 00:05 S50lighttpd
-rwxr-xr-x 1 root root  530 Nis 29 00:10 S50sshd
-rwxr-xr-x 1 root root 1088 Nis 28 23:43 S80dhcp-relay
-rwxr-xr-x 1 root root 1090 Nis 28 23:43 S80dhcp-server
-rwxr-xr-x 1 root root 1421 Nis 29 00:33 S80tftpd-hpa
-rwxr-xr-x 1 root root  494 Nis 28 23:12 S99at
```

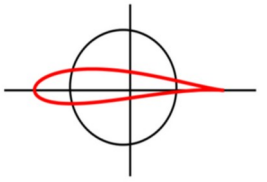



/etc/init.d/rc.S

```
#!/bin/sh

for i in /etc/init.d/S??*
    do
    $i start
    done
```

/etc/init.d/rc.K

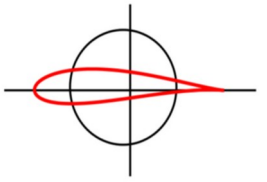
```
#!/bin/sh

for i in $(ls -r /etc/init.d/S??*)
    do
    $i stop
    done
```

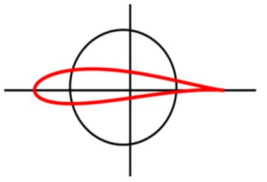
7. /etc/init.d/S??* iskeleti

```
#!/bin/sh
#

case "$1" in

  start)
     echo "Starting command..."
     command_start
     ;;

  stop)
     printf "Stopping command..."
     command_stop
     ;;

  restart|reload)
     "$0" stop
     "$0" start
     ;;

  *)
     echo "Usage: $0 {start|stop|restart}"
     exit 1
esac

Exit $?
```

-- BİTTİ --



EKLER



EK 1, dd, device dump, device destroy

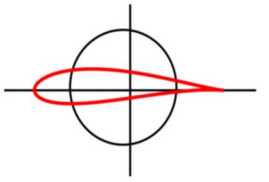
Bir dosyayı okur ve başka bir dosyaya kopyalar.

cp komutundan farklı olarak mount edilmemiş cihazları, blok blok okuyabilir.

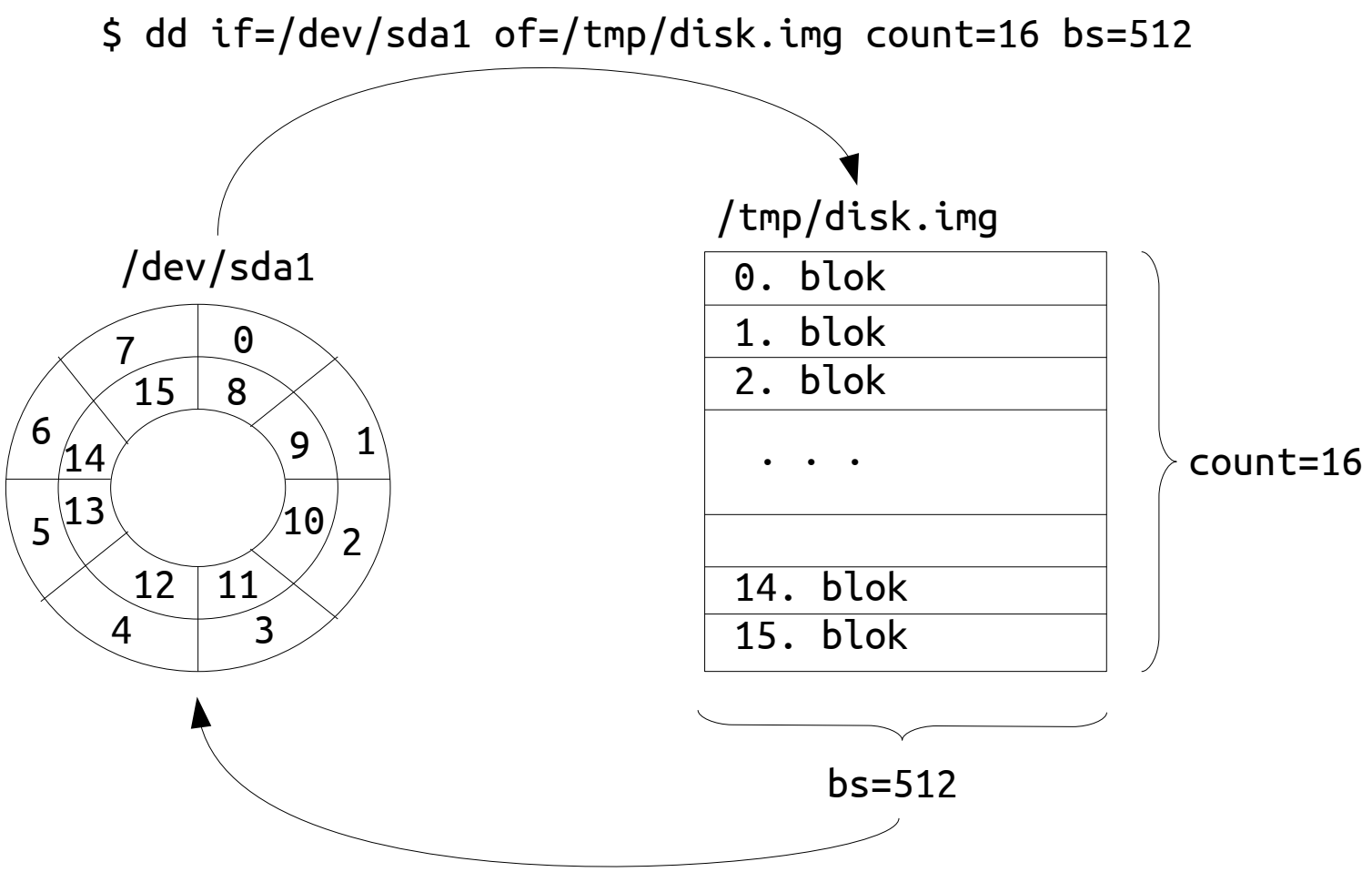
cp komutu, sadece mount edilmiş dosya hiyerarşisini takip ederek çalışabilir.

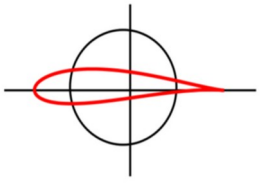
```
$ dd if=/dev/zero of=riscv_disk bs=1M count=64    # komutunun sözde kodu ...
```

```
dd(){
    if= open("/dev/zero", O_RDONLY);    // if=/dev/zero
    of= open("riscv_disk", O_WRONLY);   // of=riscv_disk
    char bs[1024*1024];                 // bs=1M
    count= 64;                          // count=64
    for(i= 0; i< count; i++){
       n= read(if, bs, sizeof(bs));     // if'den bs[] içine n<=1M adet 0 oku
       write(of, bs, n);                // okunan sıfırları of dosyasına yaz
       }
}
```

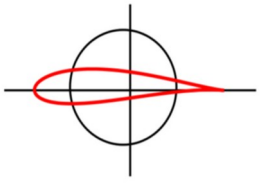
EK 1, devam


$ dd if=/dev/sda1 of=/tmp/disk.img count=16 bs=512
/dev/sda1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
/tmp/disk.img
0. blok
1. blok
2. blok
. . .
14. blok
15. blok
count=16
bs=512
$ dd if=/tmp/disk.img of=/dev/sda1 count=16 bs=512

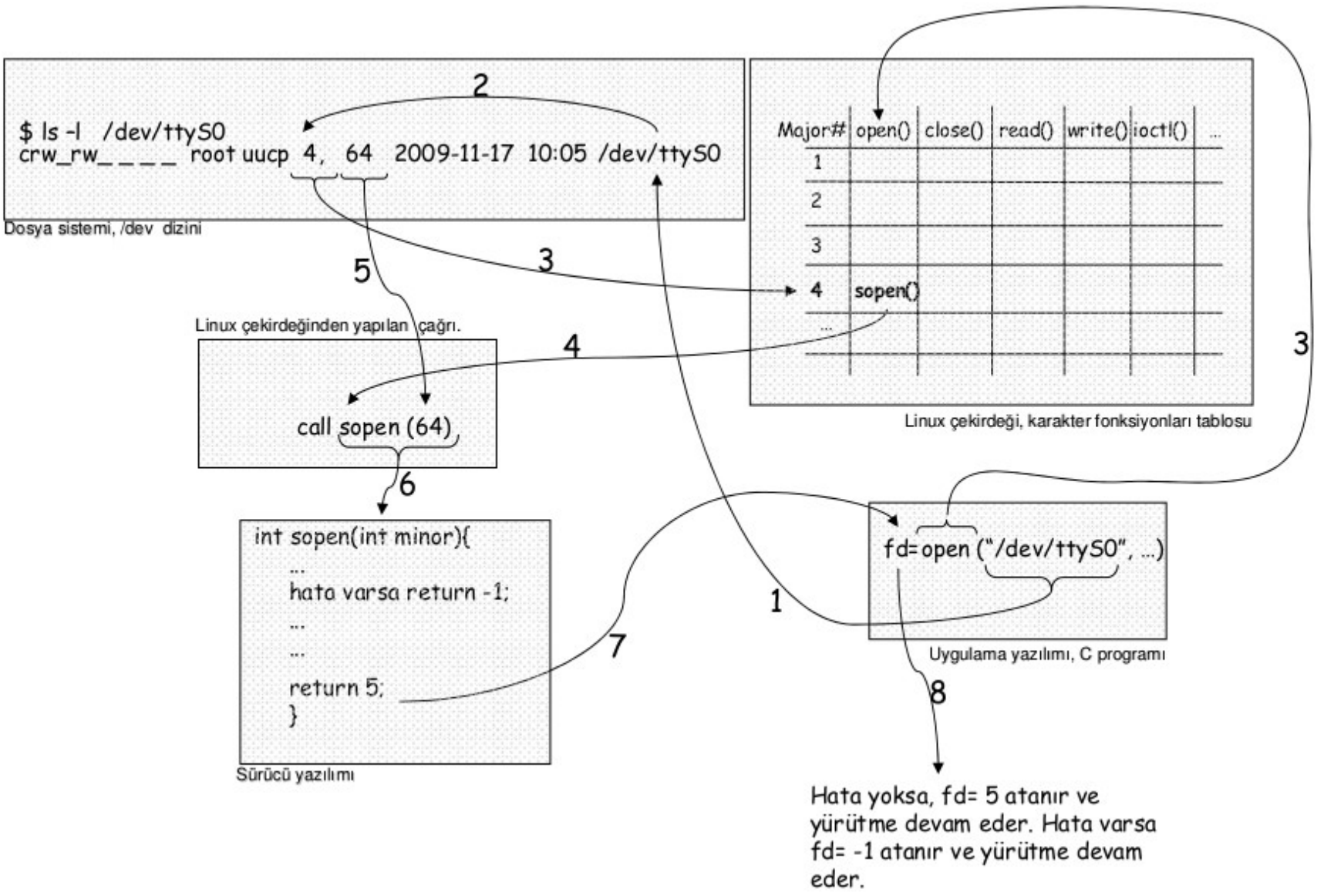


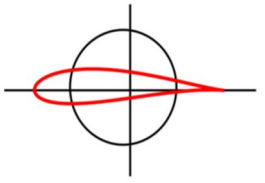
EK 2, Character Device Functions Switch Table

	Major#	open()	close()	read()	write()	ioctl()	...
bellek aygıtları	1						
pseudo tty slave aygıtları	2						
pseudo tty aygıtları	3						
tty aygıtları	4	**sopen()**	**sclose()**	**sread()**	**swrite()**	**sioctl()**	
diğer aygıtlar	...						

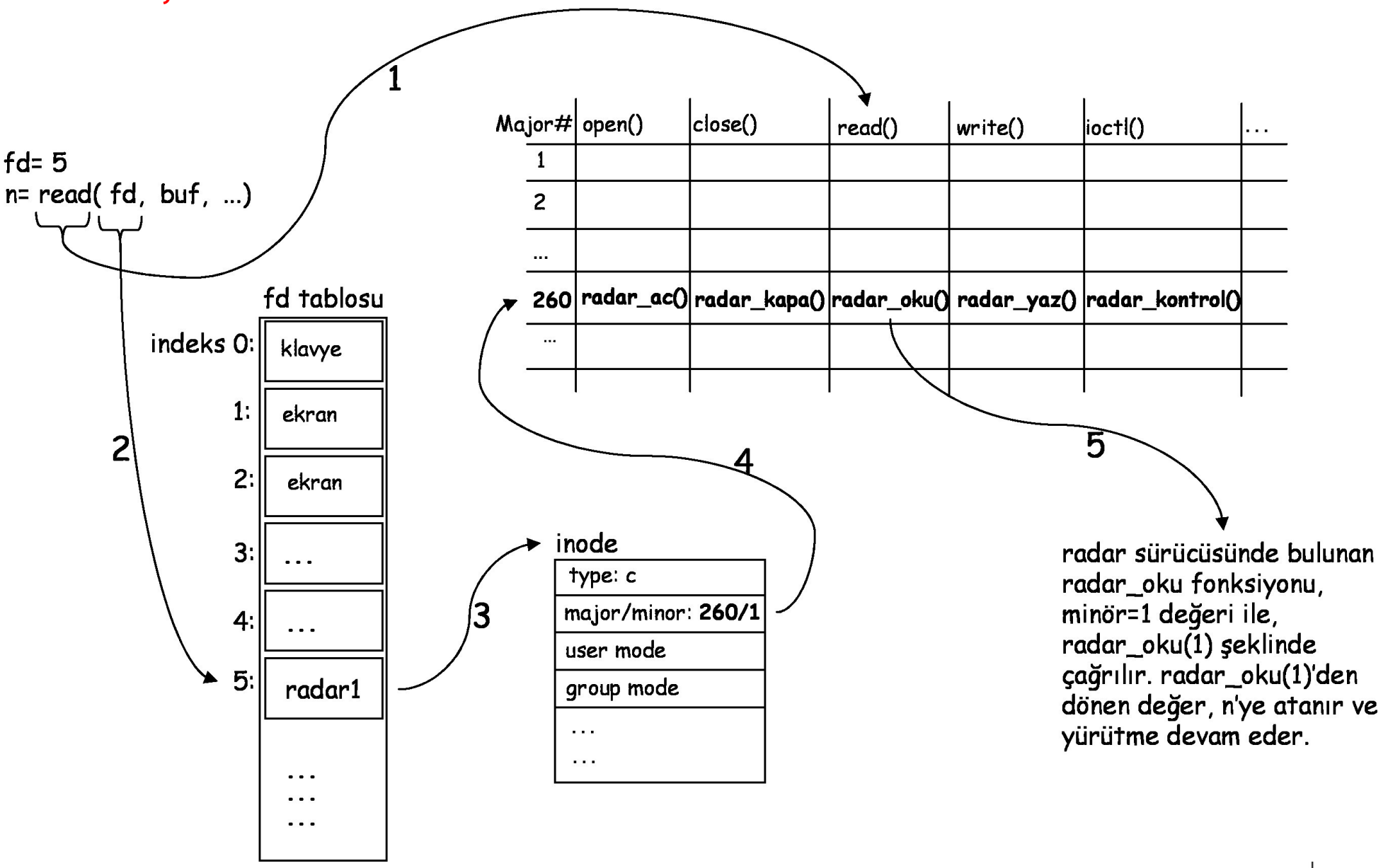


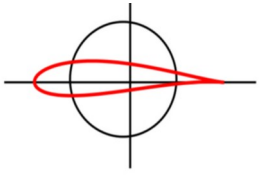
EK 3, VFS Open


$ ls -l /dev/ttyS0
crw_rw____ root uucp 4, 64 2009-11-17 10:05 /dev/ttyS0
Dosya sistemi, /dev dizini
Major# open() close() read() write() ioctl() ...
1
2
3
4 sopen()
...
Linux çekirdeğinden yapılan çağrı.
call sopen (64)
int sopen(int minor){
...
hata varsa return -1;
...
...
return 5;
}
Sürücü yazılımı
fd=open ("/dev/ttyS0", ...)
Uygulama yazılımı, C programı
Linux çekirdeği, karakter fonksiyonları tablosu
1
2
3
4
5
6
7
8
Hata yoksa, fd= 5 atanır ve yürütme devam eder. Hata varsa fd= -1 atanır ve yürütme devam eder.

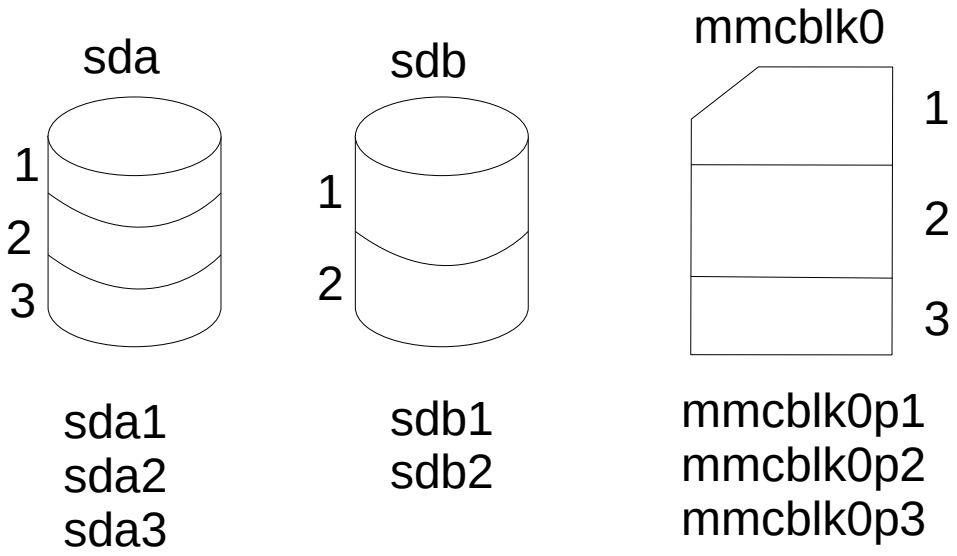


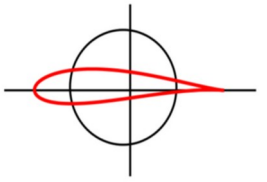
EK 4, VFS Read


fd= 5
n= read(fd, buf, ...)
1
2
3
4
5
Major# open() close() read() write() ioctl() ...
1
2
...
260 radar_ac() radar_kapa() radar_oku() radar_yaz() radar_kontrol()
...
fd tablosu
indeks 0: klavye
1: ekran
2: ekran
3: ...
4: ...
5: radar1
...
...
...
inode
type: c
major/minor: 260/1
user mode
group mode
...
...
radar sürücüsünde bulunan radar_oku fonksiyonu, minör=1 değeri ile, radar_oku(1) şeklinde çağrılır. radar_oku(1)'den dönen değer, n'ye atanır ve yürütme devam eder.



EK 5, Disklerin İsimlendirilmesi

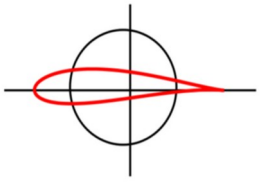




EK 6, /dev/null

Linux sisteminin çöp tenekesidir

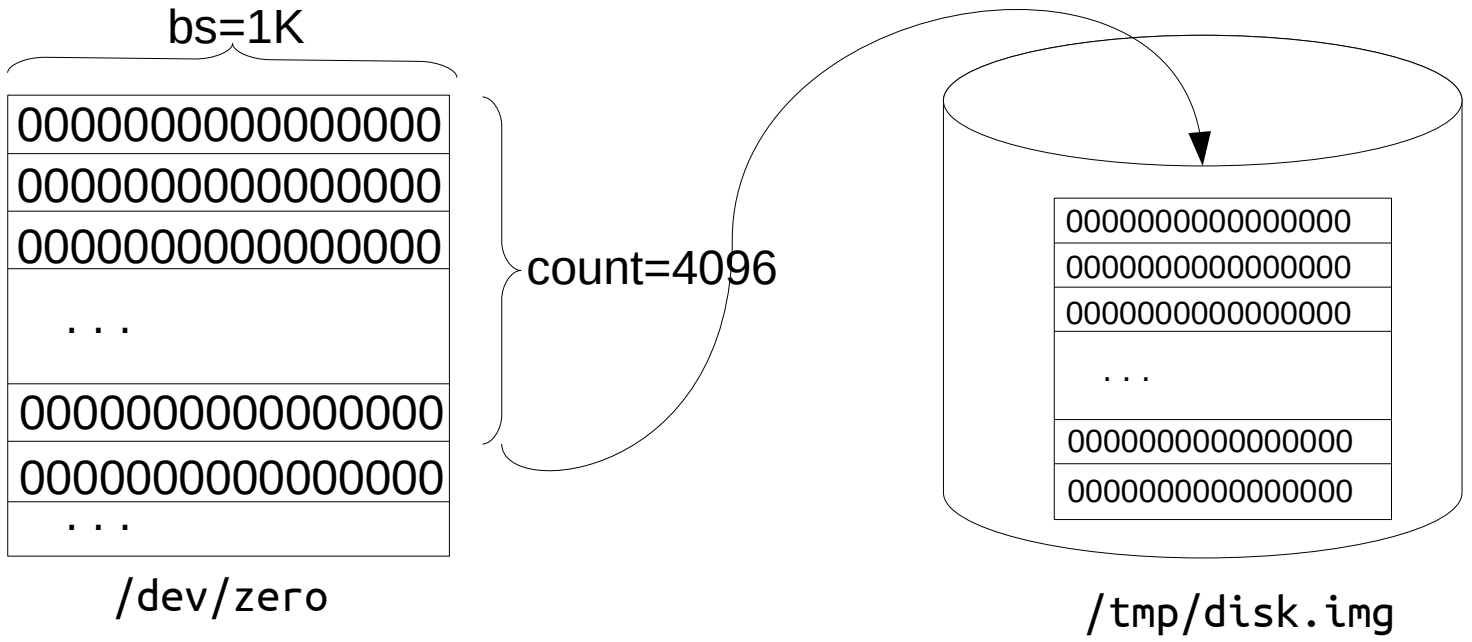
```
$ nohup prog 1>/dev/null 2>/tmp/prog.err &
$ dd if=/dev/sda of=/dev/null
```

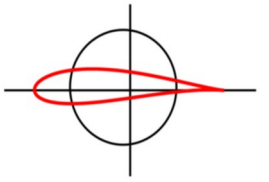


EK 7, /dev/zero

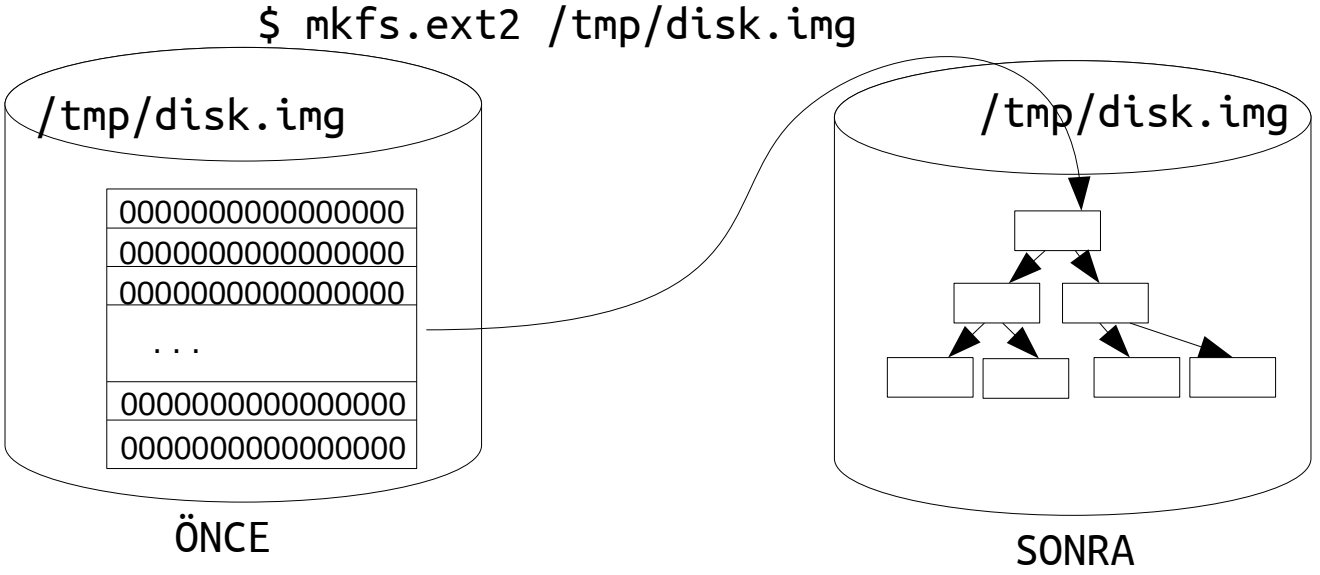
İçinde sonsuz adet “sıfır” bulunan bir dosyadır.

```
$ dd if=/dev/zero of=/tmp/disk.img bs=1K count=4096
```

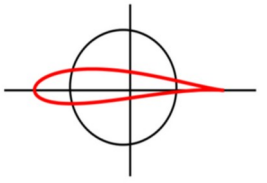




EK 8, loop device




```
$ mkfs.ext2 /tmp/disk.img
$ mkdir /mnt/sanal.disk
$ mount -o loop /tmp/disk.img /mnt/sanal.disk
...
$ umount /mnt/sanal.disk
$ gzip disk.img
```

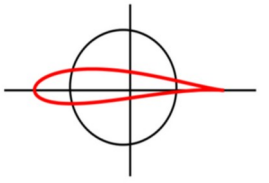
EK 9, /dev/random /dev/urandom

/dev/random cihazı sistemdeki entropi değişimine göre keyfi sayı üretir.

Sistemde entropi değişikliği yoksa sayı üretilmez. Bundan dolayı üzerinde iş yapılmayan makinelerde sayıların üretilme hızı çooook yavaştır. Fakat nitelikli keyfi sayılar üretilir.

/dev/urandom cihazı hızlı keyfi sayı üretir. Üretilen sayılar, keyiflik bakımdan çok da nitelikli değildir.

```
$ dd if=/dev/random

$ cat /dev/random | hexdump -C

$ dd if=/dev/urandom of=/dev/sdc   # DENEMEYİN  !!!
```

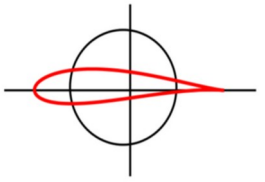


EK 10, /dev/full

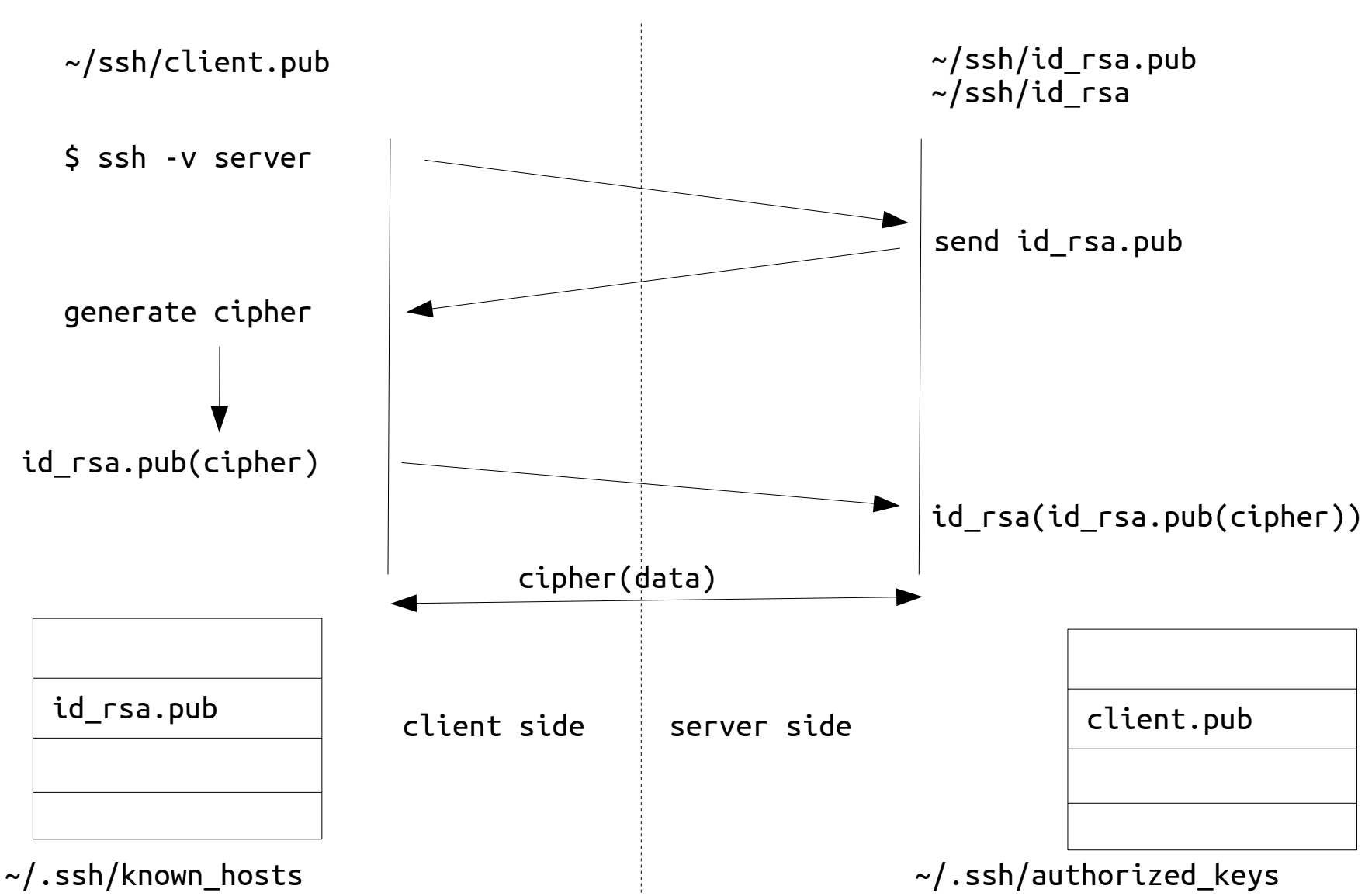
Bu cihaz tam dolu bir cihazı temsi eder.
Uygulama programlarında “tam dolu dosya” testi için kullanılabilir.

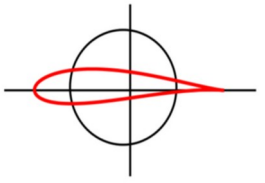
Bu cihazdan veri okunabilir ama yazılamaz.
Veri yazılacağı zaman cihaz dolu veya dosya dolu hatası alınır.

```
$ echo 123 > /dev/full
$ echo $?
```

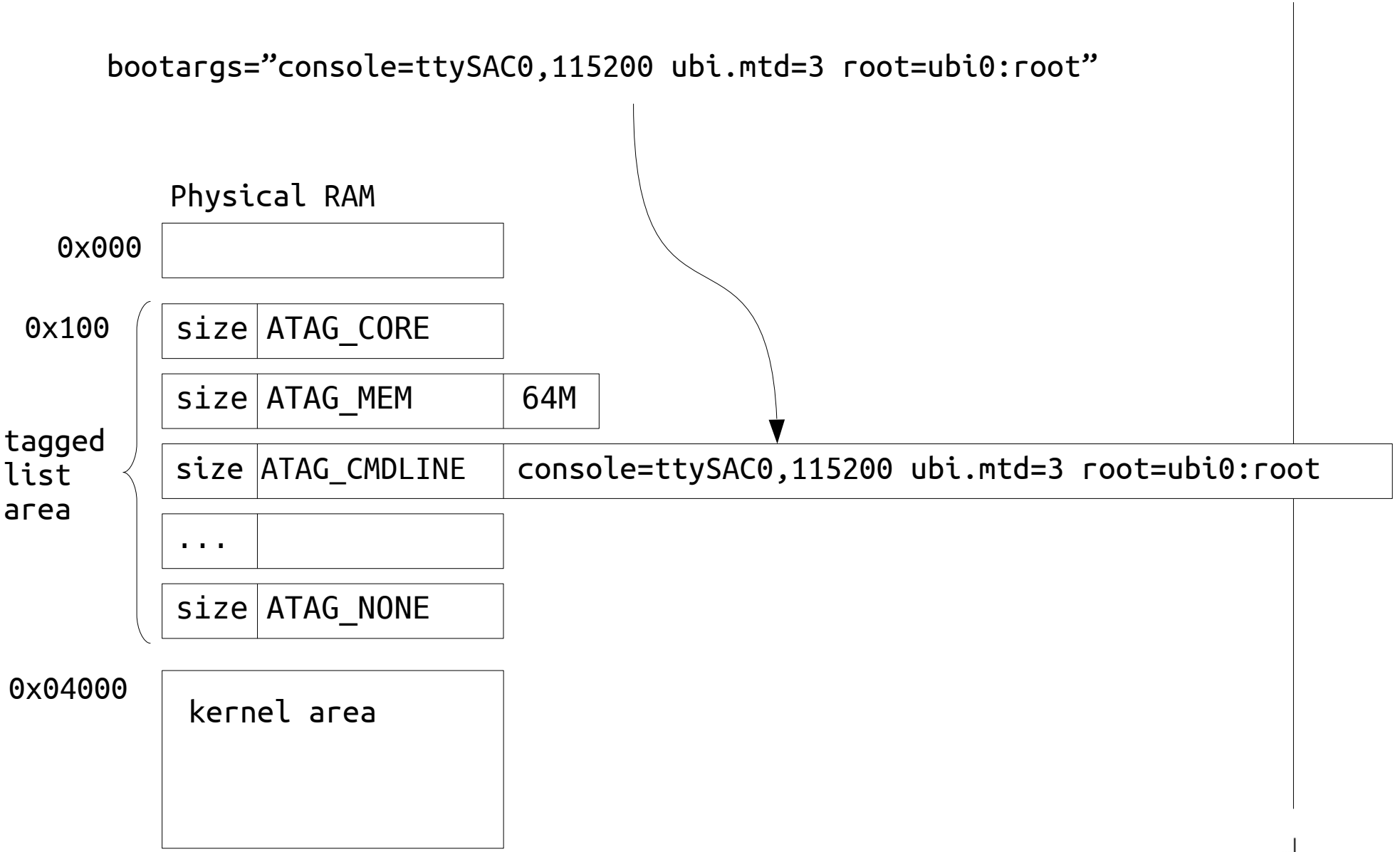


EK 11, ssh

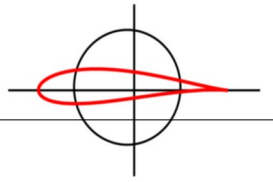

~/ssh/client.pub
~/ssh/id_rsa.pub
~/ssh/id_rsa
$ ssh -v server
send id_rsa.pub
generate cipher
id_rsa.pub(cipher)
id_rsa(id_rsa.pub(cipher))
cipher(data)
id_rsa.pub
client side
server side
client.pub
~/.ssh/known_hosts
~/.ssh/authorized_keys



EK 12, Kernel Parameters Area, Single Linked List



UCanLinux.Com



EK 12

Parametre listesi fiziksel RAM belleğin başlangıç adresinden sonra, tam 0x100 adresinden başlar.

Parametre listesinin ilk elemanı CORE ve son elemanı NONE olmak zorundadır.

MEM zorunludur.

Diğer etiketlerin hiç biri olmayabilir.

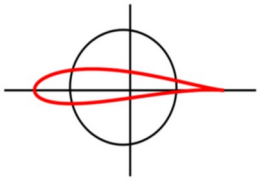
Parametre listesi 0x4000 adresini geçmemelidir.

Linux gözü kapalı biçimde 0x4000 adresinden sonrasını kullanmaya başlar, parametreler ezilebilir.

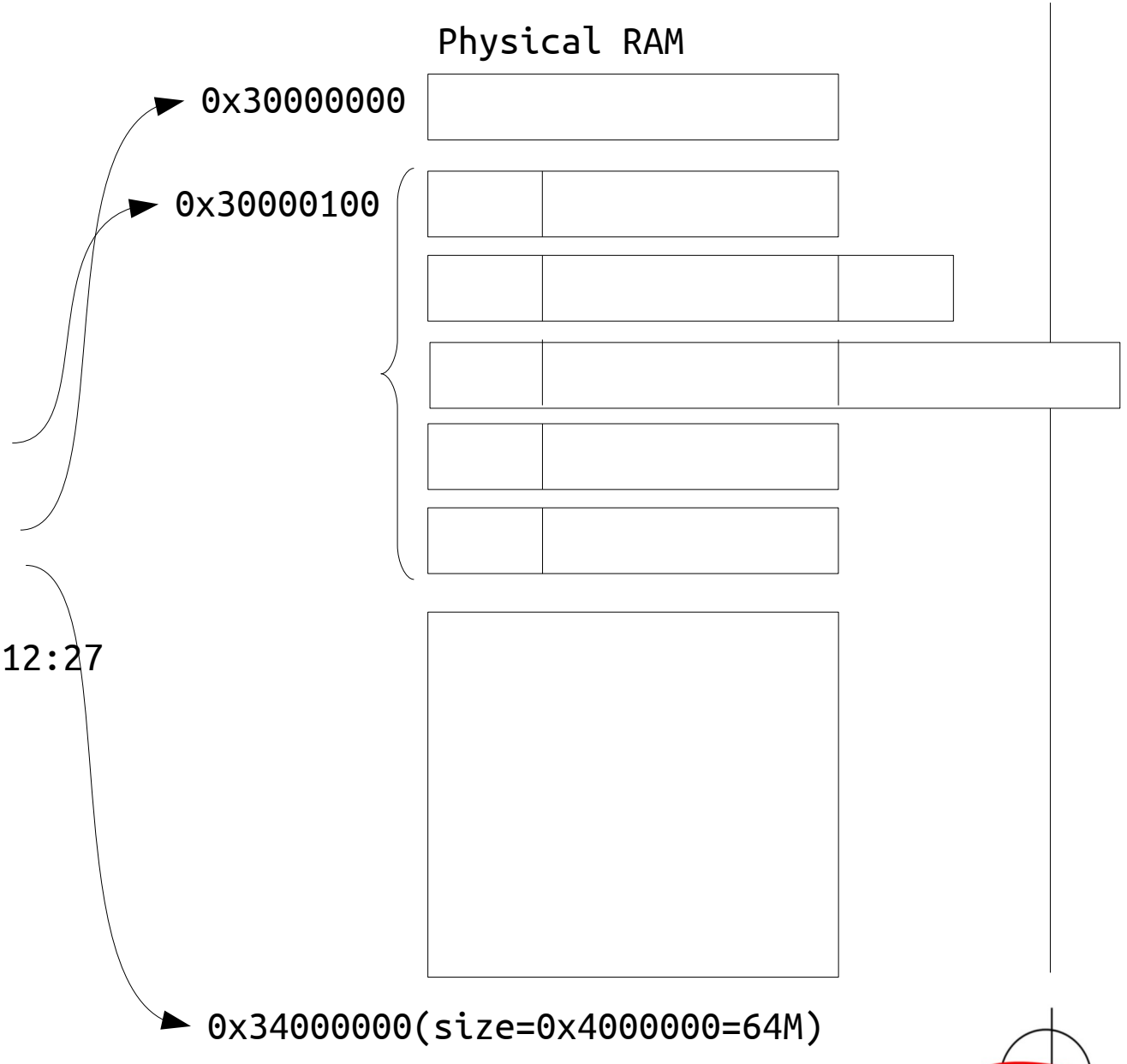
Parametreler u-boot tarafında bootargs ile,

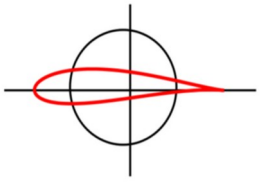
çekirdek derlemesinde “boot parameters” ile,

modül yüklerken modül parametreleri şeklinde verilebilir.

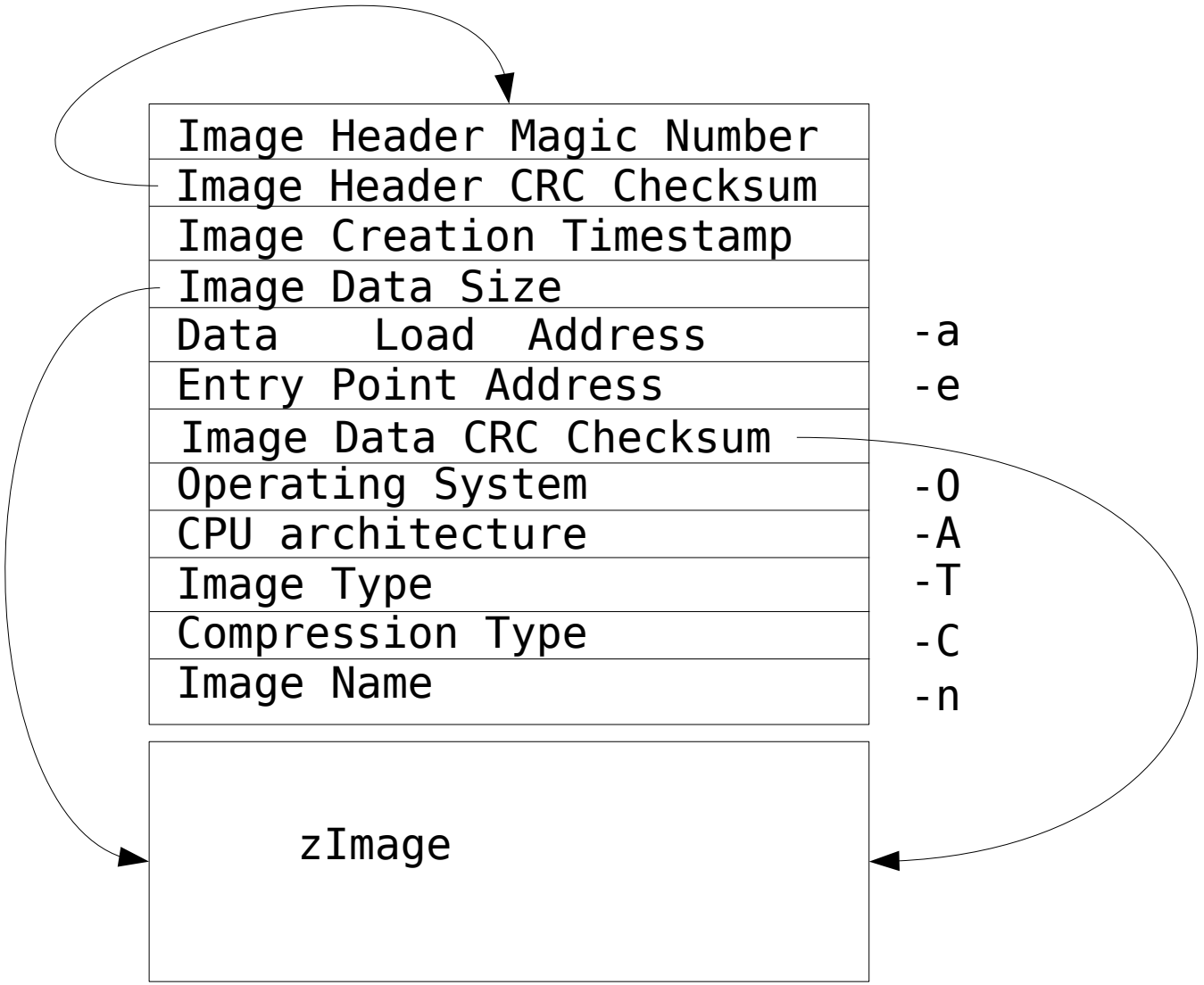


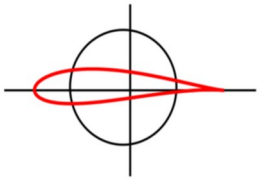
```
MINI2440 # bdinfo
arch_number = 0x000007CF
env_t       = 0x00000000
boot_params = 0x30000100
DRAM bank   = 0x00000000
-> start    = 0x30000000
-> size     = 0x04000000
ethaddr     = 08:08:11:18:12:27
ip_addr     = 10.0.0.111
baudrate    = 115200 bps
```





EK 13, U-Boot image header





EK 14, Eksik programların tamamlanması

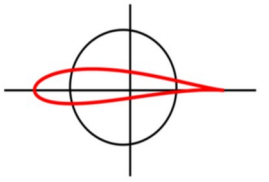
Busybox, 400 civarında unix komutu sağlar. Bu komutların dışında ihtiyaç olan program veya kütüphaneler farklı yollardan temin edilebilir.

En temel yöntem, eksik programın kaynak kodunun indirilip cross derlenmesidir. Ufak programlar veya kütüphaneler için kolaylıkla yapılabilir.

Bu yöntem yeterli gelmezse, aynı makinede kurulabilen başka bir dağıtımdan, istenilen kütüphane veya programlar kopyalanabilir. Örneğin raspian dağıtımında bulunan gerekli her bir dosya, kendi sistemimize kopyalanabilir. Buradaki temel kısıt, toolchain'lerin aynı olmasıdır.

Bazen aşırı kütüphane bağlılığı olabilir. Ya da mevcut programın başka bağımlılıkları olabilir. Bu durumda en kesin çözüm buildroot kullanmaktdır. BR ile istenen paketler seçilir ve kurulur. Sonra rootfs.tar içinden istenen paketler kendi sistemimize kopyalanır.

Gömülü sisteme klasik paket yöneticisi kurmak pek mantıklı değildir. Gömülü sistemler, standard dağıtımlarda olduğu gibi güncellenmezler. Her gömülü sistem kendi yapısına uygun bir güncelleme mekanizması geliştirmelidir.

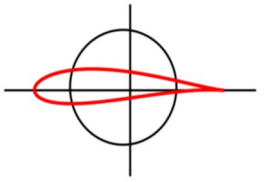


EK 15, Standard bir paketin native/cross derlenmesi

Linux paketleri genelde “configure/make/make install” ile derlenir.

Örnek, zlib paketinin derlenmesi

```
$ wget https://zlib.net/current/zlib.tar.gz
$ tar xvf zlib.tar.gz
$ mkdir /tmp/RootFS
$ cd cd zlib-1.3.1
$ ./configure --prefix=/tmp/RootFS  # native
$ CROSS_PREFIX=arm-linux-gnueabihf-  ./configure -prefix=/tmp/RootFS  # cross
$ make install

$ cd /tmp/RootFS
$ ls -l
$ tree .

$ cd lib/pkgconfig
$ ls
$ more zlib.pc
$ export PKG_CONFIG_PATH=/tmp/RootFS/lib/pkgconfig
$ pkgconf --cflags --libs zlib
```

EK 16, pkg-config

Yüklü kütüphaneler hakkında bilgi temin eder.

Bir kütühane kullanılacak, ama
  include dosyaları nerede?
  kütühaneler hangi dizinde yüklü?
  kütüphanenin adı ne?
  farklı bir dağıtımda bütün bu bilgiler nasıl sabit kalacak?

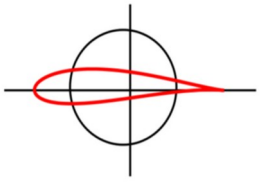
Bütün bu sorunları pkg-config çok şık bir biçimde çözer.
Esas amaç standard dışı dizinlerde bulunan kütüphaneler hakkında bilgi edinmektir.

Daha çok derleme aşamasında kullanılan -I bilgisi ve link aşamasında kullanılan
-L ve -l bilgisi elde edilir.

Bunun dışında kütüphane ismi ve tanımı, kütüphanenin URL şeklindeki kaynağı, sürüm numarası gibi pek çok meta bilgi pkg-config tarafından sunulur.
Bu bilgilerin bütünü .pc uzantılı bir dosyadan temin edilir.

örnek

libxml2 kütüphanesi'nini include dosyası bir dağıtımda /usr/local/xml, farklı bir dağıtımda /opt/include içinde bulunsun.



EK 16

Bu durumda Makefile yazarken bütün bu dizin bilgilerine sahip olmak gerekir. Ama pkg-config bu bilgileri kuruluş sırasında yaratılan .pc dosyasıdan elde eder.

```
$ gcc -O2 -Wall test.c -o test -I/usr/local/xml

# yerine aşağıdaki ifade daha geneldir.

$ gcc -O2 -Wall test.c -o test `pkg-config --cflags xml`
```

Tabii ki libxml için her iki dağıtımda da .pc dosyasının varlığı kabul edilmektedir. Her kütüphanenin .pc dosyası olmayabilir.

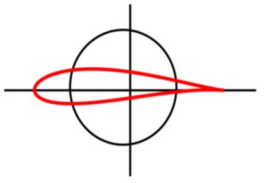
.pc dosyaları genelde aşağıdaki dizinlerde aranır.

/usr/lib/pkgconfig,
/usr/share/pkgconfig
/usr/local/lib/pkgconfig
/usr/local/share/pkgconfig

Ubuntu da
/usr/lib/x86_64-linux-gnu/pkgconfig

Bu dizinler dışındaki .pc dosyalarının yerleri **PKG_CONFIG_PATH** çevre değişkeni ile verilebilir.

.pc dosyaları standard yerler dışında ayrıca burada da ararır.



EK 16

```
$ pkg-config --list-all

$ pkg-config --cflags python
-I/usr/include/python2.7 -I/usr/include/x86_64-linux-gnu/python2.7

$ pkg-config --libs python
-lpython2.7

$ pkg-config --cflags --libs python
-I/usr/include/python2.7 -I/usr/include/x86_64-linux-gnu/python2.7 -lpython2.7
```

Nasıl kullanılır?

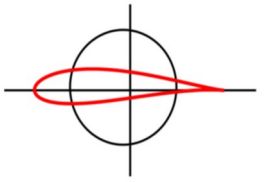
```
CFLAGS  = -g -Wall
CFLAGS += `pkg-config --cflags siemens_iot`

LDFLAGS += $(pkg-config --libs siemens_iot)   # veya
LDFLAGS += `pkg-config --libs siemens_iot`

# Doğrudan kullanım
$ gcc `pkg-config --cflags --libs siemens_iot` -Wall -O2 main.c -o main.o
```

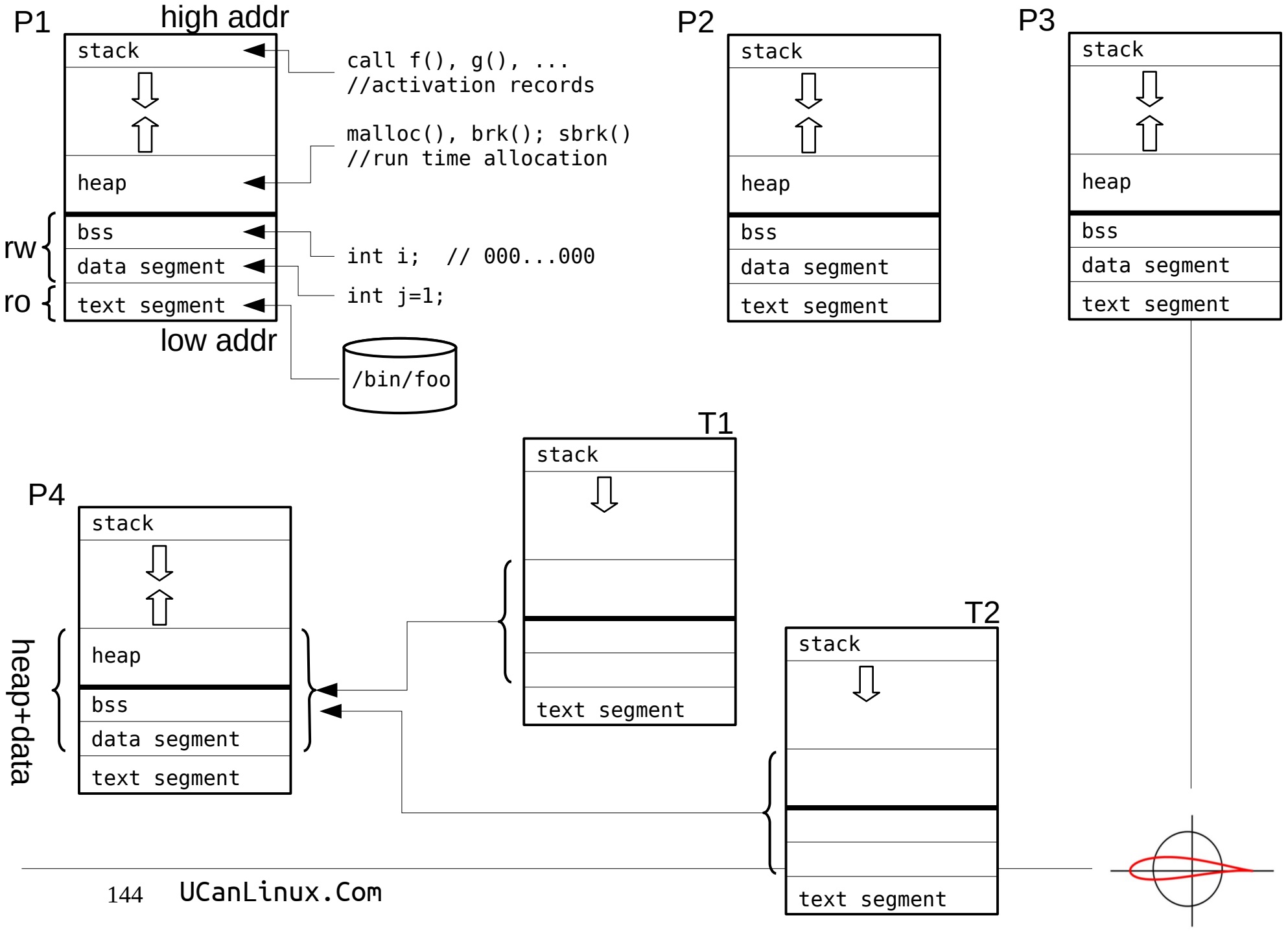
Esas amaç dağıtımlardan bağımsız Makefile kurmaktır.

Kütüphane isminin sadece bir kısmı biliniyorsa grep ile arama yapılabilir.

```
$ pkg-config --list-all | grep xml
```

Tam bir örnek için bak zlib derlemesi, Ek 16




P1
high addr
stack
heap
bss
data segment
text segment
low addr
rw
ro
call f(), g(), ...
//activation records
malloc(), brk(); sbrk()
//run time allocation
int i; // 000...000
int j=1;
/bin/foo
P2
stack
heap
bss
data segment
text segment
P3
stack
heap
bss
data segment
text segment
T1
stack
text segment
T2
stack
text segment
P4
stack
heap
bss
data segment
text segment
heap+data

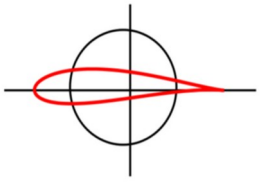
EK 17

32 bit'lik makineler için 4GB'lık sanal bir bellek ayrılır.
Genel yapı verilmiştir.
Segment adreslerini keyfi kaydırma, mmap alanı ve linux için en tepede ayrılan alanlar ayrıca gösterilmemiştir.

```
int p[4096];
int main(){
    return 0;
}
```

```
$ gcc prog.c -o prog    // int p[4096];
$ size prog
   text     data      bss      dec      hex  filename
   1418      544    16416    18378     47ca  prog
$ ls -l prog
-rwxrwxr-x 1 nazim nazim 16488 Nis 30 05:52 prog

$ gcc prog.c -o prog   // int p[4096]= {3};
$ size prog
   text     data      bss      dec      hex  filename
   1418    16944        8    18370     47c2  prog
$ ls -l prog
-rwxrwxr-x 1 nazim nazim 32888 Nis 30 05:52 prog
```

bss: block started by symbol or better save space :)



EK 18, Threads

Ortak veri alanı (data and heap segments) olan proseslere thread denir.

Ek18'de verilen şekilde 4 adet proses vardır.
Her prosesin bütün segmentleri bağımsızdır.

P4 prosesi, T1 ve T2 ile gösterilen, 2 adet thread yaratmıştır.
P4 ile T1 ve T2'nin data ve heap alanları paylaşımlıdır, ortaktır.

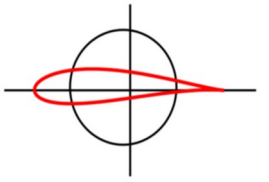
Linux'da thread kütüphanesinin adı pthread'dir.
Derleme yaparken -lpthread ile kütüphane ismi açıkça verilmelidir.

Thread'lerle ilgili config bilgileri aşağıdaki gibi elde edilebilir.

```
$ getconf -a | grep THREAD
```

Karşısı boş olan satırlar, undefined kabul edilir.
Parametre ismi doğrudan da verilebilir.

Linux'da thread'lerle prosesler hemen hemen aynıdır.
Thread'lere light weight process de denir.



EK 18

Prog 1 joinable thread
$ gcc prog1.c -o prog1 -lpthread

Prog 2 Örnek program ancak sinyal ile dudurulabilir.
Sistem komutları ile thread'leri incele.

```
$ ./prog2 &
[1] 3914
tpid 3915
tpid 3916
```

Arka planda çalışan işleri gör.

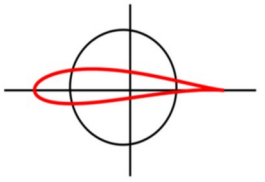
```
$ jobs
[1]+  Running                 ./prog2 &
```

pstree ile proses ağacını gör

```
$ pstree -p
prog2---2*[{prog2}]
```

pstree PID ile proses ağacını gör

```
$ pstree -p 3914

prog2(3914)---{prog2}(3915)
            |
            ` {prog2}(3916)
```

EK 18

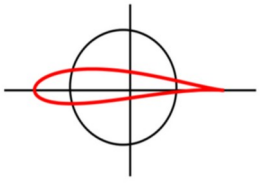
ps ile pid/ppid/tpid incele.

```
$ ps -eT | grep prog
 3914  3914 pts/8    00:00:00 prog2
 3914  3915 pts/8    00:00:00 prog2
 3914  3916 pts/8    00:00:00 prog2
```

top ile incele

```
$ top -H -p `pidof prog2`
top - 10:39:13 up  1:14,  1 user,  load average: 0,33, 0,25, 0,24
Threads:   3 total,   0 running,   3 sleeping,   0 stopped,   0 zombie
%Cpu(s): 13,6 us,  2,3 sy,  0,2 ni, 77,0 id,  6,8 wa,  0,0 hi,  0,1 si,  0,0 st
KiB Mem :  3932584 total,   468508 free,  1456856 used,  2007220 buff/cache
KiB Swap:  1999868 total,  1991736 free,     8132 used.  1842380 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 3914 nazim     20   0   88448    716    632 S  0,0  0,0   0:00.00 prog2
 3915 nazim     20   0   88448    716    632 S  0,0  0,0   0:00.00 prog2
 3916 nazim     20   0   88448    716    632 S  0,0  0,0   0:00.00 prog2
```

Not: -b -n 1 ile copy/paste yapıldı.



EK 18

/proc ile inceleme

```
$ cd /proc/3914/task/

$ ls -l
total 0
dr-xr-xr-x 7 nazim nazim 0 Ağu 20 10:26 3914
dr-xr-xr-x 7 nazim nazim 0 Ağu 20 10:26 3915
dr-xr-xr-x 7 nazim nazim 0 Ağu 20 10:26 3916
```

Farklı bir dizinden jobs komutu verilirse wd gösterilir.

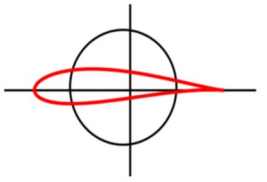
```
$ jobs
[1]+  Running  ./prog2 &  (wd: /nk/workspace/projects/linux.egitimi)
```

Durdur

```
$ kill %   <ENTER> <ENTER>
[1]+  Terminated   ./prog2
```

Kontrol et

```
$ pidof prog2
$ ps -eT | grep prog
```



joinable, detachable

Bütün thread'ler aksi söylenmedikçe joinable yaratılır.
joinable thread yaratan proses, mutlaka join() ile dönüşü beklemelidir.

Joinable thread'ler çağıran yere veri döndürebilir.

Joinable thread'lerde dönüşten sonra kullanılan kaynaklar serbest bırakılmaz.
Bellek kullanımı çok iyi analiz edilmelidir.

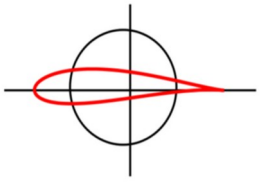
main() programı durursa, bütün thread'ler durur.
Bundan dolayı main() durmadan önce veya daha evvelinde, joinable thread'ler mutlaka pthread_join() fonksiyonu ile beklenmelidir.

detachable thread'lerin dönüş zamanı belli değildir, join() ile beklenilmez.

Dönüş bitince kaynaklar serbest bırakılır.
Çağıran yere veri döndürülemez.

Taşınabilirlik için bütün thread'ler joinable olmalıdır.
Posix bunu zorlar.
Her thread kütüphanesi detachable thread'leri desteklemeyebilir.

Joinable thread yaratılıp, join ile dönüş bilgisi elde edilmezse, bellek sızıntısı meydana gelebilir.



EK 18

Prog 3 join olmazsa ne olur?

```
$ ./prog3
4430: p[1]= 2   usleep(313208);
4429: p[1]= 1   usleep(670234);
4430: p[2]= 4   usleep(302413);
main bitti.
```

Thread'ler halen işlemekte iken, main() durunca, bütün thread'ler de durur.
join zorunludur!

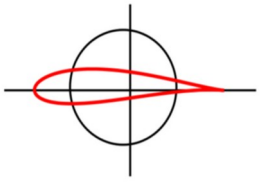
Prog 4 detachable thread örneği

```
$ ./prog4
main bitti.
```

thread'ler yürümeye vakit bulamadı!

Sona sleep(3) ekle, tekrar çalıştır.

```
$ ./prog4

  4895: p[1]= 1   usleep(599317);
  4896: p[1]= 2   usleep(680083);
  4895: p[2]= 2   usleep(42473);
  4895: p[3]= 3   usleep(245861);
  4896: p[2]= 4   usleep(855834);
  4895: bitti, toplam 6
  4896: p[3]= 6   usleep(174491);
  4896: p[4]= 8   usleep(586573);
  4896: bitti, toplam 20
main bitti.
```



EK 18

Sorunsuz çalışır.
Fakat main() program yeteri kadar beklemeli, erken bitmemelidir.

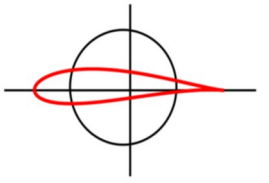
Kod sonuna pthread_exit() ekle, tekrar çalıştır.
$ ./prog4

```
$ ./prog4
  5391: p[1]= 2   usleep(688879);
  5390: p[1]= 1   usleep(639670);
  5390: p[2]= 2   usleep(829744);
  5391: p[2]= 4   usleep(38427);
  5391: p[3]= 6   usleep(85821);
  5391: p[4]= 8   usleep(170444);
  5391: bitti, toplam 20
  5390: p[3]= 3   usleep(132582);
  5390: bitti, toplam 6
```

main'in bitiş lafı gelmez.
Fakat bütün thread'lerin bitmesi beklenir.

Ototmatik join gibi düşünülebilir.
Ama thread'lerden dönüş bilgisi gelmez.

Eğer detach thread mevcutsa ya main() yeteri kadar beklemeli ya da
main()'den pthread_exit() ile çıkılmalıdır.



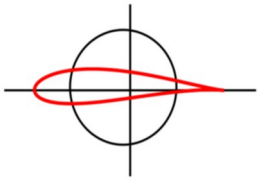
EK 18

Klasik mutex örneği

Kritik alan girişi ve çıkışında kullanılanılır.
Birden fazla mutex varsa sıraya dikkat edilmelidir.
Kritik alandan mümkün olduğu kadar çabuk çıkılmalıdır.
lock/unlock işlemleri maliyetlidir, context-swtich gerektirir.
Aşırı kullanılmından kaçınılmalıdır.
Çoklu işlemci varsa spin-lock ile context-switch engellenebilir.

Mutex için örnek kod parçası,

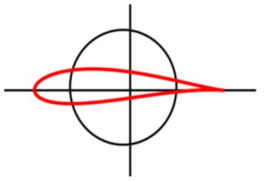
```
pthread_mutex_t count_mutex;
long count;

void increment_count(){
   pthread_mutex_lock(&count_mutex);
   count= count + 1;
   pthread_mutex_unlock(&count_mutex);
}

long get_count(){
   long c;

   pthread_mutex_lock(&count_mutex);
   c = count;
   pthread_mutex_unlock(&count_mutex);

   return (c);
}
```

EK 18

Prog 5 Thread'e giden sinyal ana programa gelmiş kabul edilir.

```
$ ./prog5 &
[1] 5935
tpid 5937
tpid 5936

$ pstree -p 5935

prog5(5935)---{prog5}(5936)
            |
            `-{prog5}(5937)

$ kill -TERM 5936
sinyal geldi Terminated
[1]+  Exit 1                  ./prog5

$ jobs
```



EK 18

Öneriler

thread'ler mümkünse kullanılmamalıdır.
Proses ve IPC tabanlı çözümler yetersiz kalırsa thread kullanılmalıdır.

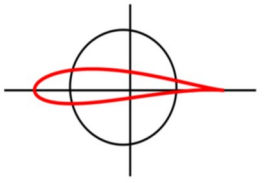
Joinable olmalı ve mutlaka main() çıkışında join() ile bekleme yapılmalıdır.

Joinable thread'lerde kullanılmayan kaynaklar açıkça serbest bırakılmalı,
işletim sisteminin insafına kalınmamalıdır.
Örneğin bir daha hiç kullanılmayacak bir dosya açıkça fclose() ile kapatılmalıdır.
Eğer kapatılmazsa, main()'deki veya sistemdeki ilk exit()'den sonra kapatılma
yapılacak ve
sistem kaynağı boşuna kullanılacaktır.

Thread'deki bir exit() veya aynı sınıftan bir işlem, main()'in olduğu ana proses
dahil, bütün thread'leri kapatır.

Thread'lerdeki sistem çağrıları mutlaka thread-safe olmalıdır.
Birden fazla thread tarafından sorunsuzca kullanılabilen veya reentarnt olan
bütün fonksiyonlar thread-safe'dir.
man sayfalarından her çağrı tek tek incelenmelidir.
man 7 pthreads sayfasında, thread kullanılırken nerlere dikkat edilmelidir bölümü
mutlaka okunmalıdır. Yok yoktur.

Sinyaller bütün thread'lere gönderilir. Sinyallere karşı tedbir alınmalıdır.



EK 18

Kritik bölgeler önceden tespit edilmeli ve mutex ile korumaya alınmalıdır.

Kritik bölgeler aşırı derecede çağrılmamalıdır.
mutex işlemi maliyetli bir işlemdir, çünkü her mutex context-switch gerektirir.

Kritik bölgelerde oyalanmamak gerekir.
Bu bölgelere giriş çıkış süresi çok az olmalıdır.

Kritik bölgelerden çıkarken mutex'ler serbest bırakılmalıdır.
Yoksa bütün program donabilir.

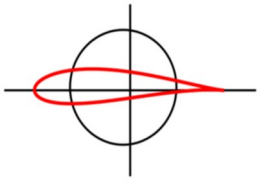
Bir thread başka bir thread'i sonlandırmamalıdır.
Bütün thread'leri ana prosesin yönetmesi daha mantıklıdır.

pthread_cancel() fonksiyonu mümkünse hiç kullanılmamalıdır.
Thread'ler her çalışma adımında güvenle duramazlar.
Thread'ler her zaman return pthread_exit() ile bitmelidir.

Thread'ler prosesler gibi kabuktan hemen durdurulup başlatılamaz.

Multithread programların testleri son derece zordur.

Bellek sızıntılarının tespiti çok sorunludur.



EK 18

Ne zaman kullanılmalı?

Thread'lerin her zaman proseslerden çok daha hızlı başlarlar.

Daha az sistem kaynağı tüketirler, çünkü heap ve data alanları paylaşımlıdır.

Kullanımları göreceli kolaydır ve IPC gerektirmezler.

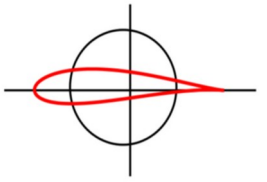
Data alanları ortak olduğu için veri paylaşımı ve ortak veri üzerinde çalışmak aşırı kolaydır.

Şahsi fikrimiz, baldıran zehiridir.

Thread safe fonksiyonlar

Birden fazla thread tarafından, veri bütünlüğü bozulmadan kullanılan fonksiyonlara thread-safe fonksiyon denir.
Bu fonksiyonların listesi posix'de verilmiştir.

```
$ man 7 pthreads
```

EK 19, Statik Kütüphanler

Statik kütüphaneler, aynı amaç için biraraya getirilmiş, arşivlenmiş object kodların bütünüdür.
Program içinde kullanılan kütüphane fonksiyonları, link aşamasında, kütüphaneden program içine alınır.
Her kod içinde statik kütüphaneler tekrar eder.

Kuruluş

```
$ gcc foo.c -c
```

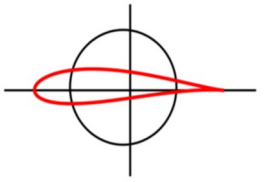
foo.o oluşur.
-c Derle ama link etme demektir.
Yani sadece *.o oluşturur.
İçinde main() olmamlıdır.

ar ile bütün *.o'ları biraraya toplanır.

```
$ ar -rcs fileName.a *.o
```

-r insert object file
-c create
-s write an object-file index into the archive

Alışkanlık gereği statik kütüphanelere .a uzantısı verilir.



EK 19

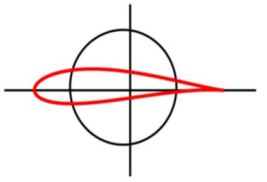
Kütühane ismi sonda verilmelidir.
Önce semboller saklanır sonra aranır.

Arşiv içindeki sembolleri listele

T: text code section

```
$ nm libtest.a

add.o:
0000000000000000 T add

sub.o:
0000000000000000 T sub

mult.o:
0000000000000000 T mult

div.o:
0000000000000000 T div
```

```
$ gcc -o prog1 prog1.c libtest.a
```

EK 19

Sembollerin incelenmesi.

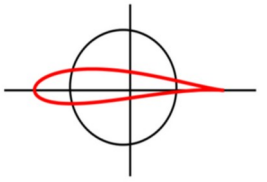
```
$ nm prog1
...
00000000004005f0 T add
...
0000000000400600 T mult
...
```

Bütün kütüphane değil, sadece kullanılan fonksiyonlar çalışabilir koda eklenir.

Kütüphaneler statik veya dinamik derlenebilir.
Dinamik derleme daha geneldir.
Her dinamik kütüphanenin genelde bir de statik kütüphanesi bulunur ama her zaman olmayabilir.

Kullanım

```
$ gcc -o prog1 prog1.c

$ gcc -o prog1 prog1.c -L. -ltest    # veya
$ gcc -Wall -o prog1 prog1.c ./libtest.a

$ ldd prog1  # içinde libtest.a yoktur
```

EK 20, Paylaşımlı Kütüphaneler

Dynamic Shared Objects, *.so

Proses adres alanına sonradan yüklenen kütüphanelerdir.

Çalışan kodun dosyasından ayrı bulunurlar.

Pek çok program tek bir kütüphaneyi ortak kullanabilir.
Bundan dolayı paylaşımlı denir.

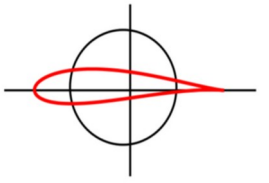
Geriye uyumluluk varsa, kütüphane güncellendiğinde uygulamanın tekrar derlenmesine gerek yoktur.

dlopen API ile yürütme zamanında yüklenip işi bitince tekrar serbest bırakılabilirler.
plugin yöntemi

Çalışabilir bir kodun ihtiyaç duyduğu kütüphaneler ldd veya objdump ile tespit edilebilir.

Uygulama programlarında aşırı tekrar eden yerler paylaşılan kütüphane yapılabilir.

RAM'de fiziksel olarak tek bir yerde bulunurlar.
Ama her programın mantıksal adres uzayı faklıdır.
Yani kodların pozisyondan bağımsız olması gerekir.
Bundan dolayı paylaşılan kütüphaneler PIC seçeneği ile derlenirler.



EK 20

Kullanımı

```
$ make

$ gcc -O2 -Wall -o prog1 prog1.c  # hata verir
```

Kütüphane araması hangi dizinlerde yapıldı?

```
$ gcc -print-search-dirs | grep libraries
```

Kütüphane isimleri libXXX.so şeklindedir ve -lXXX şeklinde derleyiciye verilir.

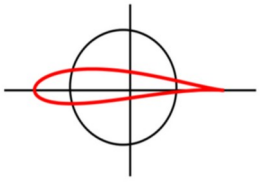
```
$ gcc -o prog1 prog1.c -ltest -L.
$ gcc -O2 -Wall -o prog1 prog1.c /opt/sysprog/lib/libtest.so
$ gcc -O2 -Wall -o prog1 prog1.c ./libtest.so
```

Statik linker kütüphane aramalarını aşağıdaki sırada yapar.
Derleme sırasında verilen -L, en önceliklidir.

```
$ ld --verbose | grep DIR
```

Derleme sorunsuz ama çalışma zamanında bu kütüphane bulunamaz.

```
$ ldd prog1   # not found gör
$ ./prog1     # error
```

EK 20

Çalışma zamanında kütüphaneyi bulabilmesi için **LD_LIBRARY_PATH** tanıtılıp, export edilebilir.

```
$ export LD_LIBRARY_PATH=.    # veya
$ export LD_LIBRARY_PATH=/opt/sysprog/lib

$ ldd prog1  # kütüphane adını gör

$ ./prog1
add(2,3) = 5
mult(2,3)= 6
```

Nereden yüklendiğini açıkça gör.

```
$ strace ./prog1
```

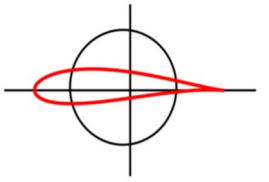
Ya da doğrudan verilir,

```
$ LD_LIBRARY_PATH=/opt/sysprog/lib ./prog1
```

Çalışma zamanında kütüphaneler aşağıdaki sırada aranır

LD_LIBRARY_PATH
/lib, /usr/lib standard kütüphaneler
/etc/ld.so.conf dosyasında, ldconfig ile üretilmiştir.

Pek de yagın olmayan DT_PATH, RPATH, RUN_PATH gibi değişkenler de mevcuttur.



EK 20

yorumlayıcı

```
$ ldd prog1 # veya
$ readelf -l prog1 | grep interpreter
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
```

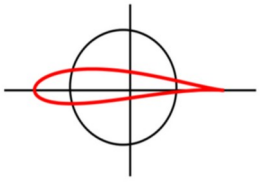
Bu çok özel kütüphaneye run-time linker veya interpreter(yorumlayıcı) denir.
Bu program yürütme zamanında çalışır.

Bir de compile-time linker vardır.
Bu da ld komut ile gerçeklenir ve derleme zamanında, genelde gcc içinde kapalı bir biçimde kullanılır.

Compile-time linker elf kodu içine kullanılacak kütüphanelerin isimlerini yazar, aynı zamanda link aşamasında kütüphanelerin tablolarını kullanır.

Dinamik bağımlı bir program çalışacağı zaman önce program sonra yorumlayıcı yüklenir.

Yorumlayıcı ld.so.cache'i kullanarak, link aşamasında program içinde isimleri gömülü olan dinamik kütüphaneleri bulur ve gerekli adres çözümlemelerini yapar ve programı başlatır.



EK 20

Programın doğrudan bağlı olduğu kütüphanaler readelf ile de bulunabilir.

```
$ readelf -d prog1 |grep NEEDED
0x0000000000000001 (NEEDED)             Shared library: [libtest.so]
0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
```

objdump'da kullanılabilir.

```
$ objdump -p prog1 | grep NEEDED
  NEEDED               /opt/sysprog/lib/libtest.so
  NEEDED               libc.so.6
```

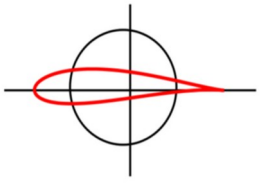
readelf aynı zamanda kütüphaneler için de kullanılabilir.

```
$ readelf -d libtest.so  | grep NEEDED
0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
```

Böylece kütüphanelerin bağlı olduğu kütüphaneler de tespit edilebilir.
ldd bu işi otomatik olarak yapar.

Yürütme zamanında tespit için

```
$ strace -e trace=open ./prog1
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
open("/opt/sysprog/lib/libtest.so", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
add(2,3) = 5
mult(2,3)= 6
+++ exited with 0 +++
```

EK 20

```
$ sleep 100 &
[1] 11521

$ lsof -p 11521 | grep ".so"
sleep   11521 nazim  mem    REG    8,1  1868984  654540 /lib/x86_64-linux-gnu/libc-
2.23.so
sleep   11521 nazim  mem    REG    8,1   162632  654402 /lib/x86_64-linux-gnu/ld-
2.23.so
```

Yorumlayıcılar, yani 64 bit x86 için ld-linux-x86-64.so.2 programı, toolchain içinde bulunur, ve "başka kütüphanelere" bağımlı değildir.

```
$ readelf -d /lib64/ld-linux-x86-64.so.2 | grep NEEDED
```

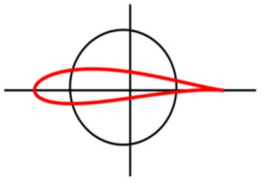
Yorumlayıcı başka kütüphanelere bağımlı olmadığı gibi çok yetenekli bir programdır. Örneğin kütüphanaleri yükelerken, birbirlerine olan bağımlıklarını da tespit edebilir.

Kütüphane sürümleri

```
$ ls /lib/x86_64-linux-gnu/libz*
libz.so      -> libz.so.1.2.8    # linker name, sadece kütüphane istenirse
libz.so.1    -> libz.so.1.2.8    # shared object name, majör uyumluluk
libz.so.1.2.8                    # real name, tam uyumluluk
```

libz.so.Majör.Minör.Patch

Majör geriye uyumluluk yok,	libz.so.3 ile libz.so.2 uyumsuzdur.
Minör geriye uyumluluk mevcut	libz.so.2.x ile libz.so.2.y uyumludur.
Patch her zaman uyumlu	bütün libz.2.5.x'ler uyumludur.



EK 20

```
$ make  # /opt/sysprog/lib/libtest.so'yu kur.
```

Kütüphanede majör bağımlılığın istenmesi

libtest.so.3 gibi bağımlılık istenirse, statik link sırasında libtest.so.3.0.1 dosyasının aslında libtest.so.3 olarak işleneceği soname seçeneği ile belirtilir.

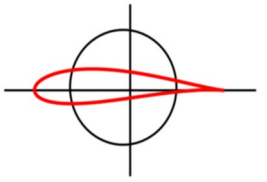
Dosya ismi libtest.so.3.0.1 olmasına rağmen yorumlayıcı libtest.so.3 olarak arama yapacaktır.

Bu durumda libtest.so.3 isimli bir dosya mevcut değildir.
Hatayı engellemek için libtest.so.3 sembolik ismi oluştururlur.

```
$ ln -s libtest.so.3.0.1 libtest.so.3

$ ls -l libtest*
lrwxrwxrwx 1 nazim nazim   16 Ağu 29 11:36 libtest.so.3 -> libtest.so.3.0.1
-rwxrwxr-x 1 nazim nazim 8048 Ağu 29 11:33 libtest.so.3.0.1
```

Tekrar derle

```
$ gcc -O2 -Wall -o prog1 prog1.c libtest.so.3.0.1

$ readelf -d prog1 | grep NEEDED
  0x0000000000000001 (NEEDED)             Shared library: [libtest.so.3]
  0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
```



EK 20

Kütüphane ismi olarak libtest.so.3.0.1 yazılmasına rağmen çalışma zamanında libtest.so.3 aranacaktır. Sembolik link ile hata olması engellenir.

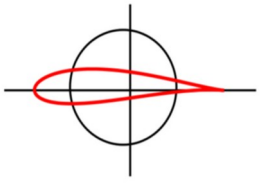
Çalışma zamanında kütüphanelerin sorunsuz kullanılabilmesi için ldconfig ile cache içine eklenebilir (gömülü sistemlerde tavsiye edilmez, rootfs ise app her zaman ayrı olmalıdır).

```
$ cp -a libtest.so.3* /usr/lib/

$ sudo ldconfig -v | more

opt/sysprog/lib:
        libtest.so.3 -> libtest.so.3.0.1

$ ldconfig -p | grep libtest
   libtest.so.3 (libc6,x86-64) => /opt/sysprog/lib/libtest.so.3
```

Sadece kütüphane adının verilmesi

Sembolik link genelde major numaralı kütüphane ismini gösterir.

```
$ cd /opt/sysprog/lib

$ ln -s libtest.so.3 libtest.so

$ ls -l
total 8
lrwxrwxrwx 1 nazim nazim   12 Ağu 29 11:56 libtest.so -> libtest.so.3
lrwxrwxrwx 1 nazim nazim   16 Ağu 29 11:36 libtest.so.3 -> libtest.so.3.0.1
-rwxrwxr-x 1 nazim nazim 8048 Ağu 29 11:33 libtest.so.3.0.1
```

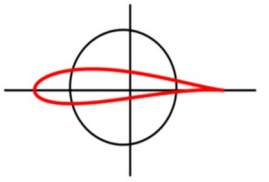


EK 20

Bu durumda derleme sorasında **-ltest** seçeneği kullanılabilir.
O yüzden bu isme "linker name" denir.

```
$ gcc -O2 -Wall -o prog1 prog1.c -L/opt/gomsis/test/ek22 -ltest

$ ldd prog1
linux-vdso.so.1 =>  (0x00007fff725fe000)
libtest.so.3 => /opt/sysprog/lib/libtest.so.3 (0x00007f5b2e479000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f5b2e0af000)
/lib64/ld-linux-x86-64.so.2 (0x000055ec58757000)
```

Dikkat edilirse, her durumda majör uyumlu kütüphane kullanılır.



EK 21, Plugins

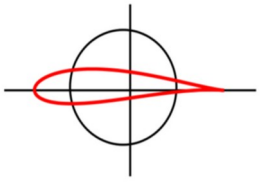
Paylaşılan kütüphaneler yürütme zamanında da yüklenebilir.
Bu kütüphaneler **dlopen(), dlclose() ve dlsym()** komutları ile yönetiliriler.
Bir tür plugin uygulamasıdır.

```
$ make
$ ls -l libtest.so
$ gcc -O2 -Wall -o prog1 prog1.c -ldl
```

prog1 içinde libtest.so yoktur.

```
$ ldd prog1
   linux-vdso.so.1 =>  (0x00007fffa4f2d000)
   libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f9a4d44e000)
   libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f9a4d084000)
   /lib64/ld-linux-x86-64.so.2 (0x00007f9a4d652000)
```

```
$ ./prog1
evrene dair sorunun cevabı 42
```

EK 21

Hatalı durum oluşturmak için kütüphane adını değiş.

```
$ mv libtest.so _libtest.so

$ ./prog1
./libtest.so: cannot open shared object file: No such file or directory
```

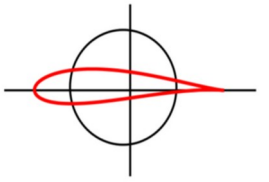
RTLD_NOW
   dlopen() dönmeden önce shared lib içindeki bütün değişkenler çözülür, kullanıma hazır hale gelir.

RTLD_LAZY
   Çağrıldığında çözümleme yap.

dlopen(NULL, _) komutunda ilk argüman NULL ise, dlsym() ile arama işlemi, main içinde ve o ana kadar yüklenmiş bütün shared lib'ler içinde yapılır.

Aynı shared object 2. kez yüklenirse bir hata olmaz ve aynı handle döner.

Kütüphane araması, ld.so'daki mantığa göre yapılır.

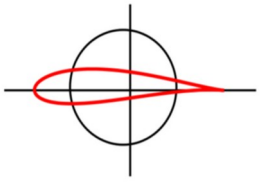


EK 22, En basit kernel modülü

En basit modül, "merhaba dünya" örneği

header'lar yüklü olmalıdır.

```
$ apt-get install build-essential linux-headers-$(uname -r)

$ make
$ file hello.ko

$ insmod hello.ko
$ insmod hello.ko  # hata verir.
$ modinfo hello.ko

$ lsmod
$ modinfo herhangi.ko

$ lsmod | head
$ dmesg|tail

$ tail /var/log/syslog

$ rmmod hello     # veya
$ rmmod hello.ko

$ lsmod | grep hello
```



EK 22

Parametre aktarımı

```
$ insmod hello.ko my_city="ankara"
$ tail /var/log/syslog
$ tail /var/log/kern.log
```

Hatalı parametre aktarımı

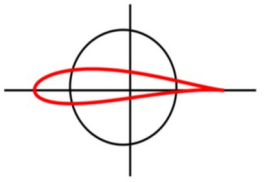
```
$ rmmod hello.ko
$ insmod hello.ko city="ankara"
$ tail /var/log/syslog   # hata mesajını gör.
```

Bağımlılık çok fazlaysa modprobe kullanılır.

İsimler aşağıdaki gibi olursa, module_init() ve module_exit() tanımlarına gerek yoktur.

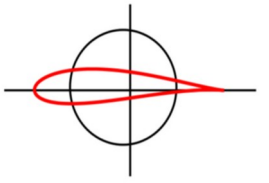
```
int  init_module(void);
void cleanup_module(void);
```

```
$ make clean  # ortalığı temizle
```



EK 22, File Operations

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, loff_t, loff_t, int datasync);
    int (*aio_fsync) (struct kiocb *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
    unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned
long, unsigned long);
    int (*check_flags)(int);
    int (*flock) (struct file *, int, struct file_lock *);
    ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t,
unsigned int);
    ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned
int);
    int (*setlease)(struct file *, long, struct file_lock **);
    long (*fallocate)(struct file *file, int mode, loff_t offset,
              loff_t len);
};
```

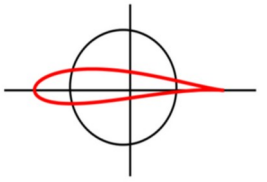
EK 23, Dosya Sistemleri

Dosya ve dizinlerin, hiyerarşik bir yapıda tutulduğu veri yapılarıdır.
Gömülü sistemlerde genelde aşağıdaki dosya sistemleri kullanılır.

- **vfat**
- **ext2**
- **ext3**
- **ext4**
- **tmpfs**
- **ramdisk**
- **initramfs**
- **cpio**
- **NFS**
- **cramfs**
- **romfs**
- **squashfs**
- **zram**
- **eCryptfs**
- **ubifs**
- **pseudo/virtual fs**

Dosya sistemleri ve parametreleri amaca uygun seçilmelidir.
Kötü seçim kaynak israfına ve fiziksel yapının bozulmasına sebeb olabilir.

Linux pek çok dosya sistemine destek verir.
Fakat çok az dosya sistemi Gömülü Sistemler için uygundur.
Dosya sistemi öncelikle cihaza uygun seçilmelidir.
Prensip olarak swap kullanılmamalıdır.

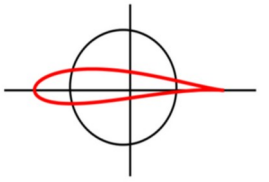


Dosya sistemlerinin kurulduğu bazı ortamlar,

SD/MMC/eMMC	vfat, ext2, ext3, ext4, ...
NAND/NOR Flash	ubifs, jffs2, yaffs2, ...
RAM:	ramdisk, tmpfs, ramfs, zramfs, ...
Network	NFS, samba, ...
Pseudo/virtual	devfs, procfs, sysfs, pipefs, debugfs, ...

Seçilen her dosya sistemi için çekirdeğe uygun destek verilmelidir.
Dosya sistemleri genelde çekirdekte çok yer kaplarlar.
Kullanılmayan dosya sistemleri çekirdeğe eklenmemelidir.
Kök dosya sistemi hariç, bütün dosya sistemleri modül olarak derlenebilir.

cat /proc/filesystems komutu ile çekirdeğin o an için desteklediği dosya sistemleri incelenebilir.



VFAT

MS-Windows dosya sistemidir.

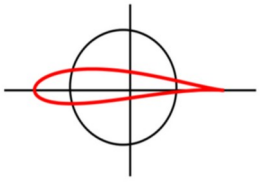
255 karaktere kadar dosya isimlerine izin verir.

Linux ihtiyaçlarını karşılayamaz.
Sahiplik, mod, sembolik link vs. mevcut değildir.
Asla kök dosya sistemi vfat olmamalıdır.

İç yapısı aşırı basit olduğu için daha çok u-boot gibi açılış-yükleyicileri tarafından kullanılır.

Genelde SD/MMC kartların 1. bölümüne kurulur.

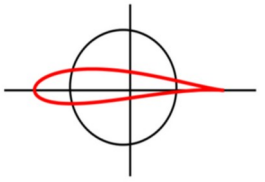
```
$ cd /tmp

$ dd if=/dev/zero of=disk1.img bs=1M count=32
32+0 records in
32+0 records out
33554432 bytes (34 MB) copied, 0,0855615 s, 392 MB/s

$ mkfs.vfat disk1.img
mkfs.vfat 3.0.14 (23 Jan 2023)

$ mkdir /mnt/disk1

$ mount disk1.img /mnt/disk1
```

```
$ df /mnt/disk1
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/loop0         32686     0     32686   0% /mnt/disk1

$ ls -l /mnt/disk1
total 0

$ mkfs.vfat /dev/mmcblk0p1
mkfs.vfat 3.0.14 (23 Jan 2023)

$ mkdir /mnt/disk1

$ mount disk1.img /mnt/disk1

$ df /mnt/disk1
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/loop0         32686     0     32686   0% /mnt/disk1

$ ls -l /mnt/disk1
total 0
```

Çekirdek desteği aşağıdaki gibi verilir.

```
   File systems -->
         DOS/FAT/NT Filesystems -->
               <*> VFAT (Windows-95) fs support
```



ext2

Linux'un en eski dosya sistemlerinden biridir.
Çok uzun zamandan beri ext2 ile ilgili hiç bir “bug report” yapılmamıştır.

Kök dosya sistemi için gerekli bütün ihtiyaçları karşılar.
Masaüstü sistemlerde artık kullanılmamaktadır.
Ani kapanmalara karşı çok dayanıksızdır.

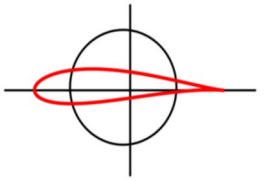
MMC cihazlarında, RO modu ile bağlanarak kullanılmalıdır.
Kök dosya sistemi olarak, RW modunda bağlanmamalıdır.

Kök dosya sistemi RW modunda kullanılacaksa, önce ro bağlanmalı,
sonra fsck yapılmalı, gerekirse repair edilmeli ve sonra rw bağlanmalıdır.

```
$ mkdir /mnt/disk2

$ dd if=/dev/zero of=disk2.img bs=1M count=32
32+0 records in
32+0 records out
33554432 bytes (34 MB) copied, 0,069849 s, 480 MB/s

$ mkfs.ext2  disk2.img
$ mount disk2.img /mnt/disk2
$ df /mnt/disk2
```

Çekirdek desteği aşağıdaki gibi verilir.

```
File systems -->
   <*> Second extended fs support
```

ext3

ext2'nin devamı ve log tabanlıdır.
Ani kapanmalara karşı çok dayanıklıdır.
Dosya bütünlüğü çok zor bozulur.
Recovery durumu çok hızlıdır.

Yerini ext4 sistemine bırakmıştır.
MMC cihazlarında, RW modu ile bağlanarak kullanılabilir.
Log tabanlı bir sistem olduğu için, diske yazılacak bütün
veriler önce bir log tablosunda tutulur, daha sonra diske yazılır.

Crash durumunda sonra e2fsck çalıştırmaya gerek yoktur.
Log sayesinde dosya bütünlüğü her zaman sağlanır.

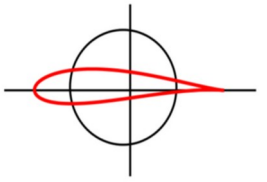
ext3 = journal + ext2

ext2 ve ext3 arasında her zaman geçiş yapılabilir.
Performans ve dosya bütünlüğü arasında ters ilişki vardır.

data=write_back crash sonrası yazılmayan veri kalabilir.
data=ordered Daha güvenlidir ama yavaştır. Tavsiye edilir.

```
$ dd if=/dev/zero of=disk3.img bs=1M count=32
$ mkfs.ext3  disk3.img
$ mkdir /mnt/disk3 && mount disk3.img /mnt/disk3 &&  df /mnt/disk3
```

```
File systems -->
   <*> Ext3 journalling file system support
      [*]   Default to 'data=ordered' in ext3
```

ext4

ext3'ün devamıdır ama ext3 ile uyumlu değildir.

ext4 dosya sistemi, ext3'ü mount edebilir ve ext3'e göre daha iyi performans gösterir.

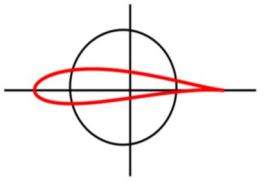
Doğrudan ext4 kullanılması tavsiye edilir.

ext3, ext2 ile uyumlu idi.
Her iki yönde de geçiş yapılabiliyordu.

Örnek dosya sistemi kuruluşu ve çekirdek desteği ext2/ext3 gibidir.

Gömülü sistemlerde, MMC tarafında kök dosya sistemi olarak rw modunda mount edilebilir.

ext2 sisteminin rw modunda bağlanması tavsiye edilmez.



tmpfs

Sanal bellektei (virtual memory) kurulan bir dosya sistemidir.

virtual memory = ram + swap

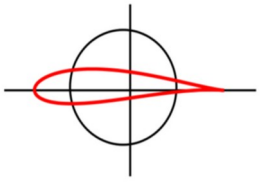
Doldukça büyür, içi boşaldıkça küçülürler.
Ayrılan kapasiteyi RAM'dan çalmazlar.
Swap edilebilirler.

Disk cache sisteminin değiştirilmiş bir hali gibidir.
Çok az yer kaplar.
Bundan dolayı seçilmese bile çekirdek tarafında desteği vardır.
/dev/shm, /dev gibi pek çok cihaz, bu dosya sistemini kullanır.

Kapasite verilmezse, RAM'in yarısı kabul edilir.
Çalışma anında kapasite artırılabilir.

umount edildiği an veriler kaybolur.
Güç kesildiği an veriler kaybolur.

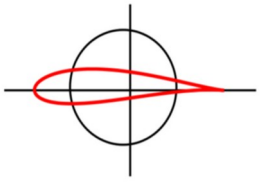
```
$ mkdir /mnt/disk4

$ mount -t tmpfs tmpfs /mnt/disk4
```

```
$ df /mnt/disk4
Filesystem     1K-blocks  Used Available Use% Mounted on
tmpfs            2011344     0   2011344   0% /mnt/disk4

$ free
             total       used       free     shared    buffers cached
Mem:       4022692    2092160    1930532          0     599184 892832
-/+ buffers/cache:     600144    3422548
Swap:      2097148          0    2097148
```

mount edilen bütün dosyalar, işleri bitince umount edilmelidir.
32MB için, ext3 dosya sisteminin maliyeti %15'tir.

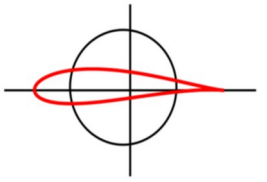
```
$ df /mnt/disk?

Filesystem     1K-blocks  Used Available Use% Mounted on

/dev/loop0         32686     0     32686   0% /mnt/disk1
/dev/loop1         31729   395     29696   2% /mnt/disk2
/dev/loop2         31729  4508     25583  15% /mnt/disk3
tmpfs            2011344     0   2011344   0% /mnt/disk4
```



```
$ mount | grep disk
/tmp/disk1.img on /mnt/disk1 type vfat (rw)
/tmp/disk2.img on /mnt/disk2 type ext2 (rw)
/tmp/disk3.img on /mnt/disk3 type ext3 (rw)
tmpfs on /mnt/disk4 type tmpfs (rw)

$ umount /mnt/disk1
$ umount /mnt/disk2
$ umount /mnt/disk3
$ umount /mnt/disk4
```

Çekirdek desteği aşağıdaki gibi verilir.

```
File systems -->
    Pseudo filesystems -->
        [*] Virtual memory file system support (former shm fs)
```

ramdisk

RAM'ın (virtual memory değil!) disk bölümü gibi kullanılmasıdır.

Dosya sistemi değildir, dosya sistemleri için altyapı hazırlar.

Sanki bir disk bölümü (disc partition) gibi üzerine hemen dosya sistemi kurulabilir.

tmpfs gibi umount edildiğinde yok olmaz.

Cihaz isimleri **/dev/ram0, /dev/ram1, ...** şeklindedir.

Ramdisk adedi, kapasitesi ve blok boyu çekirdek parametresi olarak verilebilir.

Varsayılan blok boyu 1024'tür.

Mümkünse ramdisk kullanılmamalı, bunun yerine tmpfs kullanılmalıdır.

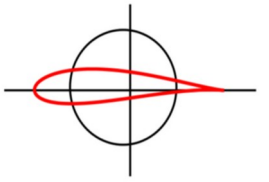
Ramdisk artık çok eski bir teknik olarak kalmıştır.

ramdisk, kullanıldığı an RAM bellekten kapasite çalar.

umount edilse bile kapasite geri verilmez.

Sanal dosya sistemi kullanmadığı için ihtiyaç durumunda swap edilemez.

Prensip olarak gömülü sistemlerde swap kullanılmaz.



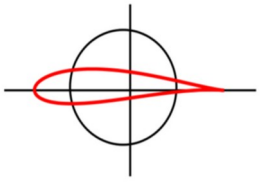
initrd

Bazı çekirdekler hala initial ram disk\footnote{initrd} kullanmaktadırlar.
Fakat initrd kavramı yerini initramfs'e (tmpfs'e) bırakmıştır.
tmpfs bir dosya sistemidir.
Çekirdek tarafından bizim hiç ilgilenmediğimiz bir yerlerde kurulur!

Ramfs veya **/dev/ram0** veya ramdisk fiziksel bir ortamdır.
Bu ortamı kullanabilmek için illa ki üzerine bir dosya sistemi kurulmalıdır,
ext2, ext3, minix, vfat, vs gibi.

/dev/ram0 üzerine dosya sistemi kurulur ve mount edilir.
tmpfs doğrudan mount edilir.

Çekirdek desteği varsa mevcut RAMDISK bilgileri aşağıdaki gibi elde edilebilir.
Modül olarak derlenmişse, brd veya rd ismi ile yüklenebilir.



```
$ dmesg | grep RAMDISK
```

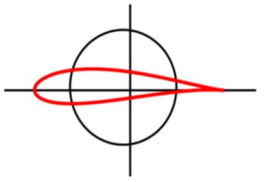
4M'lık ext2 kur ve paketle?

```
$ dd if=/dev/zero of=/dev/ram0 bs=1k count=4096
$ mkfs.ext2 /dev/ram0 4096
$ mkdir /mnt/disk
$ mount /dev/ram0 /mnt/disk
$ df
$ mount
$ umount /mnt/disk

$ dd if=/dev/ram0 of=/tmp/ram0.img bs=1k count=4096
$ dd if=/dev/ram0 bs=1k count=4096 | gzip > /tmp/ram0.img.gz
```

Doğrudan imaj olarak mount etmek, /dev/ram0 kullanmadan.

```
$ mount -o loop /tmp/ram0.img /mnt/disk
```

```
Device Drivers -->
   Block devices -->
         <*>   RAM block device support
               (16)    Default number of RAM disks
               (65536) Default RAM disk size (kbytes)
```



initramfs

Çekirdek tmpfs destekli bir dosya sistemini kök dosya sistemi olarak bağlayabilir.
Bu dosya sistemine initramfs denir, altyapısı tmpfs dosya sistemidir.

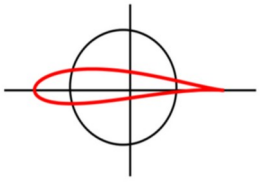
initramfs, doğrudan çekirdeğe de gömülü olabilir.

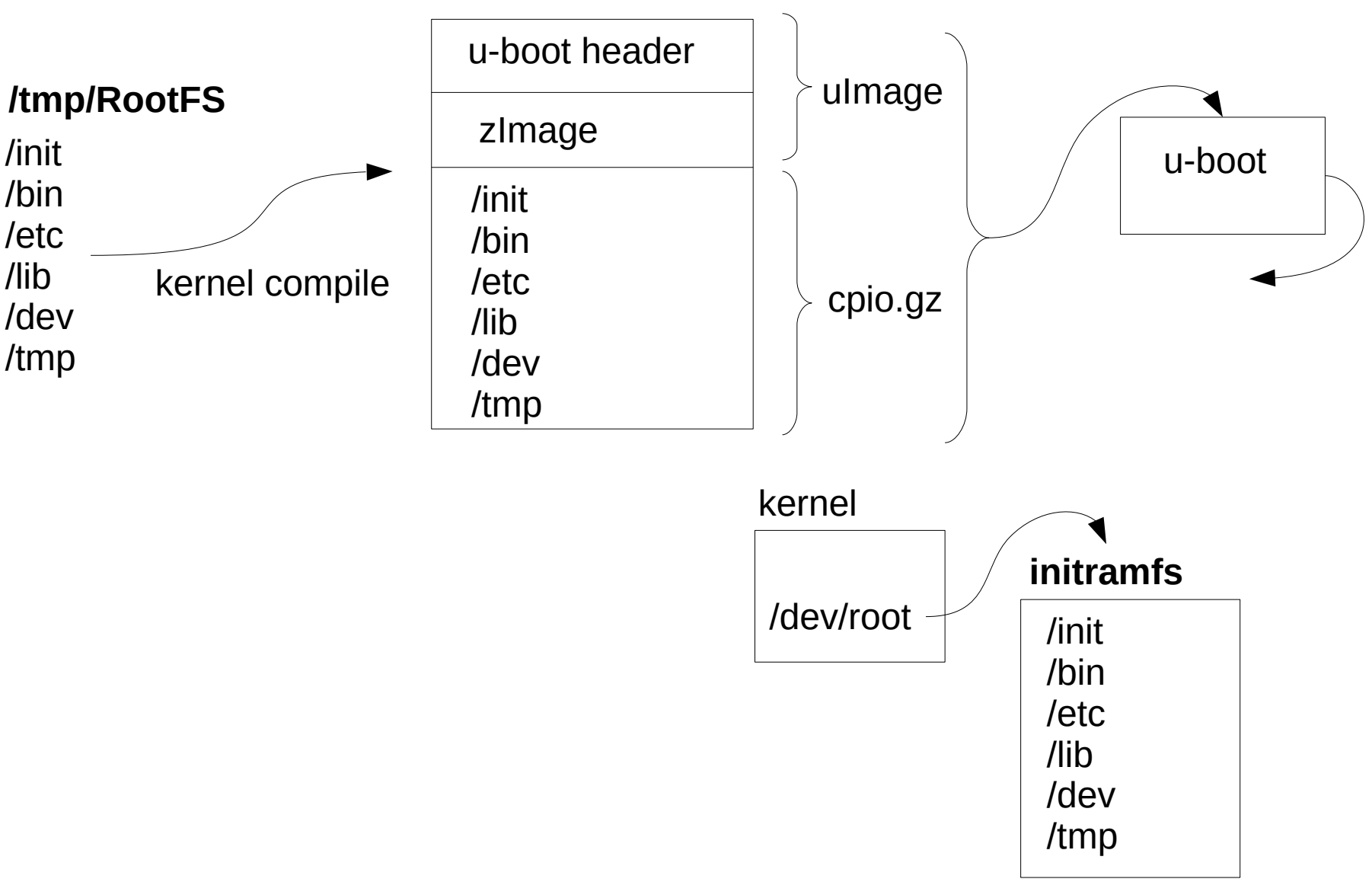
Temeli ramdisk olan dosya sistemine ise initrd denir ve artık kullanılmamaktadır.
initramfs sistemi çekirdek yüklendikten hemen sonra mount edilir.
Genelde modül yüklemek içindir veya esas kök dosya sistemi bağlanmadan önce ön hazırlıkları yapmak için kullanılır.

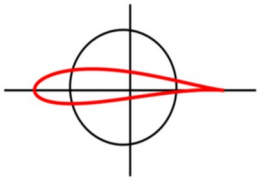
Gömülü sistemlerde esas kök dosya sistemi olarak da kullanılabilir.
Bu dosya sistemi çekirdeğin içine gömülü olabilir.
Ayrı bir dosya olarak da bulunabilir.

Ayrık olan dosya boot loader tarafından çekirdekle birlikte yüklenir ve başlangıç adresi ve dosya boyu çekirdek parametresi olarak çekirdeğe gönderilir.

Ayrık dosya her zaman cpio arşivi şeklindedir, dosya sistemi değildir.




/tmp/RootFS
/init
/bin
/etc
/lib
/dev
/tmp
kernel compile
u-boot header
zImage
/init
/bin
/etc
/lib
/dev
/tmp
uImage
cpio.gz
u-boot
kernel
/dev/root
initramfs
/init
/bin
/etc
/lib
/dev
/tmp



Çekirdek önce kendini açar, sonra da cpio arşivini, bir tmpfs dosya sistemi içine açar ve bu dosya sistemini, kök dosya sistemi olarak bağlar

Hemen peşinden **/init** komutunu gözü kapalı işletir.

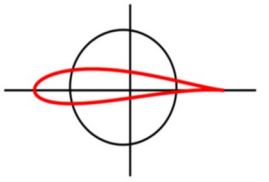
initrd tekniği ile açılış yapıldığında **/linuxrc** işletilir.

initrd sistemi artık eski kalmıştır ve kullanılmamaktadır.

Örnek olarak, **/opt/gomsis/RootFS** altında kurulan kök dosya sistemi aşağıdaki gibi çekirdek koduna eklenebilir.

```
[*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
   (/opt/gomsis/RootFS) Initramfs source file(s)
   (1000)     User ID to map to 0 (user root)
   (1000)     Group ID to map to 0 (group root)
   [*]   Support initial ramdisks compressed using gzip
```

Çekirdek derlemesi sırasında, **/opt/gomsis/RootFS** altındaki bütün dosya ve dizinler otomatik olarak cpio arşivi haline getirilir, gzip ile sıkıştırılır ve çekirdek kodunda eklenir.



Eğer RootFS çok büyükse bu teknik uygun değildir.
RootFS içindeki her program GPL veya benzeri olmak zorundadır.

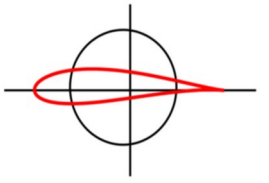
Açılış çok hızlıdır, çekirdek yüklendiği an kök dosya sistemi de yüklenmiş gibidir.
Aradaki süre çok azdır.

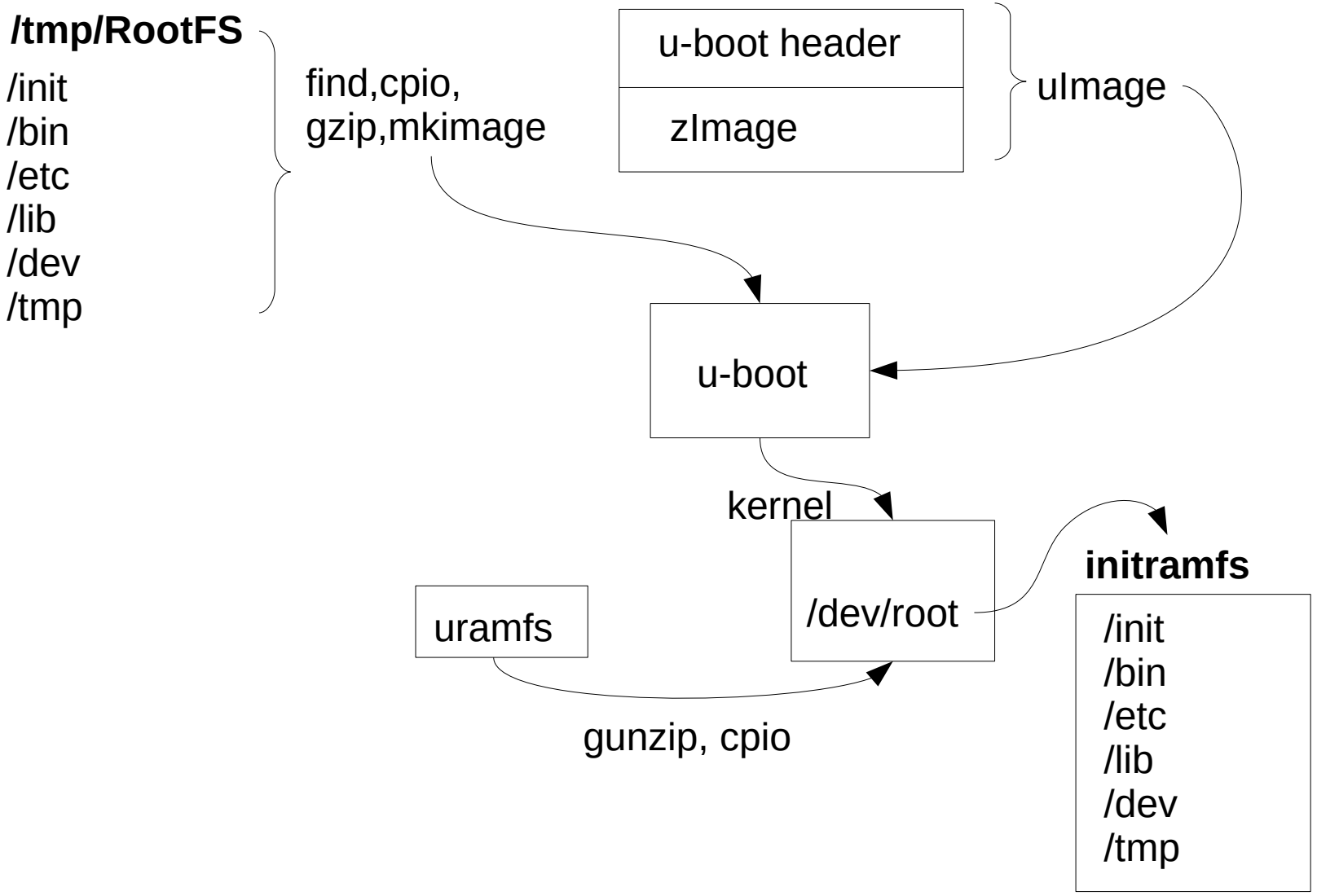
Masaüstü sistemlerde gerçek kök dosya sistemlerine geçiş için kullanılır.
Gömülü sistemlerde acil durum açılışları veya gerçek kök dosya sistemi olarak kullanılabilir.

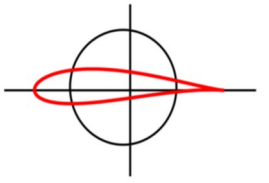
RootFS'deki her güncellemede çekirdek yeniden derlenmeli ve sisteme yüklenmelidir.

Tek bir imaj dosyası ile hem çekirdek hem kök dosya sistemi taşınır ve boot loader sadece tek dosya yükler.

...




/tmp/RootFS
/init
/bin
/etc
/lib
/dev
/tmp
find,cpio,
gzip,mkimage
u-boot header
zImage
uImage
u-boot
kernel
/dev/root
initramfs
/init
/bin
/etc
/lib
/dev
/tmp
uramfs
gunzip, cpio



Initramfs, çekirdekten ayrı da kurulabilir.
Çekirdek bilinen yolla derlenir.

```
[*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
   () Initramfs source file(s)
```

"() Initramfs source file(s)" satırında girilecek olan dizin ismi boş bırakılır.
/opt/gomsis/RootFS için cpio arşivi hazırlanır.

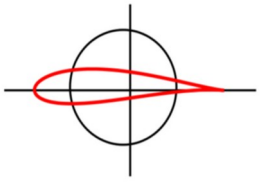
```
$ find . | cpio -v -o -H newc | gzip > ramfs.gz
$ mkimage -A arm -T ramdisk -C none
          -n "cpio test imaji" -d ramfs.gz uramfs
```

uImage ve uramfs beraber kullanılır.
uramfs içindeki program lisanslarının GPL uyumlu olması gerekmez.
Güncellemelerde sadece uramfs kullanılır, çekirdek derlenmez.

u-boot, önce çekirdeği sonra uramfs'i yükler ve
çekirdeğe uramfs'in yüklendiği adresi ve boyunu parametre olarak geçirir.

Çekirdek, daha önce bahsedildiği gibi arşivi gunzip ile açar,
tmpfs içine cpio arşivini kopyalar ve **/init** programını gözü kapalı başlatır.

Çekirdek ve uramfs ayrı ayrı yüklendiğinden, çekirdeğe gömülü sisteme göre biraz daha yavaş açılır.



cpio

cpio bir dosya sistemi değildir.
Bir arşiv programıdır.
En kaba tabiri ile bütün dosyaları alt alta ekler.
tar programı ile aynı sınıftandır.

initramfs sistemleri için de arşivleme yapar.
cpio kodu çekirdek kodu içinde gömülü durumdadır.
Aynı anda ya input ya da output modunda çalışır. output modu paketleme, arşivleme yapar.

output modunda dosya isimleri bir dosyadan satır satır okunur.

```
$ cpio -o -H newc < list.txt
$ ls falan.* | cpio -o -Hnewc > test.cpio
```

Çekirdek sadece newc arşiv formatını kullanır.

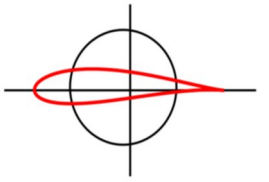
cpio içinde gerekli dosya isimleri genelde find komutu ile elde edilir.

```
$ find . | cpio -o -H newc | gzip > rootfs.img.gz
```

input veya extract modunda paket açılır.

```
$ gunzip rootfs.img.gz
$ cpio -i -d -H newc -F rootfs.img --no-absolute-filenames
```

-i input veya extract demektir.
-o output veya create demektir.



nfs

Network File System, ağ üzerinden dosyalara erişim tekniğidir.
Diğer bir deyişle, ağ üzerinden dizin paylaşım yöntemidir.

Kök dosya sistemi ağ üzerinden bağlanabilir ya da
host üzerindeki herhangi bir dizin export edilebilir.

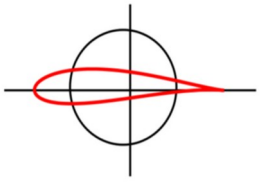
/etc/exports
/tftpboot 10.0.0.0/24(rw,insecure, no_subtree_check,async,no_root_squash)

$ exportfs -avr ile yapılan güncellemeler sunucuya bildirilir.

no_root_squash sayesinde root yetkisi ile işlem yapılabilir.
insecure ile güvenli portların dışında da port numarası kullanılabilir.
no_subtree_check ile dışarı taşan dizinlere erişim sağlanır.

Ağ için çekirdek parametreleri en genel hali ile aşağıda verilmiştir.

ip=<client-ip>:<server-ip>:<gw-ip>:<netmask>:<hostname>:
<device>:<autoconf>:<dns0-ip>:<dns1-ip>



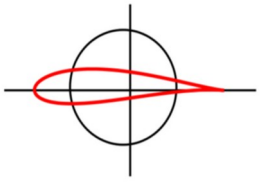
```
autoconf parameters: Documentation/filesystems/nfsroot.txt

off or none: don't use autoconfiguration (do static IP assignment instead)
on or any:   use any protocol available in the kernel (default)
dhcp:        use DHCP
bootp:       use BOOTP
rarp:        use RARP
both:        use both BOOTP and RARP but not DHCP
                    (old option kept for backwards compatibility)
Default: any
```

Örnek çekirdek parametreleri aşağıda verilmiştir.

```
ip=10.0.0.111:10.0.0.4:10.0.0.4:255.255.255.0:test1:eth0:off
   root=/dev/nfs
   rw
   nfsroot=10.0.0.4:/tftpboot/RootFS
```

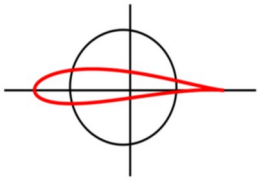
Sıradan bir dizini mount etmek için,

```
$ mount -t nfs 10.0.0.4:/tftpboot/app /mnt/nfs
 ...
$ umount /mnt/nfs
```

Kök dosya sistemi NFS üzerinden kullanılacaksa çekirdek aşağıdaki gibi derlenir.

```
[*] Networking support
    Networking options -->
      [*] TCP/IP networking
         [*]   IP: kernel level autoconfiguration

File systems -->
     [*] Network File Systems
     <*>   NFS client support
     [*]      Root file system on NFS
```

Pseudo File Systems

Sözde dosya sistemleri, kernel ve user space arasında iki yönlü bilgi alışverişi için tasarlanmıştır.
Dosya ve dizinler her zaman ya boot anında ya da tam kullanım anında kurulurlar ve sistem kapanınca yok olurlar. Fiziksel olarak bir diskte kurulmazlar bundan dolayı dosya boyları her zaman 0'dır ama içleri dolu olabilir.
Gömülü sistemlerde genelde açılış betikleri tarafından mount edilirler.
Çok kullanılan bazı dosya sistemlerinin mount edilmesi...

```
$ mount -t proc      proc   /proc
$ mount -t sysfs     sysfs  /sys
$ mount -t devtmpfs  none   /dev
$ mount -t devpts    devpts /dev/pts
$ mount -t tmpfs     tmpfs  /dev/shm
                     ^
                     bu kolonun tamamı none olabilir.
```

-- BİTTİ --