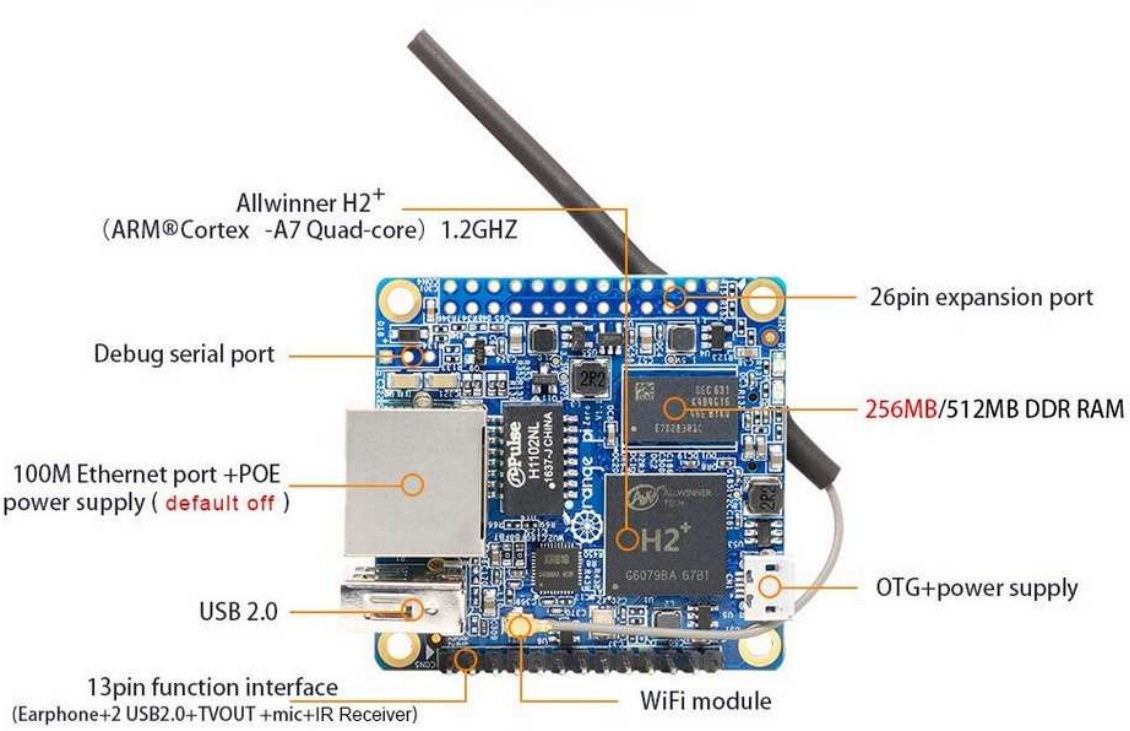
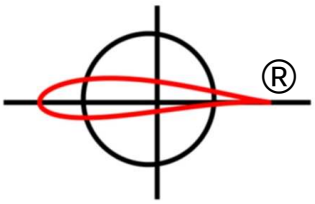
# Gömülü Linux Sistemleri
# OrangePi Zero Uygulaması

http://UCanLinux.Com

18 Mayıs 2017



Nâzım KOÇ

# İçindekiler

# Bölüm 1

# Giriş

Gömülü Linux sistemlerini öğrenmenin en iyi yollarından biri baştan sona bir GNU/Linux sistemini bir cihaza kurmaktır. Herhangi bir cihaz satın alındığında, cihazda çalışacak hazır Linux dağıtımları da beraber gelir. Bu dağıtımlar genelde imaj şeklindedir ve bir iki komutla hemen kurulabilir. Böylece cihaz üzerinde çalışan bir Linux sistemi kolaylıkla elde edilebilir. Fakat bu tür bir çalışmanın, gömülü sistem bilgisine yaptığı katma değer sıfıra yakındır.

Yüksek katma değerli bir çalışma için elimizde uygun bir bordun bulunması ve power-on ile login arasındaki bütün adımların el yordamı ile yapılması gerekli ve yeterlidir.

Piyasada çok uygun fiyatlı pek çok cihaz mevcuttur. Bu belge Orange Pi Zero cihazına sıfırdan bir Linux sisteminin nasıl kurulacağından bahsetmektedir. Bahsi geçen konuların neredeyse hiç birisi cihaza bağımlı değildir. Çok ufak değişikliklerle benzer çalışmalar farklı cihazlar üzerinde yapılabilir.

Bu cihazın seçilmesinin ana sebepleri, piyasaya yeni çıkması ve denemek istememiz, Türkiye'deki fiyatının 60 TL[1] civarında olması ve u-boot ve Linux ile barışık olmasıdır.

Burada yapılacak çalışmada cihaz üzerine açılabilir bir Linux sistemi kurulacaktır. Fakat bu sistem Ubuntu veya Raspian veya diğer dev dağıtımlar gibi dört başı mağrur ve de takla atabilen bir sistem olmayacaktır. Öyleki örnek kurulacak sistemde sadece eternet kartı çalışacaktır. Bunu dışında cihazın

[1] 256MB'lık aynı cihazın Aliexpress'deki tek satış fiyatı 7$'dır. Trump dün Suriye'yi bombaladı, bizimkiler de alkış tuttu, 1$=3.73TL'dir.

çalışan başka özellikleri var mı, bilmiyoruz. Niçin?

Burada amaç her işi yapan bir sistem kurmak değil, bir Linux sisteminin özel amaçlar için nasıl kurulacağını göstermektir. Gömülü Linux sisteminin mantığı anlaşıldığında, Yocto gibi şu anda çok moda olan sistemlerin analizi de okuyucu için daha kolay olacaktır.

Gömülü sistemler[2] genelde tek bir işi çok iyi yapması için kurulan sistemlerdir. Bundan dolayı her işi yapmak için kurulan Ubuntu veya Debian gibi işletim sistemlerini gömülü sistem projelerinde kullanmak pek de verimli olmayacaktır.

Bu çalışmada son derece basit, içinde sadece bir adet binary dosya bulunan 2 farklı gömülü sistem kurulacaktır. Açılışta istenilen gömülü sistem seçilebilecektir.

Onlarca gömülü sistem tekniği vardır. Burada çok basit olduğu için, kök dosya sistemi SD kartta ve kök dosya sistemi kernel içinde olan iki farklı gömülü sistem üzerinde çalışılmıştır. Bu sistemlerin her ikisi de SD kart üzerinden açılmaktadır. Başka örnek gömülü sistem örnekleri için `http://ucanlinux.com/wp-content/uploads/bbb_ucanlinux.pdf` adresindeki belge incelenebilir.

bbb_ucanlinux.pdf belgesi daha geniş yazılmıştır ve BeagleBone Black için örneklendirilmiştir. Bu belge dikkatlice incelenirse BeagleBone ve Orange Pi arasında neredeyse hiç bir farklılığın olmadığı görülebilir.

Okuyucunun bu belgedeki çalışmaları yürütebilmesi için Linux yüklü bir makine ile çalışması ve terminalden çalışmaya aşina olması gerekir. Bu belgede yazılan komutlar gözü kapalı şekilde terminalden girilmemeli, her komutun ne iş yaptığı ayrıca araştırılmalıdır. Ayrıca verilen bazı adresler zaman içinde kaldırılmış olabilir. İnternete düşen hiç bir belge kaybolmaz. Ufak bir araştırma ile uygun dosya veya siteler bulunmalıdır.

Gömülü sistemlerle uğraşan kişinin vaktinin bol, sabrının çok olması gerekir. Kişide veya projesinde bu özellikler mevcut değilse, hazır bir imajı cihazına kurmalı, mutlu ve mesut bir biçimde hayatına devam etmelidir.

[2]İşletim sistemi kurulmadan çalışabilen çok geniş bir gömülü sistem ailesi mevcuttur. Bizler sadece Linux altında kurulan gömülü sistemlerle ilgilenmekteyiz.

# Telif Hakları

Bu belge[3] OrangePi Zero isimli tek-kart bilgisayara Linux işletim sisteminin nasıl kurulacağından bahseder.



Bu belgenin en son sürümü `http://UCanLinux.Com` adresinden indirilebilir.

Bu belge ile ilgili her türlü bilgi, eleştiri ve yorum için `http://UCanLinux.Com` sitesindeki iletişim bilgileri kullanılabilir.

[3] `file:/nk/workspace/projects/linux.egitimi/orangepi.zero/latex`, 18 Mayıs 2017

# Bölüm 2

# Çalışma ortamı

Sıfırdan gömülü sistem geliştirmek için, çapraz derleyiciye, açılış yükleyicisi, çekirdek ve busybox'ın kaynak kodlarına ve örnek bir kök dosya sistemine gerek olacaktır. Bütün bu dizin veya dosyaları bir arada, topluca tutmak son derece yararlı olacaktır. Bizler aşağıdaki dizin yapısında çalışmaktayız.

```
$ tree --charset unicode -L 1 /opt/gomsis

/opt/gomsis
|-- bin
|-- busybox
|-- configs
|-- ftp
|-- linux
|-- RootFS.skel
|-- toolchain
`-- u-boot
```

Bu belge boyunca $ işareti terminalden yapılan girişleri temsil etmektedir. Ayrıca bazı komutlar "permission denied" hatası verebilir. Bu durumda, çalışılan Linux dağıtımına göre root kullanıcısına geçilmesi veya sudo komutunun kullanılması gerekir. Bu durumlar ayrıca belirtilmemiştir.

Verilen örnek dizin yapısı bütün projenin tek bir dizin altından kolaylıkla yönetilmesini sağlamaktadır. Bu yapının içi, ilerleyen bölümlerde tamamen doldurulacaktır.

Başlangıç dizini `/opt/gomsis` seçilmiştir. Yıllardır hep aynı dizini kullanıldığımız için bu dizin başlangıç olarak seçilmiştir. Okuyucu istediği başlangıç dizinini

seçebilir. Fakat alt dizin isimlerinin aynı olması tavsiye edilir.

Her bir dizinin içeriği aşağıda kısaca verilmiştir.

**bin** örnek bir dosya barındırır.

**configs** u-boot, busybox ve çekirdeğin config dosyalarının yedekleri.

**RootFS.skel** İçi hemen hemen boş olan iskelet kök dosya sistemi.

**ftp** wget ile indirilen dosyaların oturduğu dizin.

**toolchain** İçinde çapraz derleyici, debugger, sysroot gibi geliştirme araçlarını barındıran dizin.

**u-boot** Açılış yükleyicisinin git ile indirilen kaynak kodu.

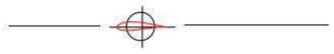
**linux** Çekirdeğin git ile indirilen kaynak kodu.

**busybox** busybox paketinin git ile indirlen kaynak kodu.

Her bir dizin ve içeriği ilerleyen bölümlerde kurulacaktır. Vakit kaybı olmaması için configs ve RootFS.skel dizinleri tar.gz olarak ayrıca verilmiştir. Bu sıkıştırılmış arşiv dosyası, ileride kullanılmak üzere aşağıdaki gibi açılabilir.

```
$ mkdir /opt   # /opt varsa bu satıra gerek yok.
$ cd /opt
$ wget http://ucanlinux.com/wp-content/uploads/orangepi_zero.tar.gz
$ tar zxvf orangepi_zero.tar.gz
$ ls -l gomsis                      # dizini incele.

# root ile çalışma, kendi userId:groupId değerini kullan.
$ chown -R nazim:nazim  gomsis
```

configs dizini içinde gerekli .config dosyaları mevcuttur. RootFS.skel içinde ise iskelet kök dosya sistemi bulunur. Yeri geldiğinde bu dosya ve dizinler kullanılacaktır.



# Bölüm 3

# Çapraz Derleyiciler

Farklı mimariler için kod üreten derleyicilere çapraz derleyici denir. Çapraz derleyiciler ile x86 makine içinde, ARM makine için kod üretebiliriz.

Derleyici, debugger, linker ve sysroot gibi geliştirme araçlarının topluluğuna toolchain denir. Orange Pi için Linaro'nun toolchain'i[1] kullanılacaktır.

Toolchain aşağıdaki gibi kurulabilir.

```
$ cd /opt/gomsis/ftp

# indir.

$ wget https://releases.linaro.org/components/toolchain/binaries/4.9-2016.02/\
     arm-linux-gnueabihf/gcc-linaro-4.9-2016.02-x86_64_arm-linux-gnueabihf.tar.xz

# Aç.

$ cd /opt/gomsis/toolchain
$ tar Jxvf ../ftp/gcc-linaro-4.9-2016.02-x86_64_arm-linux-gnueabihf.tar.xz

# Isim çok uzun, sembolik link ile basitleştir.

$ ln -s gcc-linaro-4.9-2016.02-x86_64_arm-linux-gnueabihf arm

# PATH içine gcc'nin dizinini ekle.
# Başka yerde aynı derleyici mevcutsa sorun çıkabilir.
# Bundan dolayı dizin ismi PATH önüne eklenmelidir, sonuna değil.
```

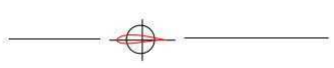
[1] Linaro firması, toolchain'lerin yerlerini devamlı değiştirir. Verilen adreste toolchain yoksa, "gcc linaro gnueabihf" anahtar kelimeleriyle arama yapılabilir.

```
$ vi ~/.bashrc

PATH=/opt/gomsis/toolchain/arm/bin:$PATH
export PATH

# mevcut terminalden çık.

$ exit

# Yeni bir terminal aç.
# Kuruluş düzgün mü test et?
# arm-linux-gnueabihf- yazdıktan sonra iki kez tab'a bas.
# Bütün toolchain komutları listelenmelidir.
# Listelenmezse PATH hatalıdır ya da exit ile çıkılmamıştır.

$ arm-linux-gnueabihf-     <TAB> <TAB>

# 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-get komutları ile kurulması tavsiye edilmez. Yukarıda verilen yöntemle aynı anda aynı toolchain'in farklı sürümleri sorunsuzca kurulup kullanılabilir.

Kuruluşun temel mantığı çok basittir. Toolchain herhangi bir dizine açılır ve arm-linux-gnueabihf-gcc komutunun bulunduğu dizin PATH içine eklenir.

Artık x86 makinede kurulan bu toolchain ile açılış yükleyicisi ve kernel derleyebiliriz.



# Bölüm 4

# Açılış yükleyicisi

Açılış yükleyicisi olarak u-boot[1] [2] kullanılacaktır.

U-boot aşağıdaki gibi indirilip derlenebilir.

```
# indir.

$ cd /opt/gomsis
$ git clone git://git.denx.de/u-boot.git

# Config oluştur.

$ cd u-boot
$ make orangepi_zero_defconfig

# Config güncelle.

# menuconfig ekranı geldiğinde, hiç bir seçeneğe dokunulmamalıdır.
# Sonra ustalaştıkça u-boot parameterleri üzerinde oynama yapılabilir.
# Şimdilik açılış gecikmesi 5 saniye yapılmalı ve prompt değiştirilmelidir.
#
# Bunun için ana menüden başlayarak aşağıdaki güncellemeler yapılmalı
# ve exit ile menuconfig'den çıkılmalıdır.
#
# (5) delay in seconds before automatically booting
#     Command line interface  --->
#     (UCanLinux > ) Shell prompt
#
# Shell prompt kısmına herkes kendi adını yazabilir.
```

[1] Kaynak: http://linux-sunxi.org/Bootable_SD_card#Partitioning
[2] Kaynak: http://linux-sunxi.org/Mainline_U-boot#Boot

```
# Sadece kozmetiktir.
# Bizler UCanLinux yazdık.
#

$ make menuconfig

# Çapraz derle.
#
# nproc komutu bilgisayardaki işlemci sayısını verir.
# -j ile paralel çalışacak iş sayısı verilir.
# Böylece derleme daha çabuk biter.
# -j zorunlu değildir.

$ make CROSS_COMPILE=arm-linux-gnueabihf- -j$(nproc)

# Sonuçları incele.

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

-rw-rw-r-- 1 nazim nazim  24576 Mar 14 09:17 spl/sunxi-spl.bin
-rw-rw-r-- 1 nazim nazim 387703 Mar 14 09:17 u-boot.img
-rw-rw-r-- 1 nazim nazim 420471 Mar 14 09:17 u-boot-sunxi-with-spl.bin
```

Buraya gelindiğinde açılış yükleyicileri artık elde edilmiştir. sunxi-spl.bin programına 1. açılış yükleyicisi[3], u-boot.img programına 2. açılış yükleyicisi[4] denir.

OrangePi açıldığında önce sunxi-spl.bin programını yükler. Bu program da u-boot.img programını yükler. u-boot.img programı da linux çekirdeğini yükler.

Her bordun 1. ve 2. açılış yükleyicilerini yükleme tekniği çok farklıdır. Bazı bord'lar hem 1. hem de 2. yükleyicileri doğrudan dosya sistemlerinden yükler. Bu dosya sistemleri belirli bir tipte olmalı, partition tekniği ve disk geometrisi tam da bordun beklediği gibi olmalıdır. Örneğin, ilk bölüm vfat olmalı, boot flag açık olmalı, disk geometrisi 63 sector/track olmalı vs gibi.

OrangePi bu konuda çok farklı bir yol izler. Açılış yükleyicilerini doğrudan boot sektör'den yükler. Bu da çok büyük bir esneklik sağlar. Boot sektörün haritası Tablo 4.1'de verilmiştir.

[3]First stage boot loader
[4]Second stage boot loader

Tablo 4.1: Açılış Alanının Haritası

| START | SIZE | USAGE |
|---|---|---|
| 0 | 8KB | Unused, available for partition table etc. |
| 8 | 24KB | Initial SPL loader |
| 32 | 512KB | U-Boot |
| 544 | 128KB | Environment |
| 672 | 352KB | Reserved |
| 1024 | - | Free for partitions |

Boot sektörün ilk 512 baytının MBR olduğunu hatırlatalım. Tablo 4.1'deki haritada `Initial SPL loader`[5] olarak gözüken alana 1. açılış yükleyicisi gelecektir. Yani buraya sunxi-spl.bin dosyasını kopyalamamız yeterlidir. `U-Boot` ile gösterilen alana ise `u-boot.img` kopyalanacaktır.

Kopyalama aşağıdaki gibi yapılabilir.

```
# SD kart geliştirme makinesine takılır.
# Örneğin notebook'a takılır.
#
# df ile bakılır.
# Eğer kart otomatik olarak mount edilmiş ise mutlaka unmount yapılmalıdır.
#
# dmesg ile kartın ismine bakılır

$ dmesg | tail

mmc0: new high speed SDHC card at address 1234
mmcblk0: mmc0:1234 SA08G 7.21 GiB
 mmcblk0: p1 p2

# Yukarıdaki örnekte mmcblk0 bizim SD kartımızın ismidir.
# Kullanılan donanıma göre farklı isimler gelebilir.
# Örnegin /dev/sdb veya /dev/sdc gibi cihaz ismleri gelebilir.
#
# Aşağıdaki komut ile ilk 1023 sektör sıfırlanır.
# seek=1 ile MBR ve devamındaki 512 bayta dokunulmaz.
#
# !!! DiKKAT !!! cihaz ismi hatalı ise mevcut makinenizi kaybedebilirsiniz.
# dmesg komutu ile veya başka bir yol ile takılan cihazın
# ismini tam olarak tespit edin.
```

[5]SPL, secondary program loader demektir. Bizler bu alana "first stage boot loader" yüklüyoruz. Donanımcılar ROM boot yükleyicisini first stage kabul ederler.

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

# Kopyalama işlemi aşağıdaki gibi yapılabilir.

$ cd /opt/gomsis/u-boot
$ dd if=u-boot-sunxi-with-spl.bin of=/dev/mmcblk0 bs=1024 seek=8

# sync demeden SD kartı çıkartma.

$ sync
```

Yukarıda verilen haritaya göre, ilk 8K'dan sonra sunxi-spl.bin programı kopyalanmalıdır. Hemen peşinden u-boot.img kopyalanmalıdır.

Bu dosyaları ayrı ayrı kopyalamak yerine doğrudan u-boot-sunxi-with-spl.bin dosyası ilk 8'dan itibaren kopyalanabilir. Çünkü derleme sırasında sunxi-spl.bin ve u-boot.img dosyaları arka arkaya eklenmekte ve u-boot-sunxi-with-spl.bin isimli tek bir dosya üretilmektedir. Böylece tek bir dd komutu ile 1. ve 2. boot yükleyicileri boot sektörde uygun alana kopyalanır.

Bölümlendirme programları ilk 1MB'lık alanı boş bırakmaktadırlar. Diğer bir deyişle dosya sistemleri ilk 1MB'lık alandan sonra başlamaktadır. Böylece açılış yükleyicileri için yeterli yer bırakılmış olur. Tabii bu söylenenler DOS tipindeki bölümlendirmeler için geçerlidir.

Açılış mesajlarını yakalamak ve u-boot ile etkileşim için seri kanal emülatörü kullanılacaktır. Bizler alışık olduğumuz için minicom programını kullanmaktayız. Okuyucu alışık olduğu herhangi bir sistemi kullanabilir.

OrangePi zero için seri kanal tanımları 115200, 8n1 ve "No flow" ile verilir. Bu değelerle minicom[6] aşağıdaki gibi başlatılabilir.

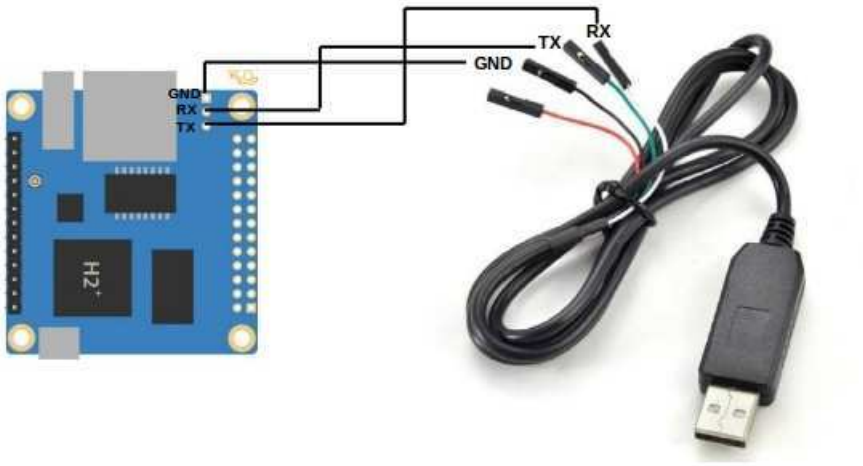
```
$ minicom -c on -w -D /dev/ttyUSB0 -b 115200
```

`-c on` seçeneği minicom'un yüzüne renk katar, "color on" demektir.

`-w` seçeneği ekrana sığmayan satırların bölünmesini engeller ve sığmayan kısmı alt satıra yazar, "line-wrap on" anlamındadır.

`-D /dev/ttyUSB0` seçeneği ile seri kanalın düğüm ismi verilir, "Device" anlamındadır. Bizim sistemimizde seri kanalın ismi `/dev/ttyUSB0`'dır. Oku-

[6] $ man minicom




Şekil 4.1: Seri kablo bağlantısı. Siyah:GND, Yeşil:TX, Beyaz:RX

yucunun sisteminde farklı olabilir. `$ dmesg | tail` komutu ile doğru isim bulunmalıdır.

OrangePi Zero ile pek çok seri kablo kullanılabilir. Bizler `gittigidiyor.com` sitesinden aldığımız 10 TL'lik, dandik marka "PL2303 USB to TTL Çevirici" kullanmaktayız.

OrangePi'de eternet girişinin hemen yanında 3 adet pin bulunmaktadır. En dışarıdaki pin GND, ortadaki RX ve diğeri TX'tir. Kablolama bu şekilde yapılabilir. Seri kablonun "power" ucu asla cihaza takılmamalıdır.

Örnek bir kablo bağlantısı Şekil 4.1'de verilmiştir.

OrangePi Zero makinesi USB kablo ile beslenir. İnce uçlu USB kablosuna gerek vardır. Daha fazla güç sağladığı için, beslemenin USB 3.0 girişinden yapılması tavsiye edilir.

Kopyalama bittikten ve minicom ile seri bağlantı yapıldıktan sonra SD kart borda takılıp güç verilirse, 5 saniyeden geriye saymaya başlayan bir mesaj gelecektir. Bu sırada bir tuşa basılırsa u-boot promptuna düşülür.

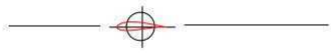
5 saniye beklenirse, u-boot.img programı çekirdeği yüklemeye çalışacak ama bulamadığı için bord tıkanacaktır.

Özetle u-boot'u derledikten sonra u-boot-sunxi-with-spl.bin programını ilk 8K'yı atladıktan sonra SD karta kopyamalak, u-boot açılışı için yeterli olacaktır.

Config yedeği aşağıdaki gibi alınabilir.

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

İlerleyen bölümlerde kernel ve busybox'ın da config dosyaları benzer şekilde yedeklenecektir.



# Bölüm 5

# Bölümlendirme

SD kartı kullanabilmek için bölümlendirme yapmak gerekir. Disk bölümlendirmesi yapan pek çok program mevcuttur. Bizler fdisk programını kullanmaktayız. Okuyucu başka herhangi bir DOS bölümlendirme programını kullanabilir.

SD karta biri 32MB, biri 16MB olan iki bölüm kurulacaktır. Her bölüme iki farklı Linux sistemi kurulacak ve açılışta kullanıcı istediği sistemi seçebilecektir.

İlk bölüme kurulacak Linux sisteminde kök dosya sistemi[1] SD kartın üzerinde olacaktır. Diğer bölümedeki sistemde ise kök dosya sistemi çekirdek içinde gömülü olacaktır. Çekirdeğe gömülen sistem otomatik olarak sıkıştırılır. Bundan dolayı 16MB'lık bir alan fazlası ile yeterlidir.

fdisk programı ile SD kart üzerine 2 bölüm aşağıdaki gibi kurulabilir. Okuyucunun makinesinde mmcblk0 yerine sdb veya sdc gibi isimler gözükebilir. Doğru ismi tespit edebilmek için dmesg komutu kullanılabilir.

```
$ fdisk /dev/mmcblk0   # /dev/sdb veya /dev/sdc veya farklı bir isim olabilir.

Command (m for help): o

Command (m for help): p
Disk /dev/mmcblk0: 7,2 GiB, 7744782336 bytes, 15126528 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

[1] Root File System

```
Disklabel type: dos
Disk identifier: 0x07f8acd3

Command (m for help): n

Select (default p): p

Partition number (1-4, default 1): 1
First sector (2048-15126527, default 2048): 2048
Last sector, +sectors or +size{K,M,G,T,P} (2048-15126527, default 15126527): +32M

Command (m for help): n

Select (default p): p

Partition number (2-4, default 2): 2
First sector (67584-15126527, default 67584):   <ENTER>
Last sector, +sectors or +size{K,M,G,T,P} (67584-15126527, default 15126527): +16M

Command (m for help): p
Disk /dev/mmcblk0: 7,2 GiB, 7744782336 bytes, 15126528 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x07f8acd3

Device         Boot Start    End Sectors Size Id Type
/dev/mmcblk0p1       2048  67583   65536  32M 83 Linux
/dev/mmcblk0p2      67584 100351   32768  16M 83 Linux

Command (m for help): w
```

Az yer kaplasın diye fdisk çıkışlarının bir kısmı kırpılmıştır. Kırpılan yerler sadece bilgi veren satırlardır. fdisk programının kullanımı son derece basittir. "Command" satırında yazılan komutlar kısaca aşağıda verilmiştir.

**o** Bölümlendirme tablosunu[2] sil, yeni DOS tablosu kur.

**p** Bölümlendirme tablosunu göster.

**n** Yeni bir tablo kur.

**w** Tabloyu diske yaz.

[2]Partition table

fdisk programı, MBR'nin[3] sadece 446 ile 512. baytları arası ile ilgilenir. Buraya tablo bilgilerini yazar. Yer çok dar olduğu için sadece 4 adet bölümlendirme yapılabilir. 4'ten fazla bölümlendirme için son bölümün extended olarak işaretlenmesi gerekir. Bu durumda 5. ve diğer bölümler mantıksal bölüm adını alır. Şimdilik bize 2 bölüm yeterlidir.

Dikkat edilirse birinci bölümün başlagıç sektörü 2048'dir. Her bir sektörde 512 bayt olduğu için ilk bölüm 2048*512= 1MB'tan sonra başlar. İşte bu ilk 1MB'lık kısma daha önceki bölümde dd komutu ile 1. ve 2. açılış yükleyicilerini yüklemiştik.

fdisk programı sadece ilk 512 baytın son kısmını günceller. Bundan dolayı yapılan bölümlendirme işlemi bizim daha önce yüklemiş olduğumuz açılış yükleyicilerine dokunmaz.

fdisk'ten "w" ile çıkılmadığı sürece bölümlendirme tablosu MBR'ye yazılmaz. Pişmanlık durumunda Ctrl+C ile çıkılabilir.

"w" denildiği an bütün bilgiler MBR'ye yazılır.

Her ne kadar fdisk programı otomatik olarak sync yapsa da, alışkanlık olması için, SD kart yerinden çekilmeden önce sync komutunun verilmesi tavsiye edilir. Şimdilik kart yerinde kalmalı, çekilmemelidir.

Artık içine veri atılabilecek iki adet disk bölümümüz mevcuttur. Her iki disk bölümüne aşağıdaki gibi ext2 dosya sistemi kurulabilir.

```
$ mkfs.ext2 /dev/mmcblk0p1 -L UCanLinux1
$ mkfs.ext2 /dev/mmcblk0p2 -L UCanLinux2
```

`/dev/mmcblk0` ismi, bütün SD kartı temsil eder. Bundan dolayı fdisk programında bu isim kullanılmıştır.

`/dev/mmcblk0p1` ile `/dev/mmcblk0p2` isimleri ise sırayla 1. ve 2. bölümü temsil eder.

SD kartın cihaz ismi kullanıcıda farklı olabilir. Bu durumda, örneğin mmcblk0 yerine sdb kullanılabilir. /dev/sdb için örnek bir giriş aşağıda verilmiştir.

```
$ mkfs.ext2 /dev/sdb1 -L UCanLinux1
$ mkfs.ext2 /dev/sdb2 -L UCanLinux2
```

[3]Master Boot Record, diskteki ilk 512 baytın adıdır.

`mkfs.ext2`[4] [5] komutu ile her iki bölüme de ext2 dosya sistemi kurulmuştur.

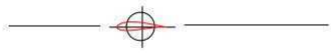
`-L` seçeneği ile dosya sistemlerine etiket[6] verilmiştir. Okuyucu istediği ismi verebilir. İsim boyu 16 baytı geçemez. İsim verildikten sonra tune2fs programı ile değiştirilebilir. Verilen etiket ismi daha sonra mount ile fsck komutunda veya /etc/fstab dosyasının içinde kullanılabilir.

ext2 dosya sistemi, gömülü sistemler için biçilmiş kaftandır. Çok az disk maliyeti ile kurulurlar. Yani dosya sisteminin iç yapısı mevcut bölümün çok küçük bir kısmını işgal eder. Çok uzun zamandır ext2 konusunda hiç bir böcü[7] rapor edilmemiştir. Fakat çok kötü bir özelliğe sahiptir. Ani kapanmalara[8] karşı çok dayanıksızdır. Bundan dolayı ext2 dosya sistemleri her zaman salt-okunur[9] bağlanmalıdır.

Şu anda piyasada bulunan pek çok bordun kök dosya sistemi log tabanlıdır ve oku/yaz[10] şeklinde mount edilir. Bu çok riskli bir kullanım tarzıdır. Log tabanlı sistemler ani kapanmaya karşı çok dayanıklıdırlar. Ama hiç biri ani kapanmaya karşı %100 garanti vermez. Üstelik log tabanlı sistemler salt-okunur bağlansa bile diske, en azından log kısmına halen yazım mevcuttur.

Bundan dolayı kök dosya sistemi olarak log tabanlı olmayan ve salt-okunur bir dosya sistemi seçmek çok önemlidir. Şu anki en iyi aday ise, SD kartlar için ext2 dosya sistemidir.

SD kartımız da artık her bir iş için hazırdır. Sonraki adımda Linux çekirdeği[11] derlenecek ve karta atılarak çekirdek ile açılış sağlanacaktır.



[4] Bakınız $ man mkfs.ext2 tune2fs

[5] Kamu spotu: man sayfalarını okuyalım, okumayanları uyaralım. İlk problemde hemen google search yapmayalım.

[6] Volume label

[7] Bug

[8] Doğrudan gücün kesilmesi durumu.

[9] Read/only

[10] Read/write

[11] İzmir'liler Linux çiğdemi mi derler acaba?

# Bölüm 6

# Çekirdek

Bu bölümdeki çalışmada Linux çekirdeği, kaynak kodundan yüklenecek ve çapraz derlenecektir.

Linux çekirdeği[1] u-boot[2] tarafından yüklenir. Çekirdek esas olarak, süreçlerin planlanması, bellek yönetimi, cihazların kullanılması gibi, daha çok donanıma yönelik işleri yapan ve neredeyse tamamı C ile yazılmış tek bir programdır. Bu programın ismi, örnek projemiz için `zImage`'dir. Tahmin edileceği gibi ismin zerre kadar bir önemi yoktur.

Bu belge daha çok komut bazında ilerleyeceği için çekirdek konusunda fazla yazı yazılmayacaktır. Çekirdeğin düzgün derlenmesi hayati derecede önemlidir ve ustalık gerektirir. İlk derlemelerde genelde sistemin açılması istenir. Sistemi ilk açan çekirdek genelde çok kaba sabadır. İçinde gereksiz pek çok modül ve istenmeyen pek çok özellik bulunacaktır. Gömülü sistem bir kez açıldıktan sonra, adım adım gereksiz özellikler iptal edilmeli ve çekirdek rafine bir hale getirilmelidir.

Çekirdek rafine bir hale getirilmezse ne olur? Tabii ki sistem yine açılır, ama hem diskte hem ram bellekte[3] fazla yer kaplar. Böcü çıkma olasılığı, çekirdek büyüdükçe artar. Büyük çekirdek daha yavaş yüklenir. Güncellenmesi ve yedeklenmesi daha uzun sürer.

[1] Linux kernel

[2] Kaynak: https://www.denx.de/wiki/DULG/Manual

[3] Çekirdek belleğe oturdu mu asla oturduğu yeri bırakmaz, swap'a girmez ve hep aynı page adreslerini işgal eder.

Bizler bu çalışmayı yaparken, OrangePi çekirdeği ana çekirdek dağıtımına[4] eklenmişti. Genelde ARM çekirdekleri çok geriden gelir ve pek azı kernel.org'a eklenir.

OrangePi'nin web sitesinde "mainline kernel kullanılabilir" [5] dese de bu tam doğru değildir. Bunu anlamak 3 günümüze mal olmuştur. Çünkü bazı yamalar henüz mainline'a eklenmemiştir. Bizler bu eklemeleri aşağıdaki gibi yapacağız.

Muhtemelen bir süre sonra bu yamalar da mainline'a eklenecektir. Fakat en azından bu belge yazılırken mainline kernel'daki mevcut OrangePi çekirdeği işlememektedir.

OrangePi için uygun çekirdek kodu aşağıdaki gibi indirilebilir[6].

```
# Çekirdek kodunun indirilmesi ve yamanması.
# Bilgi için bakınız, /opt/gomsis/bin/kernel

$ cd /opt/gomsis
$ git clone --depth 1 -b v4.9-rc1 git://git.kernel.org/pub/scm/linux/\
      kernel/git/torvalds/linux.git

$ cd linux

$ wget https://patchwork.kernel.org/patch/9365783/raw/ -O sun8i-emac-patch-1.patch
$ wget https://patchwork.kernel.org/patch/9365773/raw/ -O sun8i-emac-patch-4.patch
$ wget https://patchwork.kernel.org/patch/9365757/raw/ -O sun8i-emac-patch-5.patch
$ wget https://patchwork.kernel.org/patch/9365767/raw/ -O sun8i-emac-patch-7.patch
$ wget https://patchwork.kernel.org/patch/9365779/raw/ -O sun8i-emac-patch-9.patch

$ for patch in 1 4 5 7 9 ; do
>    patch -p1 < sun8i-emac-patch-${patch}.patch
> done
```

Çekirdeğin derlenebilmesi için bir adet config dosyasına gerek vardır. Her bord için uygun bir config dosyası çekirdeğin kaynak kodu ile beraber gelir. Eğer uygun config dosyası bulunamıyorsa bu bord'dan vazgeçilmelidir.

[4]Mainline kernel, https://www.kernel.org/

[5]http://linux-sunxi.org/Mainline_Kernel_Howto

[6]Kaynak: https://blog.christophersmart.com/2016/10/23/building-and-booting-upstream-linux-and-u-boot-for-orange-pi-one-arm-board/

Bu bölümün sonuna kadar bütün komutlar `/opt/gomsis/linux` dizini altından girilecektir. ARM mimarisi için make girişleri aşağıdaki gibi elde edilebilir.

```
$ cd /opt/gomsis/linux
$ make ARCH=arm help
```

Ekrana dökülen bilgiler, make komutunun ARM[7] mimariler için kabul ettiği girişlerdir. Başlagıçta bu girişlerden sadece bir kaçı kullanılacaktır. Okuyucu diğer girişleri deneyebilir.

`help` çıktısının sonuna doğru ARM için hazır olarak üretilmiş config dosyalarının isimleri görülebilir. Bu tür dosyalara "defconfig" dosyaları denir ve bütün dosya isimlerinin sonunda `_defconfig` son-eki bulunur. Hazır config dosyaları ARM için `arch/arm/configs` dizini altında otururlar.

OrangePi makineleri için hazır config dosyasının ismi `sunxi_defconfig` ile verilmiştir. Bu dosya aşağıdaki gibi incelenebilir.

```
$ more arch/arm/configs/sunxi_defconfig
```

Çekirdek derlemesinde defconfig dosyasını kullanabilmek için, `/opt/gomsis/linux/` dizini altına, yani kaynak kodunun köküne, `sunxi_defconfig` dosyasını `.config` ismi ile kopyalamak gerekir. defconfig dosyasını kopyalama işi el ile, cp komutu ile yapılabileceği gibi make girişleri ile de aşağıdaki gibi yapılabilir.

```
$ make ARCH=arm clean
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- sunxi_defconfig
```

İlk make komutunda clean girişi[8] kullanılmıştır. clean komutu .config dosyası hariç, derleme sırasında üretilen bütün dosyaları siler.

Sonraki komut ise `sunxi_defconfig` dosyasını .config adı ile kaynak kodunun köküne kopyalar.

`ARCH=arm` tanımı doğrudan Makefile dosyasına değişken olarak gider ve içerde `$(ARCH)` şeklinde kullanılır. Benzer şekilde `CROSS_COMPILE` değişkeni de doğrudan Makefile içinde kullanılır.

[7] ARCH=arm'daki ARCH ifadesi architecture kelimesinden gelir. Desteklenen mimarilerin dizinlerini görmek için `$ ls /opt/gomsis/linux/arch`

[8] Diğer temizleme girişleri için, $ make ARCH=arm help | grep clean

Çapraz derleme ve ARM çekirdeği üzerindeki her türlü make komutu için bu iki değişken gerekli ve yeterlidir. `$ make ARCH=arm help` komutunda olduğu gibi bazen sadece `ARCH` ifadesi yeterlidir. Fakat fazladan bir değişken atamasının hiç bir zararı yoktur.

Hazır config dosyaları kesinlikle doğrudan kullanılmamalı ve üzerinde güncelleme yapılmalıdır. Sebebi çok basittir. Her zaman hazır config dosyalarında, bordumuza veya projemize gerekli olmayan pek çok özellik mevcuttur.

Örneğin, OrangePi'de tek bir eternet kartı olmasına rağmen, pek çok eternet kartının sürücüsü config dosyasında mevcuttur. ReiserFS gibi akla zarar bir dosya sistemi çekirdeğe eklenmiştir. Ya da ext2 gibi işimize yarayacak bir dosya sistemi çekirdekten çıkarılmıştır. Ya da seçtiğimiz kök dosya sistemi modül olarak[9] derlenmiştir. Bunun gibi sıkıntıları vardır hazır config dosyalarının.

Bu sebeplerden dolayı mevcut .config dosyası menuconfig girişi ile güncellenmelidir. Bunun için aşağıdaki gibi menuconfig girişi kullanılabilir.

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

Çıkan ekranda uygun güncellemeler yapılmalıdır. İlk defa bu işi yapacak olanlar için burası tam bir muammadır. Çekirdek derlemeye alışmak için sadece belirli bir menü seçilmeli ve alışana kadar orası ile oyananmalıdır. Örneğin "File Systems" menüsüne girilebilir ve her ekranın altında bulunan "help" menüsü okunmalıdır. Bu çok zahmetli olsa da vakit ayrılmalı ve bütün girişlerin help ekranları okunmalıdır.

Çok uzayacağı için burada girişleri işaretleme konusuna değinilmeceyektir. İnternette haddinden fazla kaynak vardır.

Bizlerin örnek olarak kullandığı config dosyası `/opt/gomsis/configs/kernel.config` altında bulunabilir. Bu dosya `.config` adı ile kaynak kodunun köküne aşağıdaki gibi kopyalanabilir.

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

Tekrar `menuconfig` girişi yapılarak bizlerin yaptığı seçimler incelenebilir. Çeşni olması için 1 adet modül seçilmiştir. Okuyucu istediği özelliği modül olarak ekleyebilir. Sadece kök dosya sistemi desteği modül olamaz.

[9] Kök dosya sistemleri modül olarak derlenmezler.

`.config` dosyası oluşturduktan sonra aşağıdaki gibi derleme yapılabilir.

```
$ cd /opt/gomsis/linux
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage -j$(nproc)
$ ls -l arch/arm/boot/zImage
```

`zImage` girişi ile derleme başlar. Derleme sonunda elde edilen `zImage` isimli çekirdek `ls` komutu ile incelenebilir.

`make menuconfig` sırasında, bazı girişler `<M>` veya {`M`} olarak işaretlenmiştir. Bu tür işaretlenen özelliklere modül denir ve çekirdeğin kodu içinde bulunmazlar, ayrı bir yerde, genelde kök dosya sistemi içinde `/lib/modules` içinde bulunurlar.

Modüller, Linux ayakta ve çalışırken, ihtiyaç halinde çekirdeğe dahil edilirler ve istenirse tekrar çekidekten çıkarılırlar. Böylece çekirdek çok modüler bir yapıya sahip olur ve çekirdeğin kod boyu bu sayede inanılmaz derece küçülür.

Modüller çekirdekten ayrı oldukları için, zImage girişi ile derlenemezler. Modüller, aşağıdaki gibi ayrı bir make girişi ile derlenirler.

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

Modüller her zaman tek bir dosyadır ve sonları `.ko`[10] uzantısı ile biter. Şu ana kadar zImage ve çeşitli *.ko dosyaları elde edilmiştir.

Bir de son olarak `dtb`[11] dosyası elde etmek gerekir. dtb dosyası, bordun donanımına ait parametreleri içinde barındırır.

dtb dosyası `dts`[12] dosyalarından, derleme ile elde edilir. dts dosyası C yapısını[13] fazlasıyla andırmaktadır. Örnek bir dts dosyası aşağıdaki gibi incelenebilir.

```
$ more arch/arm/boot/dts/sun8i-h3-orangepi-one.dts
```

Görüleceği gibi, donanım hakkında pek çok parametre bir C yapısı gibi tanıtılmıştır. dts kavramı çekirdeğe katılmadan önce buradaki bilgiler doğrudan

[10] Kernel object
[11] Device tree blob
[12] Device tree source
[13] C structure

çekirdek kodu içine gömülmekteydi. dts sayesinde kernel önce açılmakta ve sonra dtb dosyasını yükleyerek donanım hakkında gerekli bigilere sahip olmaktadır.

Donanımda yapılacak güncellemelerde, kernel derleme yerine sadece dts üzerinde değişiklik yapılması yeterli olacaktır. Özellikle referans dizaynlar üzerinde geliştirme veya güncelleme yapanlar hemen hemen sadece dts güncellemeleri ile sistemlerini kurabilmektedirler.

Tahmin edileceği gibi, pek çok dts dosyası vardır ve bu dosyalar metin tabanlı veya kısaca ASCII'dir. ASCII dosyayı çekirdeğin tanıması mümkün değildir. dts dosyaları dtc[14] derleyicisiyle çekirdeğin anlayabileceği dtb dosyaları haline getirilir.

dtb, dtc ve dts kavramları, aşağıdaki benzetme ile daha anlaşılır bir hale getirilebilir.

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

hello.c isimli C programı gcc ile derlenir ve hello isimli çalışabilir kod üretilir. hello.c programı metin tabanlıdır veya ASCII'dir. gcc ile derlenir ve çalışabilir hello kodu elde edilir.

Benzer şekilde `foo.dts` dosyasına C programı gözü ile bakılabilir. dtc komutu ile `foot.dts` derlenir ve `foo.dtb` dosyası elde edilir. Bu dosya tabii ki yürütülebilir değildir ama çekirdek tarafından, açılış sırasında doğrudan yüklenebilir.

dts dosyaları, dtc[15] komutu ile derlenebilir. Fakat bu zahmetli bir iştir. Aşağıdaki gibi make girişi ile dts dosyaları, çekirdeğin anlayacağı dtb dosyalarına çevrilir.

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- dtbs
$ ls -l arch/arm/boot/dts/sun8i-h3-orangepi-one.dtb
```

Sunxi firmasının pek çok bilgisayarı ve aynı bilgisayarın pek çok donanım konfigürasyonu için pek çok dtb dosyası üretilir. OrangePi Zero için `sun8i-h3-orangepi-one.dtb` dosyası kullanılacaktır.

[14] Device tree compiler
[15] $ man dtc

Bu ana kadar çekirdek derlenmiş ve 3 tip dosya elde edilmiştir. Bu dosyalar aşağıdaki tabloda özetlenmiştir.

**zImage** Çekirdek.
`$ make zImage` ile elde edilmiştir.
İncelemek için `$ ls -l arch/arm/boot/zImage`

***.ko** Modüller.
`$ make modules` ile elde edilmiştir.
`zImage`'ye ait olan, ama zImage yerine `/lib/modules` altında oturan programlar.
İncelemek için, `$ find . -name "*.ko" -exec modinfo {} \;`

**dtb** Device tree blob.
`$ make dtbs` ile elde edilmiştir.
Cihaza ait donanım tanımlarını[16] barındırır.
İncelemek için, `$ ls -l arch/arm/boot/dts/sun8i-h3-orangepi-one.dtb`

Artık çekirdeğimiz teste hazırdır. Çekirdeği test edebilmek için, `zImage` ve `dtb` dosyasını, SD kartın 1. bölümünde /boot altına kopyalamak yeterlidir. Kopyalama işi aşağıdaki gibi yapılabilir.

```
$ mkdir /mnt/disk1                # mount point yarat.
$ mount /dev/mmcblk0p1 /mnt/disk1 # SD'nin ilk bölümünü bağla.
$ mkdir /mnt/disk1/boot           # Ilk bölüme boot dizinini kur.
$
$ cd /opt/gomsis/linux

$ cp arch/arm/boot/zImage /mnt/disk1/boot # Çekirdeği kopyala.
$ cp arch/arm/boot/dts/sun8i-h3-orangepi-one.dtb /mnt/disk1/boot/dtb

$ umount /mnt/disk1
```

İşlemler çok basit olduğu için ayrıca anlatılmıştır. Okuyucu istediği başka bir yöntemle de aynı işlemi gerçekleyebilir. SD kartın 1. bölümünün cihaz ismi olan `/dev/mmcblk0p1` düğümü, okuyucunun makinesinde farklı olabilir. Dikkat etmek gerekir.

[16] Device description

`sun8i-h3-orangepi-one.dtb` ismi çok uzun olduğu için `dtb` adı ile kopyalanmıştır. Kopyalama işleri tamamlanınca, SD kartın birinci bölümünde boot/zImage ve boot/dtb dosyaları yaratılmış olacaktır.

Bir önceki bölümde u-boot ile bordu açmıştık. Şimdi SD kartı tekrar borda takıp, bordu açıp, u-boot seviyesinde bir tuşa basarak açılışı durduralım, aşağıdaki komutları girelim ve gelen sonuçları inceleyelim.

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

> bdinfo

arch_number = 0x00000000
boot_params = 0x40000100
DRAM bank   = 0x00000000
-> start    = 0x40000000
-> size     = 0x10000000
baudrate    = 115200 bps
TLB addr    = 0x4FFF0000
relocaddr   = 0x4FF62000
reloc off   = 0x05F62000
irq_sp      = 0x4BF3DE90
sp start    = 0x4BF3DE80
Early malloc usage: 10c / 400
```

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

Aşağıdaki 2 komut ile çekirdek ve dtb dosyası bordun kendi belleğine yüklenir.

```
> ext2load mmc 0 0x46000000 boot/zImage
> ext2load mmc 0 0x49000000 boot/dtb
```

SD kartın 1. bölümünde ext2 dosya sistemi mevcuttu. Bundan dolayı ext2load komutu kullanılmıştır. ext2load komutu, SD kartta bulunan ve ext2 formatlanmış bir dosya sisteminden RAM belleğe dosya yükler.

`mmc 0` ifadesi, 1. SD kart demektir. İkinci bir SD kart olsaydı, `mmc 1` kullanılacaktı. OrangePI Zero makinesinde her zaman 1 adet SD kart okuyucu bulunmaktadır. Bundan dolayı bütün *load[17] komutlarında `mmc 0` ifadesi kullanılır.

`mmc 0` ifadesi aslında `mmc 0:1` ifadesinin kısa yazılmış halidir. ":" dan sonra bölüm numarası[18] verilir. Örneğin SD kartın 2. bölümünden bir dosya belleğe yüklenecek olsa, `mmc 0:2` kullanılacaktı.

Sonraki 0x46000000 sayısı, dosyanın yükleneceği belleğin başlangıç adresidir. `bdinfo` komutunda bordun fiziksel RAM adresinin 40000000 değerinden başladığı tespit edilmişti. Bu adrese tamamen keyfi ama çok da yukarılarda olmayan bir değer eklenerek çekirdeğin yükeleneceği adres verilmiştir. Kullanıcı çeşitli keyfi adresler deneyebilir. Çok yukarıda olmaması yani 40000000'e çok yakın olmaması ve mevcut adres uzayının içinde kalınması yeterlidir.

boot/zImage ifadesiyle de, yüklenecek dosyanın yani çekirdeğin ismi verilir. Tahmin edileceği gibi boot/ dizini tamamen keyfidir. Okuyucu zImage'yi nereye kopyaladıysa, ext2load komutunda bu ismi vermesi yeterlidir. Defacto standard olduğu için /boot dizini seçilmiştir ama bir zorunluluk yoktur.

Özetle `ext2load mmc 0 0x46000000 boot/zImage` komutu, SD kartın 1. bölümünde, ext2 dosya sisteminde oturan `boot/zImage` dosyasını 46000000 adresine yükler.

Benzer şekilde `ext2load mmc 0 0x49000000 boot/dtb` komutu da dtb dosyasını 49000000 adresine yükler. Bu adres de tamamen keyfidir. Adres uzayının içinde kalınması ve çekirdek ile birbirlerini ezmemesi yeterlidir. Okuyucu farklı adresler deneyebilir.

Son olarak aşağıdaki komut ile çekirdeğin çalışması sağlanabilir.

```
> setenv bootargs console=ttyS0,115200
> bootz 0x46000000 - 0x49000000
```

setenv komutu, u-boot komutudur ve değişkenlere atama yapılmasını sağlar. Diğer bir deyişle, C veya bash programlamada `x=1` şeklinde yapılan bir atama, u-boot'un etkileşimli ortamında `setenv x 1` şeklinde yapılır.

Örnekte verilen setenv komutu, bootargs isimli bir değişkene atama yapar. setenv sonucunda bootargs değişkeni "console=ttyS0,115200" değerini alır.

[17] fatload, ext2load, vb.
[18] Partition number

bootargs değişkeninin sağında bulunan bütün ifadeler, gözü kapalı bir biçimde çekirdeğe aktarılır. Örneğin aşağıdaki gibi bir ifade yazılırsa, çekirdeğe A=3 ve B="hava guzel" değişkenleri aktarılır.

```
> setenv bootargs console=ttyS0,115200 A=3 B="hava guzel"
```

Diğer bir deyişle, u-boot, bootargs değişkeninin sağ tarafı ile hiç ilgilenmez ve gözü kapalı bir biçimde sağ tarafın içeriğini 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.

Çekirdeğe parametre aktarmanın envai çeşit yolu vardır. Bu teknik[19] çok eski olmasına rağmen halen kullanılmaktadır.

bootz[20] komutu daha önce bordun belleğine yüklü çekirdeği kullanarak cihazı açar.

`bootz 0x46000000 - 0x49000000` örneğinde 3 parametre kullanılmıştır.

1. parametre 0x46000000 ile verilmiştir ve çekirdeğin bellek adresini gösterir. ext2load komutu ile verdiğimiz keyfi adrestir.

2. parametre initramfs adresidir. Örnek projemizde initramfs olmadığı için "-" girilmiştir.

3. parametre 49000000[21] ile verilmiştir ve dtb'nin bellek adresini gösterir. ext2load komutu ile verdiğimiz keyfi adrestir.

bootz komutu önce bootargs ile verilmiş parametreleri start+0x100 adresine yazar. Sonra dtb'nin adresini çekirdeğe bildirir ve yürütmeyi çekirdeğe verir.

Çekirdek açılmaya başlar, start+0x100 adresinden, bootargs ile kendisine verilen parametreleri okur. Sonra dtb'yi yükler ve açılışını tamamlar. Son olarak kök dosya sistemini mount eder ve yürütmeyi init veya benzer bir programa verir. Şu anda kök dosya sistemi henüz olmadığı için bord tıkanacaktır.

[19] Bakınız http://mirror.fsf.org/pmon2000/3.x/src/pmon/arch/arm/linux_at.c

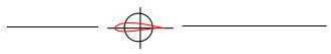
[20] Boot Linux zImage image from memory

[21] u-boot sistemi, bütün sayıları 16'lık sistemde kabul eder. Öyle ki u-boot, önünde 0x yazmasa dahi, gelen sayıyı 16'lık kabul eder.

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

Çekirdeğin `.config` dosyası aşağıdaki gibi, `kernel.config` adı ile `/opt/gomsis/configs/` dizini altına yedeklenebilir.

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



# Bölüm 7

# Kök Dosya Sistemi

Çekirdeğin açıldıktan sonra ilk bağladığı dosya sistemine kök dosya sistemi[1] denir. Çekirdek kök dosya sistemini bağladıktan sonra belirli sırada bazı programları[2] çalıştırmayı dener. Bu programlar sıra ile `/sbin/init, /etc/init, /bin/init` ve `/bin/sh` ile verilmiştir.

O halde bir dizin yapısı kurup, örneğin `/sbin` dizini altına `init` isimli bir program atarsak, çekirdek doğrudan bunu çalıştıracaktır. Şimdi böyle bir dizin yapısı adım adım inşa edilecek ve içleri gerekli programlarla doldurulacaktır.

Kök dosya sistemi kuruluşu, ilk başta çok karışık gelebilir. Okuyucu her adımı farkında olarak yapmalıdır. Kök dosya sistemi bir kez kurulduktan sonra pek çok açılış tekniğinde doğrudan kullanılabilir. Bu belgede 2 farklı açılış tekniği uygulanacaktır.

Kök dosya sisteminin kuruluş mentalitesi çok basittir. Önce boş dizinler yaratılır, sonra her bir dizinin içi çeşitli yerlerden kopya yapılarak doldurulur. Şimdi adım adım bu işlemler yapılacaktır.

Öncelikle aşağıdaki gibi iskelet bir kök dosya sistemi[3] kurulur.

```
$ cd /opt/gomsis
$ mkdir RootFS.skel
```

[1] Root file system

[2] Çekirdeğin kaynak kodunda, bakınız `/opt/gomsis/linux/init/main.c`'deki kernel_init() fonksiyonu.

[3] Root file system skeleton

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

İçin tamamen boş olan çeşitli dizinler kurulmuştur. Bütün dizinler `RootFS.skel` dizini altında yaratılmıştır. `RootFS.skel` ismi tamamen keyfidir. Yıllardır aynı ismi kullanmaktayız, okuyucu istediği ismi verebilir. Örneğin buildroot sisteminde iskelet kök dosya sisteminin oturduğu dizine `skeleton` adı verilmiştir.

Örnek bir iskelet kök dosya sistemi `/opt/gomsis/RootFS.skel` altında bulunabilir. Okuyucunun tek tek dizinleri kurmasına gerek yoktur.

Bütün dizinler veya alt dizinler boş olmasına rağmen sadece `/etc` dizini içinde birkaç dosya vardır. Bu dosyaların iç yapıları son derece basittir ve herhangi bir Linux dağıtımından veya üzerinde çalışılan host makineden elde edilebilir.

İskelet dosya sisteminde `/etc` dizini altındaki dosyalar kısaca aşağıda özetlenmiştir.

**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 numaraları.

**rcS** Açılış betiği. /etc/inittab tarafından tetiklenir.

**resolv.conf** ../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 betik.

Okuyucu `$ man 5 dosya_ismi` komutu ile her bir dosya için kılavuz sayfalarına bakılabilir. Bazı dosyaların kılavuz sayfaları olmayabilir. Bütün dosyalar metin tabanlı olduğu için `more` komut ile içine bakılabilir.

İskelet dosya sistemindeki diğer boş dizinler çeşitli kaynaklardan doldurulacaktır. Aşağıdaki bazı boş dizinlerin hangi kaynaklardan doldurulacağı kısaca verilmiştir. İlerleyen bölümlerde her biri üzerinde tek tek durulacaktır.

**/bin** busybox ile.

**/sbin** busybox ile.

**/usr/bin** busybox ile.

**/usr/sbin** busybox ile.

**/lib/modules** Çekirdek ile.

**/lib/firmaware** Çekirdek ile.

**/lib** Toolchain ile.

**/usr/lib** Toolchain ile.

**/tmp** rcS ile mount.

**/proc** rcS ile mount.

**/sys** rcS ile mount.

**/etc** Örnek herhangi bir sistemden kopya ile.

**/dev** Açılışta kernel ile veya rcS mount ile veya mdev ile.

## 7.1 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.

Pek çok komut çok az seçenek[4] ile ve daha basit halleri ile yeniden yazılmış ve tek bir yürütülebilir dosya altında toplanmıştır. Bu projeye busybox projesi[5] ve elde edilen çalışabilir dosyaya `busybox` denir.

Busbox'ın mantığı aşağıda verilen programdaki gibi düşünülebilir. Kolay okunması için C'ye benzer bir biçimde kodlama yapılmıştır. Tam da C değildir.

```
// apletler

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 sisteminin ana girişi.
//
int main(int argc, char *argv[]){

   if ( argv[0] == "mkdir" ) return mkdir_main(argc, argv);
   if ( argv[0] == "ls"    ) return ls_main(argc, argv);
   if ( argv[0] == "cd"    ) return cd_main(argc, argv);
   ...

   if ( argv[0] == "busybox"){
```

[4] Options

[5] https://busybox.net/

```
      // Bu sefer argv[1]'e göre apletler birbirlerinden
      // ayırd edilirler.
      }

   ...
   err("applet not found");
}
```

Yukarıdaki örnekte, mkdir, ls ve cd komutları, birileri tarafından[6] el yordamı ile yazılmış sisteme eklenmiştir. Busybox komutlarına aplet denir.

Busybox programı derlendikten sonra, içinde onlarca komut veya aplet barındıracaktır. Fakat sonuçta tek bir dosya elde edilecektir. Tek bir dosya bütün komutları basitçe yürütemez. Bunun için her bir komut bir sembolik link ile busybox programına bağlanmıştır. Örnekte verilen 3 apletin çalışması için aşağdaki gibi sembolik linkler kurulabilir.

```
$ ln -s busybox mkdir
$ ln -s busybox ls
$ ln -s busybox cd

$ ls -l

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

Görüleceği gibi, bütün apletler busybox programını gösterir. Herhangi bir busybox komutu yani aplet girildiğinde, sembolik linkler doğrudan busybox'ı çalıştırır. Busybox da argv[0] denetimi ile hangi linkin geldiğini anlar ve ilgili apletin fonksiyonunu çağırır.

Buradaki bütün incelik, sembolik linklerde, argv[0] olarak link'in bittiği dosya, yani busybox programının kendisi değil, link'in başlagıcı olan aplet isminin kabul edilmesidir. Böylece tek bir program bütün komutları birbirinden ayırt edebilecektir.

Sembolik link kullanmadan da busybox apletleri çalıştırılabilir. Örneğin `mkdir` ile `/tmp/test` dizini aşağıdaki gibi kurulabilir.

[6] https://en.wikipedia.org/wiki/BusyBox

```
$ busybox mkdir /tmp/test
```

Bu kullanım tarzında önce busybox komutu sonra aplet ismi girilir. Bu tarz pek de kullanışlı değildir.

Bu kadar laftan sonra busybox sistemini indirip, derleyip kuruluşunu yapabiliriz. Öncelikle aşağıdaki gibi kaynak kodu indirilir.

```
$ cd /opt/gomsis

$ git clone git://busybox.net/busybox.git
$ cd busybox
$ git branch -a
$ git checkout remotes/origin/1_26_stable # Son kararlı sürüme geç.
$ git branch # Sürümü gör.
```

Her zaman en sonra kararlı sürüm[7] kullanılmalıdır. Busybox sistemi içinde pek çok aplet mevcuttur. Okuyucu kullanacağı komutları, aşağıdaki gibi menuconfig girişi ile seçebilir.

```
$ cd /opt/gomsis/busybox
$ make clean
$ make menuconfig
```

Başlangıçta bütün apletler hemen hemen seçili gelir. Okuyucu sadece inceleme yapmalı ve save ile çıkmalıdır. Daha sonra, açılışı hızlandırması ve az yer kaplaması için sadece işletim sisteminde gerekli olan apletler seçilmelidir.

Gerekli apletler nasıl tespit edilir?

Çok basit. Komutları yani apletleri her zaman `/etc` dizini altında bulunan betikler[8] kullanır. Bu betiklerdeki komutları ve `init` apletini seçmek açılış için yeterlidir. Bu arada uygulama programlarının içinde `exec` veya `system` veya `popen` gibi sistem çağrıları da aplet çağırabilir. Bu tür apletler de ayrıca seçilmelidir. Bugün sadece 10 adet komut ile açılabilen profesyonel gömülü sistemler mevcuttur.

`make menuconfig` ile aplet seçimi yapıldıktan sonra, seçilen apletleri barındıran busybox çapraz derlenmelidir. Çapraz derleme aşağıdaki gibi basitçe yapılabilir.

[7] Stable version
[8] Scripts

```
$ cd /opt/gomsis/busybox

$ make CROSS_COMPILE=arm-linux-gnueabihf-

$ ls -l busybox

$ file busybox

busybox: ELF 32-bit LSB executable,
         ARM,
         EABI5 version 1 (SYSV),
         dynamically linked,
         interpreter /lib/ld-linux-armhf.so.3,
         for GNU/Linux 2.6.16,
         BuildID[sha1]=70fcc1933ddc82e043c15dfbb33f88b4d6364399,
         stripped
```

`ls -l busybox` komutu ile busybox sisteminin boyu incelenebilir. Onlarca apletin kapladığı alan, standard bir Linux komutundan daha az olacaktır.

`file` komutu ile de derlenen kodun çapraz derlenip derlenmediği kontrol edilebilir. Kolay okunması için çıkış satır satır gösterilmiştir. Burada ARM lafını görmek gerekir.

Derleme bitmiştir. Şimdi bütün seçilen apletler için sembolik linkler üretilmelidir. Bu çok zahmetli iş tabii ki install komutu ile aşağıdaki gibi basitçe gerçeklenir.

```
$ cd /opt/gomsis/busybox
$ make CROSS_COMPILE=arm-linux-gnueabihf- install
```

`make install` komutu, `/bin, /sbin, /usr/bin, /usr/sbin` dizinlerini yaratır ve bütün sembolink linkleri `busybox/_install/` altında kurar. Okuyucu mutlaka bu dizine girmeli ve içini incelemelidir.

`_install/` dizini içinde de görüleceği gibi busybox isminde tek bir yürütülebilir dosya ve pek çok sembolik link mevcuttur. Artık kök dosya sistemiminizin bütün komutları mevcuttur. Şimdi üretilen bu komutlar sonraki bölümde kök dosya sistemindeki yerlerine taşınacaktır.

Busybox sisteminin .config dosyası aşağıdaki gibi, busybox.config adı ile configs/ dizini altına yedeklenebilir.

```
$ cd /opt/gomsis
$ cp busybox/.config configs/busybox.config
```

## 7.2 RootFS'in kuruluşu

`RootFS.skel` dizini altında sadece `/etc`'si dolu olan, pek çok dizin yaratılmıştı. Şimdi bu dizinler sırayla doldurulacaktır. Orjinal RootFS.skel yapısını bozmamak için öncelikle bu dizin RootFS altına aşağıdaki gibi kopyalanır.

```
$ cd /opt/gomsis
$ rm -fr RootFS            # Eskiden kalma RootFS varsa sil.
$ cp -a RootFS.skel RootFS # Iskeletin kopyasını çıkar.
```

Bu belgenin sonuna kadar artık RootFS üzerinde işlem yapılacaktır. Şu anda RootFS içindeki sadece `/etc` dizini doludur, diğerleri boştur. Boş dizinler sıra ile aşağıdaki gibi doldururlur. Bütün komutlar, aksi belirtilmedikçe `/opt/gomsis` dizini altından verilecektir. Yapılacak işlerin özeti Şekil 7.1'de verilmiştir.

### 7.2.1 /boot

`boot` dizini içinde çekirdek ve dtb dosyası bulunmalıdır. Bu 2 dosya aşağıdaki gibi kopyalanır.

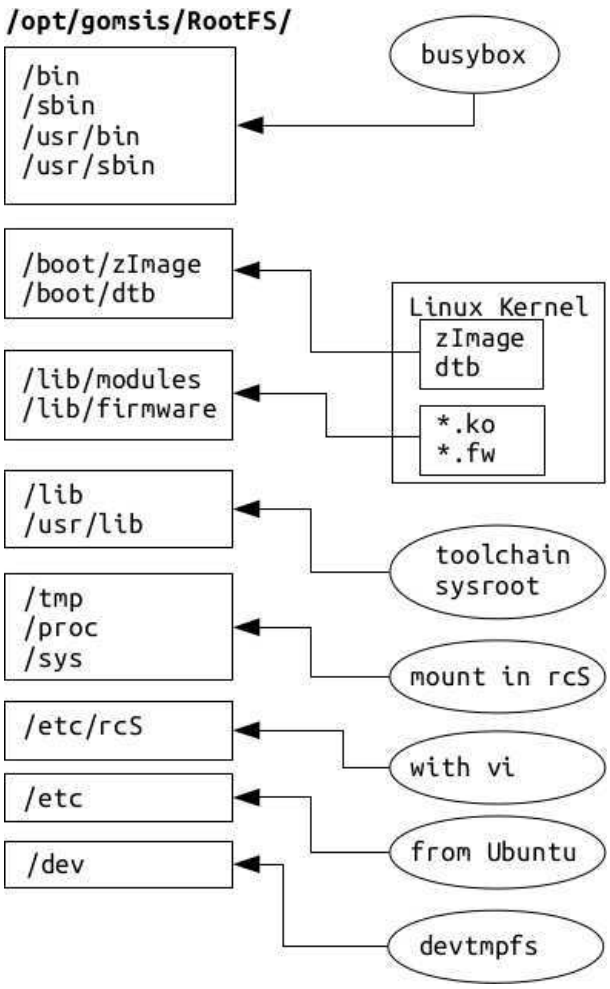
```
$ cp linux/arch/arm/boot/zImage                          RootFS/boot
$ cp linux/arch/arm/boot/dts/sun8i-h3-orangepi-one.dtb RootFS/boot/dtb
```

### 7.2.2 /bin, /sbin, /usr/bin, /usr/sbin

`*bin/` dizinlerinde Linux komutları oturur. Bu komutları önceki adımda busybox ile üretmiştik. Şimdi üretilen bu komutlar uygun dizinlere aşağıdaki gibi kopyalanabilir.

```
$ cp -a busybox/_install/* RootFS/
$ rm RootFS/linuxrc
```

Tekrar hatırlatmak gerekirse, busybox derlenirken, `make install` girişi ile, bütün komutlar `_install/` dizini altına kopyalanmıştı. `_install` dizini altında `/bin, /sbin, /usr/bin, /usr/sbin` alt dizinleri bulunmaktadır. Yukarıdaki `cp` komutu ile bu 4 dizin `RootFS` altındaki aynı dizinlere kopyalanacaktır.




Şekil 7.1: Kök dosya sisteminin çeşitli kaynaklardan inşa edilmesi.

Bir de fazladan `linuxrc` linki mevcuttur. `linuxrc` dosyası çok eski initrd tarzındaki açılışların başlagıç dosyasıdır. Artık bu teknik kullanılmadığı için `linuxrc` dosyası `rm` komutu ile silinmiştir.

### 7.2.3 /lib

Kütüphaneler `toolcahin`'den elde edilir. Bütün toolchain paketleri içinde sysroot denilen özel bir dizin yapısı vardır. include ve kütüphaneleri barındıran dizinlere sysroot denir.

sysroot dizinleri genelde çok karışık bir alanda bulunur ve bunun için bir standard yoktur. Kullanılan toolchain'in sysroot dizinini öğrenmek için `--print-sysroot` seçeneği aşağıdaki gibi kullanılabilir.

```
$ arm-linux-gnueabihf-gcc --print-sysroot
```

Bu seçenek sysroot dizinlerinin mutlak yol isimlerini[9] verir. Kopyalama işlemini basitleştirmek için yol ismi `SYSROOT` isimli bir değişkene aşağıdaki gibi aktarılabilir.

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

`SYSROOT` adının hiç bir önemi yoktur. Okuyucu istediği ismi verebilir. Kütüphaneler aşağıdaki gibi kopyalanabilir.

```
# 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/
```

[9] Absolute path names

İlk 3 kütüphane zorunludur. Bu kütüphaneler olmazsa 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[10] 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.

Yukarıdaki 3 kütüphane dinamik derlenmiş bütün programlar için olmazsa olmazdır. Bu kütüphanelerden birisi eksik olursa login gelmez.

Eğer busybox, statik derlenirse bu kütüphanelerin hiç birine gerek kalmaz. Çünkü kütüphaneler otomatik olarak busybox kodunun içine eklenir. `lib` dizinine dahi gerek yoktur. Çok özel çalışamalar hariç bu tür bir uygulama hiç tavsiye edilmez.

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

Bir uyglama için gerekli kütüphaneyi tespit etmek çok kolaydır. Uygulama ARM makinede işletilir, eksik kütüphane varsa hata olarak ekrana raporlanır. Bu kütüphane hemen toolchain içinden `RootFS/`'e veya doğrudan ARM makineye kopyalanır.

Gerekli kütüphaneleri bulmak için başka yollar da vardır, `objdump` veya `ldd` komutlarını kullanmak veya `strace` ile uygulama programını çalıştırmak gibi.

Zorunlu kütüphaneler dışında 4 kütüphane daha kopyalanmıştır. Muhtemelen başka kütüphanelere de ihtiyaç olabilir. Kopyalanan bu kütüphaneler login'e kadar sistemi getirebilmekte ve dışarından telnet ile erişime izin vermekte ve internete çıkışı sağlamaktadır. Daha iyisi can sağlığı.

Kütüpahe kopyalarken çok büyük bir incelik vardır ve uyulmazsa saç baş yoldurur. Kopyalama yaparken, kütüphane toolchain'de nerede oturuyorsa, RootFS'de de tam aynı yerde oturmalıdır.

[10] Interpretper

Örneğin toolchain'de `.../lib/falan/hede.so` kütüphanesi `lib/falan` dizini altında bulunmaktadır. Daha genel bir ifade ile `hede.so` kütüphanesi `$SYSROOT/lib/falan` dizini altında oturmaktadır. Bu durumda `hede.so` kütüphanesi `RootFS/` içinde de, benzer şekilde `RootFS/lib/falan` altında oturmalıdır. Sebebi de çok basittir.

Toolchain derleme yaparken, yani busybox veya bir uygulama programı veya herhangi bir programı derlerken kullandığı kütüphanelerin `PATH` bilgilerini program içine gömer. Daha sonra ARM makinede bir program çalışırken, gerekli kütüphaneler yine aynı `PATH` içinde aranırlar. Aranan kütüphane farklı bir `PATH` içinde ise sorun çıkacaktır.

Aslında bu sorunu çözmenin tabii ki başka yolları da vardır. Fakat en temizi kütühanelerin `SYSROOT/` ve `RootFS/` dizinleri arasında birebir aynı `PATH` ile kopyalanmasıdır.

### 7.2.4 /lib/modules, /lib/firmware

Çekirdek içinde olması gereken ama az yer kaplasın diye modül olarak derlenen programların `/lib/modules` altında atılması gerekir. Yürütme zamanında ihtiyaç duyulan modüller buradan yüklenir. Modüller bulunmak zorunda değildir. `make menuconfig` sırasında modüller işaretlenmemişse, üretilmezler. Bu durumda bu bölümdekileri yapmaya gerek yoktur.

Çekirdek derlemesi bittikten sonra mevcut modüller yani `*.ko` dosyaları doğrudan find komutu ile bulunarak `RootFS/lib/modules` altında kopyalanabilir. Müşteriye verilecek sistemde bu şekilde yapılmalıdır.

Profesyonel bir gömülü sistemde modül sayısı birkaç taneyi geçmez. İdealinde de hiç olmaması gerekir. Çünkü gömülü sistemler katıdır. Kullanılacak her cihaz önceden bellidir ve tak-çıkar şeklinde bir iş pek nadir yapılır. Ya da modül yüklemesi gereken uygulamalar genelde çok ama çok azdır.

Modülleri `find` komutu ile bulup kopyalamak yerine aşağıdaki gibi çekirdeğin `make` girişi ile de kopyalama yapılabilir.

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

`make modules_install` komutu bütün modülleri, modül bağımlılıklarını vs.

bulur ve RootFS/lib/modules/<Çekirdek_Sürümü>/ dizini altına kopyalar.

Modüllerin yükeleneceği dizin INSTALL_MOD_PATH ile verilir.

Aynı kök dosya sistemini birden fazla çekirdek kullanabilir. Her çekirdeğin de kendine ait modülleri vardır. Modüller birbirleri ile karışmasın diye çekirdek sürümleri bilgisi ile ayrı bir dizin kurulur. Böylece her çekirdeğin modülleri, kendi çekirdek numarasına ait dizin altına atılır. Çalışma sırasında ise uname -r komutu ile çalışan çekideğin sürümü elde edilir ve uygun dizine geçilir.

Eğer aynı numaralı 2 çekirdek varsa, karışıklık olmaması için, make menuconfig sırasında, çekirdeğe bir son ek verilir. Böylece aynı çekirdekler birbirleri ile karışmazlar.

### 7.2.5 /lib/firmware

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

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

firmware dosyaları çekirdeğe doğrudan binary olarak gelirler. Kodları kapalıdır. Bundan dolayı ayrıca modüller gibi derlenmezler. Doğrudan install edilirler. Diğer bir deyişle bütün firmare dosyaları doğrudan /lib/firmware altına kopyalanır.

Firmware dosyalarında, modüllerdeki gibi çekirdek sürümü diye bir kavram yoktur. Firmware'ler doğrudan ilgili donanım içine çekirdek tarafından gönderilirler.

Kök dosya sisteminin kuruluşu bu adımla son bulmuştur. Kök dosya sisteminde bulunan diğer dizinler açılış sırasında /etc/rcS betiği tarafından veya doğrudan kernel tarafından kurulurlar.

Sistem açıldıktan sonra diğer dizinlerin nasıl doldurulduğu veya kullanıma alındığı aşağıda kısaca bahsedilmiştir. Bu adımdan sonra okuyucu terminalden herhangi bir komut girmemelidir. Aşağıdaki yazılar sadece bilgi içindir.

### 7.2.6 /dev

`/dev` dizini içinde cihazların düğüm isimleri mevcuttur. Bu dizinini kurmanın pek çok yolu vardır. En pratiği ve günceli bu dizini doğrudan çekirdeğe kurdurmaktır.

Çekirdek derlemesi sırasında, `make menuconfig` girişinde, aşağıdaki gibi seçim yapılırsa, `/dev` dosya sistemi açılışta otomatik olarak kurulacak ve bizim `/dev` dizinine bağlanacaktır.

```
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, çekirdeğe kendi içinde bir `/dev` dosya sistemi kurmasını söyler. Bu dosya sistemine `devtpmfs` dosya sistemi denir. Yani `ext2` veya `vfat` gibi bu da bir dosya sistemidir. Çekirdek tarafından otomatik olarak RAM bellekte kurulur.

`Automount devtmpfs at /dev, after the kernel mounted the rootfs` seçeneği ile de kök dosya sistemi mount edildikten hemen sonra çekirdek içinde kurulan `devtmpfs` dosya sisteminin bizim `/dev` dizinine mount edilmesi sağlanır.

Bu seçenek `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.

### 7.2.7 /proc, /sys

`/proc` ve `/sys` dizinleri sözde[11] dosya sistemlerine mount edilirler. Sözde dosya sistemleri gerçekte var olmayan dosyaları barındırır. Bu dosyalara ihtiyaç olduğunda, çekirdek dosyaları hemen o anda üretir. Dosya tarihlerine bakılırsa bu kolaylıkla görülebilir. Dosya tarhileri ya sistemin açıldığı zamana ya da komutun girildiği zamana aittir.

Bu dosyalar açılış sırasında aşağıdaki gibi `/etc/rcS` içinden bağlanırlar.

[11] Pseudo

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

Bazı dağıtımlar bağlama işini `/etc/inittab` içinde veya başka yerlerden yaparlar. Çok da önemli değildir. Daha basit ve pratik olduğu bizler rcS içinden bağlamaktayız.

### 7.2.8 /tmp

Geçici dosyaların yazıldığı yerdir. Güç kesilince içindeki bilgiler kaybolur. `/tmp` dizini aşağıdaki gibi `/etc/rcS` içinden bağlanabilir.

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

### 7.2.9 /var

Aynı `/tmp` dizini gibidir. Geçici dosyaların yazıldığı yerdir. Güç kesilince içindeki bilgiler kaybolur. `/var` dizini aşağıdaki gibi `/etc/rcS` içinden bağlanabilir.

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

Pek çok gömülü sistemde `/var` dizini doğrudan `/tmp` dizinine sembolik olarak bağlanmıştır. Yani ayrı bir mount yapılmamıştır.

### 7.2.10 /dev/pts

`/dev` bağlandıktan sonra `/dev` içinde `pts` isimli ayrı bir bağlama daha yapılır. `devpts` 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. Bu dosya sistemi yine `/etc/rcS` içinde, aşağıdaki gibi bağlanır.

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

mount komutlarındaki hiç bir seçenekten ayrıca bahsedilmemiştir. Geniş bilgi için mount komutunun man sayfasına bakılabilir.

### 7.2.11 /dev/shm

Bu dizin de `/tmp` gibi geçici bir dizindir. Aslında yapı olarak `/tmp, /var` ve `/dev/shm` dizinleri aynı işi yaparlar. Geçici bilgileri saklarlar. Pek çok gömülü sistemde bütün geçici dizinler `/tmp`'ye sembolik olarak bağlanmıştır.

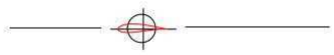
Fakat yine de bu dizinler kurulur. Bazı programlar illa ki bu dizinleri isterler ayrıca her dizinin mount seçenekleri birbirinden farklıdır. Örneğin `/var` dizini için `nodev` seçeneği varken `/tmp` için bu seçenek mevcut değildir.

## 7.3 RootFS

Buraya gelindiğinde kök dosya sistemimiz `RootFS` altında kurulmuş ve açılış içini hazırdır. Çekirdek açıldıktan sonra kök dosya sistemini ve `devtmpfs` dosya sistemini bağlar. Hemen peşinden `/sbin/init` programını çalıştırır. `/sbin/init` programı da `/etc/inittab` dosyasını kullarak açılışı yönetir.

Bu bölüme kadar Linux işletim sisteminin 3 ayağını tamamlamış olduk, U-Boot, Kernel ve RootFS. Son ayak ise açılış betikleridir.

Son bölümde açılış betikleri kurulacak ve sistem login seviyesine kadar açılacaktır.



# Bölüm 8

# Açılış Betikleri

Çekirdek yüklendikten ve yürütmesi başladıktan sonra kök dosya sistemini mount eder. Hemen peşinden kök dosya sisteminde `/sbin` dizini altında `init` isimli bir programı çalıştırmayı dener.

`init` programının içeriğinin ne olduğu hiç önemli değildir. Çalışabilir olması yeterlidir.

Kullanıcı kendisinin yazmış olduğu bir uygulama programına `init` adını veririse, çekirdek bu programı doğrudan çalıştıracaktır.

Ya da daha genel olarak çekirdek parametresi olarak, bootargs değişkenine `init=/falan/hede` gibi özel bir dizin veya program ismi yazılabilir. Bu durumda çekirdek `init=` ifadesinin karşısında bulunan `/falan/hede` programını yürütecektir.

Özetle kök dosya sistemi bağlandıktan sonra, ilk çalıştırılacak program `/sbin/init` programıdır. Eğer bu program bulunamazsa sıra ile `/etc/init, /bin/init` ve `/bin/sh` programları yürütülmeye çalışılır. Bunlar da yoksa çekirdek panikler ve sistemi tıkar.

Linux işletim sisteminde açılışı yöneten `init`[1] [2] isimli bir program vardır. Çekirdek, bu programı ilk proses olarak başlatır. Yani bu prosesin `PID` numarası[3] 1'dir[4].

[1] Initialization kelimesini kısaltılmışıdır.

[2] $ man init ; man 8 init

[3] $ pstree -p

[4] Masaüstü sistemlerde `init` yerine `systemd` kullanılır. `systemd` programı POSIX'e ait değildir. Gömülü sistemler için `systemd` programı son derece büyük ve gereksizdir.

`/sbin/init` programı çalışır çalışmaz, `/etc/inittab` dosyasını okur. `/etc/inittab` dosyası açılışı yönlendirir. Örneğimizde kullandığımız dosya aşağıda verilmiştir.

```
::sysinit:/etc/rcS

::respawn:/sbin/getty -L ttyS0 115200 vt100

::shutdown:/bin/sync
::shutdown:/bin/umount -a -r
```

`/etc/inittab` dosyası, birbirlerinden `":"` ile ayrılmış 4 kolondan oluşur. Bu kolonlarda aşağıdaki bilgiler bulunur.

```
id:runlevels:action:process
```

Örnek sistemimizde `busybox`'ın `init` programı kullanılacaktır. Bu program standard `inittab` dosyasından biraz daha farklı ve daha basittir.

Örnekte görüleceği gibi ilk 2 kolon boştur. Yani `id` ve `runlevel` kolonu boştur. Sebebi çok basittir. Gömülü sistemlerde açılış çok basittir, `id` ve `run level` kavramına hiç gerek yoktur.

Sonraki kolonlarda ise `sysinit, respawn` ve `shutdown` kelimeleri mevcuttur. Bu kelimeler 4. kolonda çalışacak programın davranışını tespit eder.

`init` programı çalışır çalışmaz 3. kolonda `sysinit` olan satırı arar ve sağındaki programı gözü kapalı çalıştırır. Bu durumda ilgili satırın 4. kolonunda bulunan `/etc/rcS` programı çalışacaktır. `/etc/rcS`'nin çalışması tamamen bittikten sonra diğer satırlar sıra ile işletilir.

`::respawn:/sbin/getty -L ttyS0 115200 vt100` satırı ile seri kanal üzerinden terminal erişimi sağlanır. Terminalden login olduktan sonra exit ile çıkılırsa, `getty` programı da son bulur. Programın tekrar başlatılması gerekir. Bu durumda 3. kolonda `respawn` yazılırsa, sonlanan bir program `init` tarafından otomatik olarak tekrar başlatılır.

Eğer `/sbin/getty` satırı mevcut değilse, seri kablo ile sisteme erişim mümkün olmaz. Müşteriye verilen sistemlerde bu satır iptal edilmelidir.

3. kolonda `shutdown` bulunan satırlar ise, `reboot, halt, poweroff` gibi sistemi kapatan veya yeniden başlatan komutlar girildiğinde otomatik olarak

Standard dağıtımlarda `init` ve `systemd` aynı programdır.

çalışır. 3. kolonu aynı olan birden fazla satır varsa, örnekte 2 adet `shutdown` satırı mevcuttur, her bir satır sıra ile işletilir. Bir satırdaki komutun yürütmesi bitince diğer satıra geçilir. Build root programı bu özelliği bolca kullanmaktadır.

İlk `shutdown` satırında, `/bin/sync` komutu ile blok cihazların ara bellekleri boşaltılır. Sonraki `/bin/umount` komutu ile de bağlı bütün dosyalar otomatik olarak `umount` edilir. En az 1 dosya mount durumundadır ve asla `umount` edilemez. En azından `init`'in üzerinde çalıştığı dosya halen meşgüldür, ve `unmount` edildiğinde, "file system busy" hatası alınır. Bu durumda kullanılan `-r` parameteresi, halen kullanılmakta olan dosya sistemlerinin `read/only` olarak tekrar mount edilmesini sağlar. Böylece güç kesildiğinde bord temiz bir biçimde kapanmış olacaktır.

Aslında bu son yazdıklarımız şehir efsanesi gibidir. Standard dağıtımların, basınç altında sıkıştırılarak gömülü sistem üretme çabalarının bir ürünüdür.

Gömülü sistemlerin pek azı, terminalden `reboot, poweroff` veya `halt` komutları girilerek kapatılır. Genelde fişi çekilir veya pili çıkarılır. İşte bu yüzdendir ki ani kapanmalara karşı korunmak için kök dosya sistemi her zaman read/only bağlanmalıdır. Ani kapanabilen gömülü sistemlerde `shutdown` satırlarının bulunmasına hiç gerek yoktur.

Çok popüler olan bordlarla gelen gömülü Linux sistemlerinde kök dosya sistemleri read/write bağlanmıştır. Hiç biri ani kapanmalara karşı korumalı değildir. En basit koruma yöntemi olarak log tabalı dosya sistemleri seçilmiştir. Fakat bu sistemler de ani kapanmalarda hiç bir zaman %100 koruma sağlamazalar.

Ayırca müşteriye verilecek gömülü sistemde `::respawn:/sbin/getty` satırının bulunması gerek yoktur. Müşterinin seri kanaldan sisteme bağlanması istenen bir özellik değildir.

Bu kadar laftan sonra geriye olmazsa olmaz bir tek `sysinit` satırı kalmıştır. `init` programı ilk olarak bu satırın son kolonunda buluna programı, yani `/etc/rcS` programını çalıştırır.

`/etc/rcS` programı bir betiktir ve açılış için gerekli olan programları başlatır.

## 8.1 /etc/rcS

Örnek bir açılış betiği aşağıda verilmiştir. Soldaki numaralar açıklama içindir, betiğe dahil değildir.

```
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
```

01. satır, bu dosyanın hangi program ile yürütüleceğini belirtir. Bu dosya, yani `/etc/rcS` dosyası `/bin/sh` komut ile yürütülecektir. `+x` seçeneği `debug off` demektir. Bu seçenek `-x` yapılırsa, `debug on` olur ve her bir satır işletilmeden önce ekrana yazılır. Özellikle ilk açılışlarda hata aramak için çok faydalıdır.

02. satırda, komutların nerede aranacağı belirtilir. Bu sayede, betik içinde `/bin/mount` yerine kısaca `mount` yazılabilir.

03. ve 16. satırlar arasında dosya sistemleri mount edilir ve gerekli dizinler kurulur. Kök dosya sistemi kuruluşunda bu satırlar genişçe açıklanmıştı.

17. satırda, inernet domain isimlerinin çözüleceği makine adresi verilir. Bizler google için DNS adresi verdik. Uygun DNS adresleri burada verilebilir.

17. satır çok ilginç bir özelliğe sahiptir. Bizim kök dosya sistemimiz read/only'dir. Bu satır `nameserver 8.8.8.8` ifadesini `/etc/resolv.conf` dosyasına yazar. Tabii ki önce dosyayı yaratır. Ama kök dosya sistemi salt okunur olduğu için dosya yaratılamaz ve hata oluşur. Hatayı çözmek için bu dosya yazılabilir olan `/tmp/resolv.conf` dosyasına sembolik olarak yönlendirilmiştir. Böylece salt/okunur bir dosya sistemi içine, sembolik yönlendirmelerle yazım yapılabilmektedir.

18. satırda çekirdeğe `mdev`[5] isimli bir programın hotplug için kullanılacağı söylenir. Bu tür programlara, kernel tarafında "helper program"[6] denir.

Borda herhangi bir cihaz takıldığı zaman, çekirdek "user space" tarafında bulunan `mdev` programını çalıştırır. Çalıştırıken de cihaz hakkında, "isim, adres, bus, vb" pek çok bilgiyi `mdev` programına geçirir. `mdev` programı da kernel'dan gelen bu bilgileri kendi config dosyası ile karşılaştırır ve gelen bilgiler ile uyuşan bir pattern varsa, bu pattern ile verilen modülü yükler veya başka herhangi bir betiği veya komutu çalıştırır. Özetle, `echo /sbin/mdev > /proc/sys/kernel/hotplug` komutu, çekirdeğe user space tarafında çalışacak programın ismini aktarır. Çekirdek de bir event meydana geldi mi, bu programı yürütür.

19. satırda sisteme ait hostname bilgisi tanıtılır. Domain name A.B.C ise, hostname olarak A varsayılır. Diğer bir deyişle domain name ile verilen ismin, ilk noktaya kadar olan kısmına hostname denir. Bundan dolayı hostname

[5] Bakınız https://git.busybox.net/busybox/plain/docs/mdev.txt
[6] Bakınız https://www.kernel.org/doc/pending/hotplug.txt

olarak domain name verilirse, ilk noktadan sonraki kısımlar otomatik olarak gözardı edilir.

20. satırda sistem kayıtçısı başlatılır. Her türlü program C tarafında `syslog` ile veya betik tarafında `logger`[7] komutu ile bu kayıtçıya bilgi yazabilir. Kayıtçı, bu bilgileri `/var/log/syslog` veya `/var/log/messages` gibi dosyalarda tutar.

21. satırda çekirdek mesajlarını `syslog`'a aktaran program başlatılır. Çekirdek "user space" tarafında bulunan sistem kayıtçısından habersizdir. `klogd` programı çekirdekten gelen mesajları `syslogd` programına yönlendirir. Bu program başlatılmazsa, çekirdekten gelen mesajlar kaybolur.

22 ve 23. satırlarda `localhost` adresi, yani "127.0.0.1" adresi ve rota bilgisi tanıtılır.

25 ile 26. satırlarda statik IP ve gerekli rota bilgisi vardır. Bu satırlar açıklama şeklinde bırakılmıştır. Statik IP isteyen kullanıcı bu satırları açıp kendi IP bilgilerini yazabiir.

28. satırda dinamik IP bilgisi elde edilir. Bu bilgi elde edildikten sonra `udhcpc` programı tarafından otomatik olarak `/etc/udhcpc.script` betiği çağrılır. Okuyucunun betiğin içini incelemesi tavsiye edilir. Aynı anda ya static ya da dinamik IP kullanılmalıdır. İkisini birden kullanılması anlamsızdır.

29. satırda telnet sunucusu başlatılır. Böylece uzaktan borda erişim mümkün olur.

`/etc/inittab` dosyasında `::respawn:/sbin/getty -L ttyS0 115200 vt100` şeklinde bir tanım vardı. Bu tanım seri kanaldan login olmayı sağlamaktadır. Aynı zamanda u-boot seviyesinde bootargs ifadesi olarak, yani çekirdeğe geçirilecek parametre olarak, `setenv bootargs console=ttyS0,115200` tanımı yapılmıştı. Bu tanım çekirdek mesajlarının aynı zamanda `ttyS0`'a yani seri kanala akmasını söyler. Diğer bir deyişle konsol olarak[8] seri kanal tanıtılmıştır. Bu tanımlamalar ile seri kanala hem konsol hem de terminal görevi yüklenmiştir.

Seri kanaldan login yapıldıktan sonra, çalışma sırasında kernel mesajları ekrana düşmeye başlar. Örneğin editör içindeyken birden kernel mesajı gelir ve bütün ekran görüntüsünü karıştırır. Bu durumda telnet ile farklı bir yerden bağlanmak en temiz çözümdür.

[7] $ man syslog logger

[8] Konsol, çekirdek mesajlarının aktığı yerdir. "Line discipline" özelliğine sahip değildir. Terminal ise login yapılan yerdir. Normalde konsoldan login yapılmaz.

## 8.2 Niçin bütün bilgiler sabittir?

Bizler, hem eğitimlerimizde hem de yazdığımız belgelerde "atomik indirgeme" yönetimini uygulamaktayız. Bu bizim uydurduğumuz bir yöntemdir. Tekniği de son derece basittir. Bu teknikte anlatılacak bir konu indirgenebilecek en alt seviyeye kadar indirgenir ve öyle anlatılır. Araya başka hiç bir katman eklenmez. Böylece konunun sistematiği veya felsefesi en yalın hali ile aktarılmaya çalışılır. Araya giren her katman karmaşıklığı artırır. Bundan dolayı indirgenebilecek en alt seviyede anlatılır konular.

`u-boot` ile açılış örneğini verecek olursak, girilen ifadelerde pek çok tekrar eden bilgi olmasına rağmen, bunlar her seferinde tek tek yazılmıştır. Esasında tekrar eden bilgiler değişkenlere atanır ve u-boot içinde bu değişkenlere $ ile atıf yapılır.

Okuyucu açılışta `print` komutu girerse, pek çok değişken tanımı görür. Bu çıkışı çalışarak kendisini kolaylıkla geliştirebilir.

Benzer şekilde `/etc/rcS` dosyasının her bir komutu aslında `/etc/init.d/` altında `start/stop` betikleri ile başlatılıp durdurulabilir ki build root sistemi ve pek çok dağıtım bu şekilde çalışır.

`/etc/rcS` örneğinde IP bilgileri tabii ki doğrudan buraya yazılmayabilir. Dışarıdaki bir dosyaya yazılıp doğrudan `/etc/rcS` içinden kullanılabilir. Hatta açılışta dinamik veya statik IP alıp almayacağı sorgulanabilir.

Bütün bunları anlatımın içine eklemek ortalığı karıştırmaktan başka bir fayda sağlamayacaktır. Okuyucu öncelikle bütün kontorlü kendisinde olan bir gömülü sistem kurmalı ve sonra da uygun bir biçimde yayılarak sistemi geliştirmelidir.

Bu kadar lafı kuruluş işlemini bitidiğimiz için ettik. Son konu olan açılış betikleri ile de Linux kuruluşunun son ayağı da tamamlanmıştır. Artık `RootFS` dizini altında kurulan kök dosya sistemi SD kartın 1. bölümüne kopyalanarak test başlatılabilir.

# Bölüm 9

# SD Kartın 1. Bölümünden Açılış

Şimdiye kadar yapılan çalışmalarda `/opt/gomsis/RootFS` altında busybox destekli kök dosya sistemi kurulmuştur. Bu kuruluş host makinede yapılmıştır. Şimdi SD kart host makineye takılmalı ve `RootFS` dizininin içeriği 1. bölüme kopyalanmalıdır. Bu işlemler aşağıdaki gibi yapılabilir[1].

```
# Boş bir dizin yarat, varsa bir şey yapma.

$ mkdir /mnt/root

# SD kartı host makineye tak.
# Daha önce 1. bölüme ext2 dosya sistemi kurulmuştu.
# Şimdi 1. bölümü bağla.
#
# mmcblk0p1 ismi okuyucunun makinesinde farklı olabilir.
# $ dmesg|tail ile kontrol et.

$ mount /dev/mmcblk0p1 /mnt/root

# Kök dosya sistemini SD'ye aktar.

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

# Bütün dosyaların sahipliklerini root:root yap.
# -R: recursive
```

[1]Kullanıcının automounter kullanmaması ve her işi el yordamı ile yapması tavsiye edilir. Ubuntu'da bunun için `$ dconf-editor` komutu girilir. Gelen program içinden find seçeneği ile automount aranır ve seçili kutular kapatılır.

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

# Bağı kopar.

$ cd    # cihaz dışına çıkmayı garantile.
$ umount /dev/mmcblk0p1

# Kartı çıkar ve borda tak.
```

Kart çıkarılıp borda takılır ve borda güç verilir. u-boot seviyesinde açılış durdurulur ve aşağıdaki komutlar girilerek login'e kadar gelinir.

```
> ext2load mmc 0 0x46000000 boot/zImage

> ext2load mmc 0 0x49000000 boot/dtb

> setenv bootargs console=ttyS0,115200 earlyprintk \
         root=/dev/mmcblk0p1 ro rootwait

> bootz 0x46000000 - 0x49000000
```

Her adım düzgün yapılmışsa aşağıdaki gibi bir ekran çıktısı elde edilmelidir. root ile girilir, şifre yoktur.

```
UCanLinux > ext2load mmc 0 0x46000000 boot/zImage
4114112 bytes read in 371 ms (10.6 MiB/s)

UCanLinux > ext2load mmc 0 0x49000000 boot/dtb
13655 bytes read in 101 ms (131.8 KiB/s)

UCanLinux > setenv bootargs console=ttyS0,115200 earlyprintk \
            root=/dev/mmcblk0p1 ro rootwait

UCanLinux > bootz 0x46000000 - 0x49000000

## Flattened Device Tree blob at 49000000
   Booting using the fdt blob at 0x49000000
   Loading Device Tree to 49ff9000, end 49fff556 ... OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 4.9.0-rc1+ (nazim@nkoc)
               (gcc version 4.9.4 20151028 (prerelease) (Linaro GCC 4.9-2016.02) ) #1
               SMP Thu May 18 18:51:24 +03 2017
[    0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c5387d
...
...
udhcpc: sending discover
udhcpc: sending select for 192.168.1.29
udhcpc: lease of 192.168.1.29 obtained, lease time 268435455
deleting routers
route: SIOCDELRT: No such process
adding dns 8.8.8.8
adding dns 0.0.0.0


 _   _  ____            _     _
| | | |/ ___|__ _ _ __ | |   (_)_ __  _   ___  __
| | | | |   / _` | '_ \| |   | | '_ \| | | \ \/ /
| |_| | |__| (_| | | | | |___| | | | | |_| |>  <
 \___/ \____\__,_|_| |_|_____|_|_| |_|\__,_/_/\_\


Welcome to UCanLinux
http://UCanLinux.Com

UCanLinux login: root
root@UCanLinux:~ #
```

Login geldikten sonra SD kart üzerinde oturan kök dosya sistemi üzerinde güncelleme yapılamaz. Çünkü kök dosya sistemi salt-okunur bağlanmıştır. root ile login olduktan sonra kök dosya sistemi, aşağıdaki gibi yaz-oku haline getirilip, canlı canlı SD kart üzerinde güncelleme yapılabilir.

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

# Bu arada güncellemeleri yap.
# Sonra tekrar aşağıdaki gibi r/o moda geçir.

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

Daha önce ext2load, bootargs ve bootz ifadelerinden bahsedildiği için ayrıca üzerinde durulmayacaktır. Yeni gelen bazı parameterler aşağıda açıklanmıştır.

`bootargs` ifadesinin sağındaki bütün ifadelerin çekirdeğe aktarıldığını, u-boot sistemi ile hiç bir ilgisinin olmadığını söylemiştik. Bu yeni kullanımda 3 yeni parametre gelmiştir.

Çekirdek açılış mesajlarını aynı C'deki gibi print fonksiyonu ile yazar. Tabbi ki bu fonksiyonun arkasında libc olmadığı için çok basittir. Çekirdekteki print fonksiyonunun adı printk'dır[2]. Çekirdek açılır açılmaz `printk` ile mesaj yazmaya başlar. Fakat henüz konsol mevcut değildir. Konsol yoksa bu mesajlar nereye yazılacaktır, kaybolacak mıdır?

`earlyprintk` parametresi, konsol henüz yokken mesajların seri veya VGA konsolun veya konsol ne ise oranın ara belleğine yazılmasını sağlar.

Konsol daha sonra aktif olduğunda ara bellekte birikmiş olan bu bilgiler konsola aktarılır. Böylece konsol aktif olmadan önceki printk mesajları kaybolmaz. Müşteriye verirken bu parametre iptal edilmelidir. Geliştirme aşamasında parametre kullanılmalıdır ki mesaj kaybı olmasın.

Çekirdek, açılışını bitirdikten sonra kök dosya sistemini mount eder. Kök dosya sistemi envai çeşit yerde olabilir. `root=/dev/mmcblk0p1` tanımı, çekirdeğe kök dosya sisteminin nerede olduğunu söyler. Çekirdek de bu tanımı kullanarak kendi içinde, mealen aşağıdaki gibi bir mount[3] komutunu çalıştırır.

```
$ mount o -ro /dev/root /dev/mmcblk0p1
```

[2]Print kernel

[3] $ man 2 mount

`ro` parametresi kök dosya sisteminin salt-okunur bağlanmasını söyler. Kök dosya için varsayılan mount seçeneği, SD/MMC/eMMC kartları ve her türlü disk için her zaman read/write biçimindedir. Kök dosya sistemi read/write bağlanmamalıdır.

Saç baş yolduran ve bu işlere ilk başladığımızda bizi hayattan bezdiren bir hatanın çözümü ise `rootwait` seçeneğidir.

Çekirdek açıldıktan sonra kök dosya sistemini bağlamaya çalışılır. `root=/dev/mmcblk0p1` tanımını kullanan çekirdek, hemen bu cihazı mount etmeye çalışır. Amma velakin SD kartın çekirdek tarafından tespit edilesi on milisaniye'nin birkaç katı sürebilir. Diğer bir deyişle SD kart henüz tespit edilmeden mount işlemi başlayabilir. Çekirdekte pek çok işin paralel çalıştığını hatırlatalım. Bu durumda yaptığınız her iş düzgün olmasına rağmen kök dosya sistemi bağlanamaz ve çekirdek panikler.

`rootwait` seçeneği, kök dosya sisteminin oturduğu cihaz kernel tarafından tespit edilene kadar açılışı durdurur. Çekirdek, kökün oturduğu cihazı tespit ettikten sonra, kök dosya sistemini bağlamak için mount komutunu işletir.

Açılış mesajları incelirse bekleme mesajı görülebilir.

## 9.1 Otomatik Açılış

Sistem tekrar açıldığında u-boot da girdiğimiz bilgiler kaybolur. Bordun dışarıdan müdahale edilmeden açılmasına otomatik açılış[4] denir. U-boot'un otomatik açılması için `bootcmd` isimli bir değişken tanımlanmıştır. Eğer u-boot çevre değişkenleri içinde `bootcmd` değişkeni tanımlı ise, içindeki bilgiler otomatik olarak çalıştırırlır. Böylece bord kendiliğinden açılır.

Otomatik açılış için bir örnek aşağıda verilmiştir. Bord açılırken u-boot seviyesinde açılış bir tuşa basılarak durdurulur ve aşağıdaki komutlar girilir[5].

[4] Autoboot

[5] Okuyucunun "\" işaretlerini girmesine gerek yoktur. Tek satır halinde yan yana yazım yapabilir. Satır çok uzun olduğu ve A4 sayfasına sığmadığı için bu şekilde parçalanmıştır.

```
> setenv bootargs console=ttyS0,115200 earlyprintk root=/dev/mmcblk0p1 ro rootwait

> setenv bootcmd "ext2load mmc 0 0x46000000 boot/zImage; \
                  ext2load mmc 0 0x49000000 boot/dtb; \
                  bootz 0x46000000 - 0x49000000"

> saveenv

> reset
```

`bootargs` tanımında bir değişiklik yoktur. Eskisi gibi, hiç değiştirilmeden kullanılmıştır. `bootcmd` değişkenine ise, açılışta çalışmasını istediğimiz komutlar girilmiştir. U-boot açılışta bu değişkenin içindekileri otomatik olarak çalıştırır. Böylece otomatik açılış gerçeklenir.

Bord kapatıldığında `bootargs` ve `bootcmd` değişkenleri kaybolur. Bütün değişkenlerin kalıcı olarak saklanması için `saveenv` komutu girilir. Bu komut, bütün değişkenleri Tablo 4.1'de verilen, environment ile gösterilen ve 544. KBayttan başlayan, özel olarak ayrılmış bir bölüme kaydeder.

Değişkenler boot sektörde 544. KBayttan itibaren raw biçimde saklanırlar. U-boot açıldıktan sonra 544. KBayttan itibaren değişkenleri okur ve kendi içine alır. Okuma bittikten sonra hemen `bootcmd` değişkenini arar. Bulursa ki `saveenv` ile kaydetmiştik, hemen içindeki komutları yürütür ve bordu dışarıdan müdahalesiz açar.

U-boot sisteminde arka arkaya gelen komtular ";" ile veya "&&" ile tek satırda yazılabilirler. `bootcmd` komutunda 3 adet u-boot komutu birbirlerinden ";" ile ayrılarak yazılmıştır. "&&" ile yazıldığında ise, komutlar "mantıksal ve"[6] gibi davranır ve ilk hatalı dönüşte yürütme durur.

U-boot betiklerinde ana değişkenler her zaman `bootargs` ve `bootcmd`'dir. Anlatımı kolaylaştırmak için u-boot değişkenleri en basit halleri ile kullanılmıştır.

U-boot'da `print` komutu girildikten sonra gelen ifadeler incelenebilir. İncelemenin `bootcmd`'den geriye doğru yapılması tavsiye edilir. Yani önce `bootcmd` satırı sonra da `bootcmd`'nin içinde bulunan değişkenler hiyerarşik sırada incelenebilir. Bu inceleme sonucudan görülecektir ki aslında SD karttan bordu açmak için sadece birkaç tanım yeterlidir. Ki bu zaten yukarıdaki örnek ile de gösterilmiştir.

[6]Logical and

# Bölüm 10

# Initramfs Destekli Açılış

Bir önceki sistemde kök dosya sistemi SD kartta idi. Linux çekirdeği kök dosya sistemini pek çok yerden mount edebilir. Hatta daha güzel bir biçimde, hiç mount etmesine gerek kalmadan doğrudan kök dosya sistemini kendi kaynak kodunun içine alıp, açılışta doğrudan RAM bellekte kök dosya sistemini kurup, mount edebilir.

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 esas 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.

Bu yöntemde kök dosya sistemi doğrudan çekirdek koduna gömülecektir. Sistem açıldıktan sonra artık SD karta gerek olmayacaktır. SD kart yerinden çekilebilir ve Linux sorunsuz çalışmaya devam edecektir.

Bir önceki örnek sistemde kök dosya sistemi SD kartta olduğu için, Linux çalışırken SD kart yerinden çekilemez.

Çekirdeğe gömülü kök dosya sistemi kurmak son derece basittir. Önceki sistemde bulunan boot dizini tamamen silinecektir. Yani kernel ve dtb'nin bulunduğu dizin silinecektir. Sonra kernel yeniden derlenecek ve `/boot` dizini silinmiş örnek kök dosya sistemi çekirdek koduna eklenecektir.

Daha sonra içine RootFS gömülmüş çekirdek ve dtb dosyası SD kartın 2. bölümüne atılacak ve sadece bu 2 dosya ile, yani zImage ve dtb dosyası ile

bord açılacaktır.

Bu işlemleri topluca aşağıda verilmiş olup, son derece basit olduğu için üzerinde ayırca durulmayacaktır. Daha önce kurulan RootFS'in bozulmaması için, RootFS.initramfs adı ile taze bir kopya üzerinde çalışma yapılacaktır.

```
# Daha önce kurulmuş bir örnek sistem varsa sil.
#
$ cd /opt/gomsis
$ rm -fr RootFS.initramfs

# RootFS'in kopyasını al ki ilk kurulan RootFS bozulmasın.
#
$ cp -a RootFS RootFS.initramfs

# boot dizini ile işimiz yoktur.
# Çünkü RootFS.initramfs dizini zaten zImage içine
# alınacaktır. zImage'nin kendi kendini içine alması
# anlamlı değildir.
#
$ rm -fr RootFS.initramfs/boot

# Yeni rcS dosyasını eskisini üzerine yaz.
# Yeni dosyada sadece tek satır farklıdır.
#
$ cp bin/rcS RootFS.initramfs/etc/

# Initramfs sistemlerinde ilk program /sbin/init değil,
# /init'tir.
# Bundan dolayı /init'i /sbin/init'e sembolik olarak bağla ki
# standard init ile açılış yapabilelim.
#
$ cd RootFS.initramfs
$ ln -s /sbin/init
$ cd ..

# Gereksiz dosyaları sil.
#
$ rm -f RootFS.initramfs/lib/modules/*/build
$ rm -f RootFS.initramfs/lib/modules/*/source

# Aşağıdaki seçimleri yaparak çekirdeği yeniden derle.
#
# General setup  --->
#  [*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
#      (/opt/gomsis/RootFS.initramfs) Initramfs source file(s)
#      (1000)  User ID to map to 0 (user root)
#      (1000)  Group ID to map to 0 (group root)
```

```
#
# Okuyucu 1000 yerine kendi userId:group değerini yazmalıdır.
# Bu değerler $ id komutu ile bulunabilir.
# Derlem bittikten sonra, zImage'nin boyu kontrol edilmelidir.
# İçine kök dosya sistemi eklendiği için zImage şişecektir.
# Şişme miktarı kök dosya sisteminin mevcut boyutundan her zaman
# daha azdır. Çünkü, çekirdek kodu içinde, kök dosya sistemi
# sıkıştırılarak saklanır.
#
$ cd /opt/gomsis/linux
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage -j$(nproc)

# İmajın boyu 2 kat arttı.
#
$ ls -l arch/arm/boot/zImage
-rwxrwxr-x 1 nazim nazim 10496376 May 18 21:06 arch/arm/boot/zImage
```

Artık elimizde şişmiş bir zImage ve eskiden kurulmuş dtb mevcuttur. Sadece bu ikisini SD kartın 2. bölümünün `/boot` dizinine atarak bordu açabiliriz. SD kartın 1. bölümündeki kuruluşu bozmamak için bu yeni sistem 2. bölüme atılacaktır. Yeni initramfs sistemi SD'nin 2. bölümüne aşağıdaki gibi atılabilir.

```
# Bölümlendirme yaparaken SD kartın 1. ve 2. bölümüne dosya sistemi
# zaten kurulmuştu.
# Burada 2. bölüm mount edilir ve zImage ile dtb, /boot altına atılır.
#
$ mount /dev/mmcblk0p2  /mnt/root

# SD kartta /boot dizinini kur.
#
$ mkdir /mnt/root/boot

# Çekirdek ve dtb'yi kopyala.
#
$ cd /opt/gomsis
$ cp linux/arch/arm/boot/zImage /mnt/root/boot/
$ cp linux/arch/arm/boot/dts/sun8i-h3-orangepi-one.dtb /mnt/root/boot/dtb

# Boyları ve içerikleri incele.
#
$ df /mnt/root
$ ls -lR /mnt/root

# Sal gitsin.
```

```
#
$ umount /mnt/root

# Kartı çıkar ve borda tak.
```

Kuruluş bitmiştir. Kuruluşta dikkat edilmesi gereken birkaç nokta var. Bunlar sıra ile aşağıda verilmiştir.

Çekirdek initramfs sistemlerinde hiç bir zaman `/dev` dizinini bağlamaz. Çünkü initramfs'i gerçek kök dosya sistemi kabul etmez. Ama kendi içinde `devtmpfs` dosya sistemini de peşinen kurmuştur. Initramfs sistemlerinin açılışında, çekirdek içinde kurulan `devtmpfs` dosya sistemi, açılışta açıkça mount edilmelidir. Bunun için önceki `/etc/rcS` sistemine, aşağıdaki tek satırı eklemek yeterlidir.

```
mount -t devtmpfs devtmpfs  /dev
```

Bu sayede çekirdek içindeki device dosya sistemi, user space tarafında kullanılabilir hale gelecektir. `/etc/rcS` içinde `mdev -s` komutu ile de `/dev` dosya sistemi inşa edilebilir. Ama bu yöntem çok yavaştır, tavsiye edilmez.

Bir diğer önemli özellik ise kök dosya sisteminde `/init` adlı bir dosyanın gerekliliğidir. SD kart ile olan açılışta çekirdek gözü kapalı bir biçimde `/sbin/init` dosyasını çalıştırmaktadır. Bu yöntemde ise `/init` dosyasını çalıştırır. İki dosyayı birbirine sembolik olarak bağlarsak sorun kalmaz. Sistem SD karttaki gibi açılacaktır.

Çekirdeği derlerken `make menuconfig` sırasında örnek kök dosya sistemi yani RootFS.initrams dizini altındaki dosyalar,
`(/opt/gomsis/RootFS.initramfs) Initramfs source file(s)`
seçeneği sayesinde çekirdeğe eklenmektedir.

RootFS.initramfs içindeki dosyaların owner ve group değerleri `root:root` olmalıdır. ama her zaman root şifresine sahip olmayabiliriz. Özellikle üniversitelerin Linux makinelerinde root şifresi öğrencilere verilmez. Bu durumda `RootFS.initramfs` dizini altındaki dosyaların owner ve group değerleri nasıl root:root yapılacaktır?

Bunun için çok güzel bir çözüm mevcuttur. `make menuconfig` sırasında aşağıdaki gibi girişler yapılır.

```
(1000)  User ID to map to 0 (user root)
(1000)  Group ID to map to 0 (group root)
```

Bu girişler, `RootFS.initramfs` altında bulunan ve userId ve groupId değerleri `1000` olan bütün dosyaların userId ve groupId değerlerini, root yetkisine gerek olmadan `root:root` veya sayısal olarak `0:0` yapar.

İşte bu yüzden `RootFS.initramfs` dizini altında bulunan bütün dosyaların değerleri hep aynı kullanıcıya ait olmalıdır.

Eğer root şiresi biliniyorsa, `$ chown -R root:root RootFS.initramfs/*` komutu ile bütün sahiplikler ve gruplar `root:root` yapılabilir. Bu durumda yukarında parantez içinde yazılan `(1000)` değerleri yerine doğrudan 0 yazılır. Ama bu tavsiye edilmez.

2. sistemin kuruluşu tamamlandı. Aşağıdaki gibi u-boot ile test yapılabilir.

## 10.1 Initramfs Sisteminin Testi

SD kart borda takılır ve bir tuşa basarak açılış durdurulur. Eğer açılış durdurulmazsa, `bootdelay` değişkenindeki kadar bekleme yapılır ve ilk örnek sistemimiz açılır. U-boot seviyesinde aşağıdaki komutlar girilerek bord açılabilir.

```
> setenv bootargs console=ttyS0,115200 earlyprintk
> ext2load mmc 0:2 0x46000000 boot/zImage
> ext2load mmc 0:2 0x49000000 boot/dtb
> bootz 0x46000000 - 0x49000000
```

Önceki açılışlardan hiç bir farkı yoktur. Sadece bootargs içindeki kök dosya sistemi ile ilgili tanımdan kırpılmıştır. Çünkü kök dosya sistemi artık çekirdek içindedir ve ayrıca bir cihaz ismi belirtmeye gerek yoktur.

Sistem açıldıktan sonra SD kart yerinde çekilebilir. Çünkü artık bütün kök dosya sistemi RAM bellektedir.

Büyük kök dosya sistemleri için bu yöntem uygun değildir. `$ free` komutu ile kullanılan bellek miktarı görülebilir. İlk sisteme göre daha fazla bellek tüketilmiştir. Çünkü kök dosya sistemi hep bellekte oturmaktadır.

Ayrıca kök dosya sistemine yazılan bütün bilgiler sistem kapanınca kaybolacaktır.

Bu teknik için otomatik açılış yapılmayacaktır. Bunun yerine sonraki bölümde u-boot'a menü eklenecek ve açılışta istenilen sistem seçilebilecektir.

# Bölüm 11

# U-Boot Menüsü

U-boot açılış sisteminin çok basit bir menü desteği mevcuttur. Menü desteği sayesinde, kurulu Linux sistemlerinden herhangi biri açılışta seçilebilir. Geliştirme aşamasında bu özellik çok faydalı olacaktır. Fakat müşteriye verilecek sistemlerde böyle bir özelliğin bulunması anlamsızdır.

Geçmiş bölümlerde 2 farklı Linux sistemi kurulmuştu. İlk sistemde kök dosya sistemi SD kartın 1. bölümünde, ikinci sistemde ise 2. bölümünde çekirdek içinde oturmaktadır.

Bu iki farklı sistem aşağıdaki gibi bir menü altında birleştirilebilir. Kullanıcı isterse console tanımlarını ayrı bir değişken olarak ekleyebilir. Biz değişiklik olsun diye iptal ettik.

```
> setenv bootargs_1 "setenv bootargs root=/dev/mmcblk0p1 ro \
                     rootwait rootfstype=ext2 earlyprintk"

> setenv bootargs_2 "setenv bootargs earlyprintk"

> setenv bootcmd_1  "ext2load mmc 0 0x46000000 boot/zImage ; \
                     ext2load mmc 0 0x49000000 boot/dtb ; \
                     bootz 0x46000000 - 0x49000000"

> setenv bootcmd_2  "ext2load mmc 0:2 0x46000000 boot/zImage ; \
                     ext2load mmc 0:2 0x49000000 boot/dtb ; \
                     bootz 0x46000000 - 0x49000000"

> setenv bootmenu_0 RootFS on MMC=run bootargs_1 bootcmd_1
> setenv bootmenu_1 RootFS on ram=run bootargs_2 bootcmd_2
```

```
> setenv bootmenu_2 Reset=reset

> setenv bootcmd bootmenu 20

> saveenv

> reset
```

Menü sistemi desteği için, u-boot derlemesi sırasında `bootmenu` seçeneği açılmalıdır.

Açılış sırasında bir tuşa basılarak u-boot seviyesinde açılış durdurulur. Sonra yukarıda verilen satırlar girilir ve `saveenv` komutu ile güncellemeler kalıcı hale getirilir.

`bootargs_1` ve `bootargs_2` değişkenlerinde yeni bir ifade yoktur. Önceden kullandığımız şekli ile tamamen aynıdır. Sadece `bootargs` değişkeni yerine `bootargs_1` kullanılmıştır. Bu bizim kullandığımız keyfi bir isimdir. Her bir açılış için ayrı `bootargs` değişkeni olacaktır.

Benzer şekilde `bootmcd` komutu doğrudan kullanılmamış, sonlarına 1 ve 2 sayıları eklenmiştir. Daha sonra menüden seçim yapıldığında uygun `bootargs` ve `bootcmd` değişkenleri çağrılacaktır.

Buraya kadar, daha önce kullandığımız ifade veya komutlar değişkenlere aktarılmıştır. Daha sonra `bootmenu_1, bootmenu_2` ve `bootmenu_3` değişkenleri kullanılmıştır. Bu değişkenler u-boot sistemine ait olan menü yapım değişkenleridir[1]. Kullanım tekniği aşağıda verilmiştir.

```
> setenv bootmenu_NO TITLE=COMMAND
```

Her bir `bootmenu_NO` ifadesi, bir menü satırını gösterir. İlk menü satırı için, `NO` değeri 0'dan başlar.

`TITLE=COMMAND` ifadesinde, `=` işaretinin sol tarafı, yani `TITLE` ifadesi, açılışta ekrana çıkacak olan ifadedir. `=` işaretinin sağ tarafı, yani `COMMAND` ifadesiyse, seçim yapıldıktan sonra çalışacak komuttur.

Örnek olarak, "`bootmenu_0 RootFS on MMC=run bootargs_1 bootcmd_1`" ifadesinde, açılış sırasında ekrana "`RootFS on MMC`" başlığı çıkacaktır. Eğer kullanıcı bu satırı seçmişse, ilgili satırda `=`'in sağ tarafında bulunan "`run bootargs_1 bootcmd_1`" ifadesi çalışacaktır.

[1] http://code.metager.de/source/xref/denx/u-boot/doc/README.bootmenu

U-boot'da run komutu ile bir değişkenin içinde bulunan ifadeler çalıştırılabilir. "`run bootargs_1 bootcmd_1`" ifadesi ile de `bootargs_1` ve `bootcmd_1` değişkenleri çalıştırılacak ve içindeki ifadeler yürütülecektir.

Her iki açılışta da `bootargs` ve `bootmcd` içerikleri farklı olduğundan, ayrı ayrı değişkenlere atama yapılmış ve seçim sırasında ilgili değişkenler yürütülmüştür.

Açılışta "`setenv bootmenu_2 Reset=reset`" satırı seçilirse, `=`'in sağında bulunan `reset` komutu çalışacaktır. Bu komut u-boot'un kendi komutudur. Bundan dolayı önüne `run` ifadesi konulmamıştır.

"`setenv bootcmd bootmenu 20`" ifadesinde belirtilen süre kadar bekleme yapılır. Verilen sayı beklenecek saniye miktarıdır. Açılışta otomatik olarak menü gelir ve geri sayım başlar. Ok tuşları ile menü üzerinde gezilirse, geri sayım durur. Enter'a basarak seçim yapılır. Verilen süre içinde seçim yapılmazsa ilk menü satırı icra edilir.

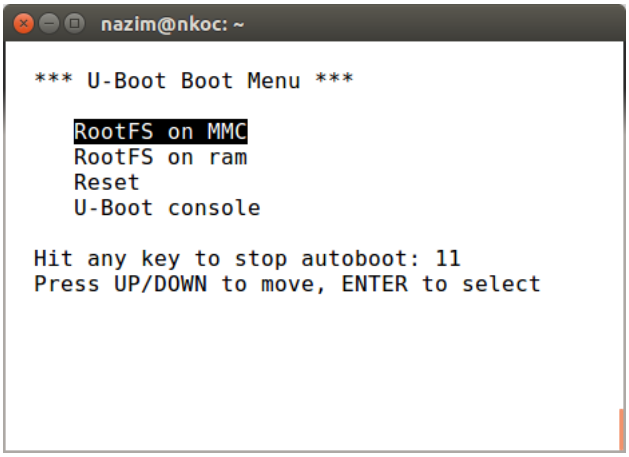
Daha önce bahsetmiştik, u-boot açılır açılmaz `bootcmd` komutunu işletir. Burada `bootmcd` komutu olarak `bootmenu 20` verilmiştir. Bu durumda u-boot hemen menü sistemini çalıştırır ve 20. saniyeden itibaren geri saymaya başlar.

Açılış sırasında, u-boot bir tuşa basılarak durdurulursa menü yerine u-boot promptu gelir. Sonra `boot` komutu girilirse, menü ekrana gelecektir.

Menü geldikten sonra "U-Boot Console" satırı seçilirse, u-boot prompt'una düşülebilir. Prompt'a boot yazılırsa tekrar menün gelir. Menüden reset girişi seçilir veya prompt'tan reset komutu girilirse, bord tekrar başlar.

Müşteriye verilecek sistemde menü olması anlamsızdır. Fakat geliştirme aşamasında aynı anda birden fazla sistemi test etmek için ideal bir yöntemdir.

Örnek menünün ekran görüntüsü Şekil 11.1'deki resimde verilmiştir.




Şekil 11.1: U-Boot menüsü.

# Bölüm 12

# Sistemin Geliştirilmesi

Bu belgede 2 adet örnek gömülü sistemin kuruluşundan bashsedilmiştir. Burada esas amaç genel kavramları vermek ve farkındalık yaratmaktır. Gömülü Linux sistemleri ile uğraşan kişiler PowerOn ile Login arasındaki her adımı çok iyi bilmeli ve her adımı farkında olarak yapmalıdır.

Bu farkındalığı yaratabilmek için kurulan sistemler çok basitçe seçilmiştir. Öyleki sistemde busybox isminde tek bir binary dosya mevcuttur. Bu şekilde kurulan sistem pek çok proje için yeterli olsa da özellikle kütüphane bağımlı sistemler için ne yapılmalıdır?

Bunun için build root[1] ile kuruluş yapılabilir. Br programı[2] hemen hemen standard dağıtımlara yakın bir Linux sistemi kurmaktadır.

Br ile toolchain kuruluşu, u-boot ve çekirdek derlemesi, kök dosya sistemi ve açılış betikleri tek seferde kurulabilir. Br aslında bir kabuk programıdır. Seçilen programları veya kütüphaneleri veya paketleri ilgili internet sitelerinden indirir, çapraz derler ve boş olan bir iskelet kök dosya sistemine kurar. Öyleki iş bittiğinde rootfs.tar isimli bir dosya elde edilir. Bu dosya örnek sistemimizde, örneğin SD kartın 3. bölümüne untar edilerek test edilebilir.

Gömülü sistem kuruluşu yapan naçizane tavsiyelerimiz,

Mümkünse gömülü sistem sadece busybox temelli kurulmalıdır.

Busybox temelli gömülü sistem yeterli olmazsa, ihtiyaç duyulan kütüphane

[1] Bundan sonra kısaca br denilecektir.

[2] https://buildroot.org/

veya programlar ilgili web sitelerinden indirilmeli ve çapraz derlenip busybox temelli sisteme eklenmelidir.

Derleme işi, yamalar veya bağımlılıklardan ötürü çok zor olabilir. Bu durumda br sisteminin tek paket derleme özelliği kullanılmalı ve elde edilen paketler busybox temelli sisteme tek tek kopyalanmalıdır.

Yok bu da yetmez denirse, kök dosya sistemi ve açılış betikleri tamamen br tarafından üretilip kullanılabilir. br zaten busybox'ı kullanmaktadır. Bunun dışında busybox içinde olmayan programları veya busybox içinde olan ama yetersiz kalan programları br sistemi ayrıca indirip derlemektedir.

Toolchain, u-boot ve kernel derlemeleri br tarafında yapılmamalı, bu belgede anlatıldığı gibi el yordamı ile yapılmalıdır. Okuyucu bu işleri br ile yaparsa ne kadar zor ve kontröl dışında olduğunu görecektir. br ile sadece kök dosya sistemi ve açılış betikleri kurulmalıdır. Ayrıca br'nin kurduğu açılış betikleri son derece kaba sabadır. Mutlaka elden geçirilmelidir.

Yok eğer bu da yetersiz denirse yocto[3] gibi gömülü sistem araçlarına geçilebilir. Yocto sistemi kaba bir tabirle, "siz elinizi suya sabuna dokundurmayın, her işi biz yaparız, siz sadece config dosyasını yazın" cinsinden bir projedir.

"Aman bir an önce bitirin, müşteri kapıda bekliyor, zaten hazır var niye uğraşıyorsunuz" diyen proje yöneticileri için yocto sistemi gayet uygundur. Bu tür hazır sistem kuruluşlarında yedekleme, güncelleme ve uzaktan yönetim gibi proje yaşam döngüsü içinde bulunan işlerin aslında ne kadar da zor olduğu, zaman ilerledikçe görülecektir. Tecrübeyle sabittir.

Bu belgede sadece 2 farklı örnek sistem verilmiştir. Çekirdek ve kök dosya sisteminin oturduğu yere bağlı olarak, pek çok gömülü sistem tekniği vardır. Projeye başlarken uygun gömülü sistem tekniği seçilmelidir.

Örneğin ağ ortamı varsa, çekirdek, dtb ve RootFS ağ üzerinden tftp+NFS yardımı ile yüklenir ve kullanılır. Kök dosya sistemi ufaksa initramfs tekniği güzeldir ama kalıcı bilgiler için SD kart veya kapanma durumunda verilerin kaybolmadığı uygun bir ortam kullanılmalıdır.

Mümkünse SD kart üzerinde kök dosya sistemi tutulmamalıdır. SD kart tüketici elektoniğinde resim vb saklamak için kullanılan bir ortamdır. Üzerinde kök dosya sistemi olmamalıdır. Profesyonel sistemlerde kök dosya sistemi NAND, NOR veya eMMC gibi ortamlarda olmalıdır.

[3] https://www.yoctoproject.org/

SD kart üzerine database kurulmamalıdır. Bu tam bir gaflettir. Bir süre sonra kart mutlaka bozulacaktır.

Kök dosya sistemi mutlaka read/only bağlanmalıdır.

Proje başlangıcında güncelleme, yedekleme, uzaktan yönetim ve acil açılış için gerekli çalışma yapılmalıdır.

Bu belgede esas amaç gömülü Linux sistemi kuruluşuna genel olarak yukarında bakmaktır. İçinde br sistemini ve daha farklı kuruluş tekniklerini barındıran daha ayrıntılı bir belge[4] daha mevcuttur. Bu belge her ne kadar BeagleBone Black için yazılmış olsa da mentalite aynıdır ve bahsedilen yöntemler OrangePi Zero için kolaylıkla uygulanabilir.

Son olarak cihazlarla gelen Linux sistemleri masaüstü sistemlerinin hemen hemen birebir aynısıdır. Bunlar gömülü sistem değildir. İçinde yok yoktur. Bundan dolayı gömülü sistem konusunda çalışanlar için inanılmaz rahat ve basit sistemlerdir. El atılan her kütüphane, her program her betik mevcuttur. Eksik olanlar da kolaylıkla paket yöneticileri ile kurulabilir.

Eğer projelerde birkaç adet gömülü sistem mevcutsa veya ödev veya prototip gibi geçici işler yapılacaksa bu tür hazır sistemler kullanılabilir. Ama proje müşteriye verilecek ve külliyatlı miktarda cihazdan oluşuyorsa hazır sistemleri kullanmak uzun vadede sadece baş ağrısı yapacaktır.

Bitti.

[4] http://ucanlinux.com/wp-content/uploads/bbb_ucanlinux.pdf