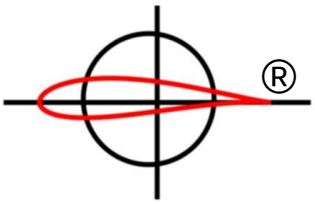
Gömülü Linux Sistemleri Eğitimi

Katılımcı Notları

http://UCanLinux.Com

22 Aralık 2016



Nâzım KOÇ

Bu belge[1] BeagleBone Black isimli tek-kart bilgisayara Linux işletim sisteminin nasıl kurulacağından bahseder.

Bu belgenin en son sürümü ucanlinux.com adresinden indirilebilir.
Bu belge ile ilgili her türlü bilgi, eleştiri ve yorum için `http://ucanlinux.com` sitesindeki iletişim bilgileri kullanılabilir.

[1] file: `/nk/workspace/projects/linux.egitimi/bbb4/latex, 7 Nisan 2017`

Şecere:

16 Mayıs 2016 İlk yayın.

23 Mayıs 2016 Bölüm 2.1.5'de tuğla yapımı ile ilgili güncelleme yapıldı, kaynaklar eklendi. Bazı yazım hataları düzeltildi.

7 Ağustos 2016 BR'nin ilk derlemesinde olabilecek hatalar için Bölüm 8.3, Sayfa 115'ye yeni paragraf eklendi. Bazı yazım hataları düzeltildi. Sayfa 31'e firmware_install paragrafı eklendi. index genişletildi. Sayfa 226'deki fopen() hataları düzeltildi.

30 Eylül 2016 Genel unbrick yöntemi 9.73 eklendi. Bazı bölümlere satır bazında eklemeler yapıldı.

22 Aralık 2016 İskelet kök dosya sisteminin nasıl elde edileceği eklendi, bakınız Sayfa 38. Dağıtıma ve firmware'e bağımlı olduğu için tuğla yapımı bölümleri iptal edildi.

7 Nisan 2017 Kapak eklendi.

İçindekiler

Şekil Listesi

Önsöz

Günümüzde, tek-kart bilgisayarlara[2] standard Linux dağıtımları yüklemek çok modadır. Çünkü artık tek-kart bilgisayarlar inanılmaz güçlenmiş ve standard bir Linux dağıtımını kaldırabilecek hale gelmişlerdir. Bu belge yazılırken RaspPi makinesi piyasaya sürülmüştür ki, 4 çekirdek ve 64 bit mimariye sahiptir. Bu makine, şu anda kullandığımız notebook'tan daha güçlüdür. Böyle olunca da gömülü Linux projelerinde, standard dağıtımlar çok yaygın ve zahmetsiz bir biçimde kullanılmaya başlanmıştır. Gömülü Sistem kavramı ölmüş müdür, can mı çekişmektedir, 112'ye haber verilmeli midir?

Gömülü sistemler[3], prensip olarak bir işi çok iyi derecede yapmak için geliştirilmesi gereken sistemlerdir. Bu tür sistemlere "adanmış sistemler" de denir.

Belirli bir iş için standard bir dağıtım kullanmak verimli bir yöntem değildir. Standard dağıtımlar aynı anda pek çok işin üstesinden gelebilmek için tasarlanmış, son derece karmaşık sistemlerdir.

Gömülü sistemlerde, standard dağıtımların kullanılması çok hızlı, basit ve ucuz bir çözümdür. Birkaç adet bilgisayardan oluşan projelerde de gömülü sistemler rahatlıkla standard dağıtımlarla kurulabilirler. Fakat bilgisayar sayısı arttıkça standard dağıtımlar maliyetli bir hale gelecektir.

1. Öncelikle standard dağıtımlar çok fazla kaynak gerektirirler. Çok fazla ram ve disk belleğe ihtiyaç duyarlar.

2. Pek çok özelliğe sahip olduklarından, bu özellikleri yönetebilmek için aşırı derecede proses çalıştırırlar.

[2]Single Board Computers

[3] Bu belgenin tamamında "gömülü Linux sistemi" yerine kısaca "gömülü sistem" ifadesi de kullanılmıştır. Gömülü sistemler, işletim sistemi olmadan da kurulabilirler. Ayrıca Linux dışında başka pek çok işletim sistemi de kullanılabilir. Bu belgede, sadece Linux işletim sistem ile kurulan gömülü sistemlerden bahsedilecektir.

3. Açılışları yavaştır.

4. Yönetimleri son derece zordur. Standard paket yönetimi dışına çıkmak pek de kolay değildir.

5. Standard dağıtımlar ani kapanmalara karşı dayanıklı değildirler. Bu soruna karşı alınan en yaygın tedbir log tabanlı dosya sistemi kullanmaktır. Yine de bu yöntem sistemin tekrar ayağa kalkması için %100 güvenli bir yol sunmaz.

6. İhtiyaç duyulmayan pek çok prosese sahip olduğu için, çökme, tıkanma, yavaşlama ve bug'lara karşı daha duyarlı olurlar.

7. Standard dağıtımların kök dosya sisteminin çökmesine karşılık bir tedbiri yoktur.

8. Standard dağıtımlarda kullanılan boot yükleyicileri genelde son derece karmaşık bir config sistemine sahiptir.

9. Benzer biçimde X server sistemleri de çok zor yönetilebilir bir yapıya sahiptir.

10. Genelde çekirdek ve kullanılan paketler, sürüm numaralarını geriden takip ederler.

11. Tek bir paketin el ile derlenip sisteme eklenmesi, paket yönetim sisteminin bütünlüğünü bozabileceğinden pek tavsiye edilmez.

Bu sakıncalara daha pek çok madde eklenebilir. Bu sakıncaları ortadan kaldırmak için projeye uygun gömülü sistemin geliştirilmesi gerekir. Tabii ki el yordamı ile projeye uygun geliştirilen bir dağıtımın da kendine has sakıncaları olacaktır.

1. Öncelikle böyle bir dağıtımın gelişitirilmesi ve test edilmesi zaman alacaktır.

2. Dağıtımı geliştirmek için boot loader, kernel, root file system derlemesi ve kurmasını bilen ve betik yazabilen tecrübeli elemanlara ihtiyaç olacaktır.

3. Dağıtım genelde donanıma bağlı olacak ve donanım değiştiği zaman ayrı bir çalışma gerekli olabilecektir.

Nihayetinde, projeye özel bir dağıtım üretildiğinde, en azından donanım gereksinimi kesinlikle standard dağıtıma göre çok daha az olacaktır. Özellikle hacimli sayıda gömülü bilgisayara sahip projelerde donanım maliyeti önemli ölçüde azalacaktır.

Bu belge BeagleBone Black makinelerine el yordamı ile kurulan gömülü Linux sisteminin baştan sona kuruluşundan ve testinden bahsedecektir. "Baştan sona" kuruluş ifadesi, power-on ile login arasındaki bütün adımları kapsayacaktır.

Herhangi bir PC veya notebook'a bir Linux dağıtımı kurulacağı zaman, kimse sıfırdan boot-loader derlemez, çekirdek derlemez, kök dosya sistemi veya açılış betiği kurmaz. Bunların tamamı hazırdır ve yığın olarak bilgisayarlara kurulur. Kuruluş bittikten sonra ufak tefek ayarlamalar ile Linux dağıtımı hazır hale gelecektir.

Aynı mantık gömülü sistemler için de geçerli olabilir. Standard herhangi bir dağıtım Beagle Board veya RaspPi veya herhangi bir tek-kart bilgisayara yüklenebilir. Çok da güzel çalışırlar. Ama çalışan bu sistem gömülü bir sistem gibi değil, PC'nin basınç altında sıkıştırılmış hali gibi görülebilir.

Yukarıda madde madde sıraladığımız nedenlerden dolayı standard bir Linux dağıtımını, gömülü sistem projelerinde kullanmak çok da verimli bir yol değildir. Peki nasıl bir yol izlenmelidir?

Bütün gömülü Linux sistemleri temelde 4 ayak üzerine kurulurlar.

1. Boot loader
2. Linux kernel
3. Root file system
4. Boot scripts

Gömülü sistem projelerinde bu ayakların her biri tek tek el yordamı ile yapılmalıdır. El yordamı ile kurulan bu sistem, çok hızlı açılacak, standard bir dağıtıma göre çok az disk kapasitesi işgal edecek ve çok daha az sayıda çalışan prosese sahip olacaktır.

Bu belgede çeşitli proje tiplerine göre yukarıdaki 4 ayak bir bütün olarak verilecektir. Bahsi geçen konuların veya kavramların çok ufak bir yüzdesi BeagleBone Black[4] sistemine bağlıdır. Çok ufak değişikliklerle, anlatılan bütün konular diğer marka gömülü cihazlara kolaylıkla uygulanabilirler.

Bu belgenin güncel hali, pdf formatında, http://ucanlinux.com adresinden indirilebilir.

Faydalı olması dileği ile,

-Nâzım KOÇ

[4]Kısaca BBB

Bölüm 1

Başlangıç

Bu belge gömülü Linux sistemleri eğitiminde, uygulama ve labratuvar çalışmalarında kılavuz olması için hazırlanmıştır. Eğitimdeki tek amacımız, PowerOn ile Login arasında uygulanan her adımı farkında olarak yapmaktır.

Bu belgede bahsi geçen konuların tamamı, doğrudan bord veya geliştirme makinesi üzerinde uygulanabilir niteliktedir. Geliştirme bordu olarak BBB seçilmesine rağmen kavramların tamamı bütün gömülü Linux sistemleri için gerçelidir.

Eğitim çalışmalarında, power-on aşamasından, login aşamasına gelene kadar[1], arada bulunan bütün adımlar gerçeklenecektir. Bütün işlemler el yordamı ile yapılacak ve hiç bir yardımcı betik veya program kullanılmayacaktır.

Login aşamasından sonrası bu eğitimin kapsama alanı dışındadır. Çünkü bu aşamadan sonraki uygulamaların tamamı proje bağımlıdır ve genel bir eğitimin içeriğine eklenmesi pek zordur.

Ayırca gömülü sistemler kurulduktan sonra, aşağıdaki konularda da, projenin gerektirdiği şekilde çalışma yapılmalıdır.

1. sistemin kapanmaya dayanıklı kılınması,

2. içeriğinin yedeklenmesi,

[1] Bundan dolayı ucanlinux.com sitesinin sloganı "Login'e Kadar Linux" seçilmiştir.

3. kök dosya sistemi veya uygulamaların yerinde veya uzaktan güncellenmesi,
4. sisteme uzaktan, güvenli erişim,
5. uygulamaların gömülü sistem mantığına uygun geliştirilmesi,
6. uygulamaların otomatik başlatılması,
7. monitör edilmesi,
8. alarm sistemlerinin kurulması ve kullanılması,
9. log'lama teknikleri, vs.

Eğitim boyunca yeri geldiğinde ve vakit oldukça, bu konulardan kısaca bahsedilecektir.

GUI ortamlarında bulunmayı haz etmediğimizden belgede bulunan bazı çizimler ASCII karakterlerle yapılmıştır. Çok şık gözükmese de işe yaramaktadır.

Yapılan işlerin farkındalığını artırmak için bütün eğitim terminaller üzerinden yapılacak, GUI[2] ve fare kullanılmayacaktır. Katılımcı, dosya sistemi kurmayı, mount etmeyi, bir dizini taşımayı, bir loop cihazı kurmayı veya bir diski bölümlendirmeyi el yordamı ile yapacak ve yapılan işin farkına varacaktır. Neredeyse her Linux kullanıcısı bu işleri bir biçimde yapmaktadır. Fakat bu işleri farkında olarak yapan çok az kullanıcı bulunmaktadır.

Aynı şekilde sadece bir iki komut ile BBB üzerine veya başka bir borda gömülü sistemler kolaylıkla kurulmaktadır. Fakat burada kullanılan katma değer veya bilgi miktarı çok ama çok az olduğu için, projelere uygun güncelleme yapmak neredeyse imkansız hale gelmektedir.

Gömülü Linux eğitiminin tek bir amacı vardır: Yapılan bütün işleri farkında olarak yapmak...

Eğitimin verimli olabilmesi için aşağıdaki gibi bir geliştirme ortamının kurulması tavsiye edilir.

Eğitimde Host tarafında mutlaka standard bir Linux dağıtımı kullanılmalıdır. Ubuntu kullanılması tavsiye edilir. Ubuntu dışındaki dağıtımlarda katılımcı paket veya program yükleme ve çalıştırma becerisine sahip olmalıdır.

[2] Şimdiye kadar bu fikrimi pek kabul eden olmasa da GUI'ler ve thread'ler yazılım mühendisiliği içine sokulmuş şeytanlardır.

Ubuntu kullanan katılımcının aşağıdaki paketleri eğitimden önce yüklemesi gereklidir.

```
$ apt-get install ssh vim-gnome libncurses5-dev lzop  \
          xinetd tftpd tftp isc-dhcp-server minicom nfs-kernel-server
```

Microp$oft Windows veya windows altında çalışan Linux emülatörleri kesinlikle kullanılmamalıdır. İşin fıtratına aykırıdır. Çünkü ağ yapılandırma sorunları, çevre cihazlarının adlandırılması ve derleme sürelerinin çok uzaması gibi pek çok sakınca çıkarmaktadır. "Ben bunların üstesinden gelirim" diyen katılımcı bu garip ortamda çalışabilir. Yine de tavsiye edilmez.

Katılımcının makinesinde en az 5 GB'lık boş disk alanı bulunmalıdır.

Eğitimde genel olarak aşağıdaki konulara değinilecektir.

1. toolchain kullanma,
2. u-boot derleme,
3. İskelet RootFS kuruma,
4. busybox destekli RootFS kuruluşu,
5. BR destekli RootFS kuruluşu,
6. Çekirdek derleme,
7. Açılış betiklerinin yazılması,
8. Bölümlendirme teknikleri,
9. initramfs teknikleri,
10. tftp, dhcp kullanımı,
11. ağ üzerinden açılış teknikleri,
12. eMMC üzerinden açılış teknikleri, vs.

Bu belgedeki her bir bölüm, baştan sona bir gömülü Linux tekniğinden bahsetmektedir. Pek çok gömülü sistem kuruluş tekniği mevcuttur. Bu belgede en çok kullanılan teknikler verilmiştir. Ayrıca verilen teknikler birbirleri ile

kombine edilerek başka kuruluşlar yapılabilir. Bütün mesele projenin gerektirdiği yöntemi tespit edebilmektir.

Gömülü sistemleri birbirinden ayıran esas özellik çekirdeğin ve kök dosya sisteminin[3] bulunduğu ortamdır.

Çekirdeğin kendisi, eMMC, MMC, NAND Flash, NOR flash, SD Kart, USB Bellek veya ağ üzerinde uzakta bir yerde oturuyor olabilir. Bu çekirdek doğrudan veya ağ üzerinden veya seri kablo ile veya başka tekniklerle yüklenebilir. Yani çekirdeğin oturduğu yer ve bu yere göre seçilen yükleme tekniği bizim birinci ayırıcı özelliğimizdir.

Çekirdeğin açılır açılmaz bağladığı dosya sistemine kök dosya sistemi denir. Kök dosya sistemi de çekirdek gibi aynı veya farklı bir ortamda bulunabilir. Hatta çekirdeğe gömülü bile olabilir. İşte ikinci ayırıcı özellik kök dosya sisteminin oturduğu yerdir. Çekirdek bu yere göre bir mount tekniği seçer.

Bu eğitim belgesi bu iki ayırıcı özellik üzerine inşa edilmiştir. Örneğin ilk kurulacak sistemde çekirdek ağ üzerinde uzaktaki bir makinede ve kök dosya sistemi ise çekirdeğin doğrudan içine gömülecektir. Belgede bulunan her gömülü sistem uygulamasında ya çekirdek ya kök dosya sistemi farklı bir ortamda olacaktır. Bölüm başlarında A.B şeklinde bir ifade olacaktır. A kelimesi çekirdeğin bulunduğu ortamı, B kelimesi ise kök dosya sisteminin bulunuduğu ortamı temsil eder. Örneğin tftp.eMMC ifadesinde, tftp kelimesi, çekirdeğin uzaktaki bir makinede oturduğunu, tftp ile ağ üzerinden yükleneceğini gösterir. eMMC kelimesi de kök dosya sisteminin eMMC üzerinde oturduğunu belirtir.

Benzer şekilde çok fazla sayıda ortam bulunmaktadır. Her ortamın A.B şeklindeki kombinasyonu pek çok sayıda gömülü sistem tekniği ortaya çıkarır. Bu belgede mevcut kombinasyonların birkaçından bahsedilecektir. Katılımcı verilen bilgilerle bütün gömülü sistem tekniklerini kendi başına uygulayabilecektir.

Açılış yükleyicisi[4] ve açılış betikleri pek nadir değiştiği için, ikinci plana atılmıştır. Her bölümde çekirdek ve kök dosya sistemi farklı ortamlarda otursa da açılış yükleyicisi her zaman cihaz içindeki bir ortamda oturmaktadır. BBB sisteminde ya eMMC ya da MMC üzerinde bulunur.

Pek çok açılış yükleyicisi mevcuttur. Günümüzde en moda açılış yükleyicisi, ARM sistemler için u-boot'tur. Bu belgede sadece u-boot'tan bahsedilecektir.

[3] Kısaca RootFS veya rootfs şeklinde de yazılacaktır.

[4] boot loader

Benzer şekilde açılış betikleri de hemen hemen hiç değişmemektedir. Tekrar hatırlatmak isteriz ki, açılış yükleyicisi, çekirdek, kök dosya sistemi ve açılış betikleri sistemin bütününü oluşturur. Çekirdek ve kök dosya sisteminin bulunduğu ortam ise gömülü sistemin genel karakteristiğini oluşturur.

BBB ile veya RaspPi ile gelen standard Linux dağıtımlarında, çekirdek ve kök dosya sistemi genelde SD/MMC ve/veya eMMC üzerinde bulunur. Bu belgede her iki teknikten de ayrıca bahsedilecektir.

Bu çalışma belgesi tek başına yeterli değildir. Bir eğitmen eşliğinde konular işlenmelidir. Bu belgenin bir kitap haline getirilmesi üzerinde halen çalışılmaktadır. Kitap haline geldikten sonra, kişiler eğitimci olmadan da gömülü Linux üzerinde kendi başlarına çalışabileceklerdir.

Bölüm 2

Initramfs Destekli Gömülü Sistem-I

En basit gömülü sistemlerden birisi, initramfs[1] destekli gömülü sistemlerdir. Bu bölümdeki uygulamada, kök dosya sistemi doğrudan çekirdeğe gömülüdür. Sistemin blok şeması Şekil 2.1'de verilmiştir. Çekirdek ağ üzerinde oturur[2], RootFS ise çekirdeğin içine gömülüdür[3]

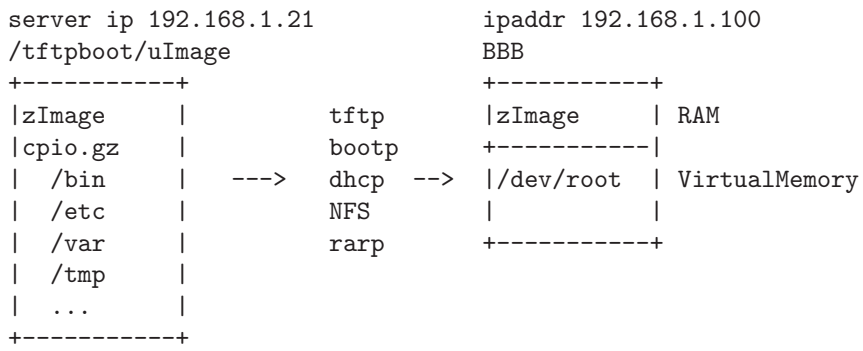
Kök dosya sisteminin RAM içinde oluşturulmasına ve çekirdek tarafından bağlanmasına initramfs denir. Initramfs, açılış sırasında doğrudan çekirdek tarafından kurulur ve bağlanır.

initramfs tekniği 2 farklı biçimde uygulanır. Kök dosya sistemi, ya bu bölümde yapılacağı gibi çekirdek içine gömülür ya da ayrı bir dosya olarak çekirdekten bağımsız yüklenir. Bu bölümde çekirdeğin içine gömülü olarak yükleme yapılacaktır.

initramfs tekniğinin esas amacı, asıl kök dosya sistemini bağlamadan önce hazırlık yapmaktır. Standard dağıtımlarda, genelde önce initramfs sistemi yüklenir, bu sistem gerekli çekirdek modüllerini yükler ve sonra yok olur. Çekirdek de esas kök dosya sistemini yükler ve açılışa devam eder.

Fakat gömülü sistemler için bu mantık pek kullanılmaz. Gömülü sistemler standard dağıtımlar gibi çok farklı yan birimlere veya donanımlara bağlı

[1] Intermedite Root File System veya Initial RAM File System
[2] Sembolik olarak tftp.* ile ifade edilebilir
[3] Sembolik olarak *.kernel ile ifade edilebilir.



Şekil 2.1: En basit gömülü sistem, tftp.kernel

değildirler. Genelde son derece katıdırlar ve yöneteceği cihazlar sabittir. Bundan dolayı initramfs, ön hazırlık yapan bir dosya sistemi yerine kök dosya sistemi olarak kullanılabilir.

Bu çalışmada gömülü sistemin tamamı ağ üzerinden yüklenecektir. Çünkü çekirdek yüklenince, çekideğe gömülü kök dosya sistemi de çekirdekle birlikte otomatik olarak yüklenmiş olacaktır. Bu sayede çekirdek derlemesi ve RootFS üzerinde çok hızlı güncelleme yapılabilecektir. Çünkü güncellemeler gömülü cihazda değil, host[4] tarafında yapılacaktır.

Initramfs sisteminin bazı özellikleri aşağıdaki gibi sıralanabilir.

1. Bu sistem acil kurtarma sistemi olarak kullanılabilir.
2. Geliştirme, test veya müşteri sistemi olarak kullanılabilir.
3. Kök dosya sistemi çok basittir ve pek çok iş için yeterli değildir.
4. RootFS, mevcut RAM'a oranla ufak olmalıdır.
5. Çekirdeğe ek yapıldığı için kaynak kodları da müşteriye verilmelidir.
6. RootFS her değiştiğinde çekirdek yeniden derlenmelidir.
7. Kök dosya sistemi mount edilmediği için, doğrudan cpio arşivinden initramfs kurulduğu için açılışta kök dosya sisteminin bozulması ihitmali hiç yoktur.

[4]PC, notebook veya host kelimeleri, standard bir Linux dağıtımı yüklü olan cihazları temsil etmektedir.

8. Sistemin genel yedeği çok kolay alınır.

9. Ani kapanmalarda RootFS'in bozulma ihtimali yoktur. Çünkü her açılışta RootFS yeniden kurulur.

10. RootFS'si güncellemek çok zordur. Çünkü çekirdek de değişmelidir. Gömülü bir sistemde çekirdeği güncellemek risklidir.

11. Kolaylıkla yedek bir sistem kurulabilir. Çekirdeğin kopyasını bir yerlerde bulundurmak yeterlidir. u-boot sistemi, ilk çekirdeği açamazsa ikinci çekirdekten kolaylıkla açılış yapabilir.

12. Kalıcı bilgiler için ayrı bir ortam kullanılmalıdır.

13. vs.

Yukarıda sıralanan özellikler bu gömülü sistemin değil, initramfs sisteminin genel özellikleridir. Örneğin bu bölümde bahsedilen initramfs sistemi, çekirdekle birlikte tftp yardımı ile yüklenir. Aynı initramfs sistemi eMMC ile de yüklenebilir. Yukarıdaki özellikler yine geçerlidir.

Bu özellikler gömülü sistem projesinin gereklerine uymuyorsa, bu tür bir gömülü sistem kullanılmamalıdır. Diğer bir deyişle, bir gömülü sistem projesine başlandığında, her gömülü sistem tipinin özellikleri çok iyi bilinmelidir ki projenin ortasında saç baş yolunmasın.

Bu bölümdeki gömülü sistem ağ üzerinden tftp ile yüklenecektir. Ayrıca ağ yapılandırması için de başka ağ protokolleri kullanılacaktır.

Katılımcı bu sistemi eksiksiz olarak anlamalıdır. Bundan sonra bahsedilecek bütün gömülü sistem örnekleri bu çalışmayı esas alacaktır.

Bu gömülü sistem için aşağıdaki çalışmalar yapılacaktır.

1. u-boot, kernel ve rootfs sıfırdan derlenecek ve kurulacak,

2. u-boot, doğrudan SD karttan yüklenecek,

3. tftp ile çekirdek+kök dosya sistemi yüklenecek,

4. login elde edilecektir.

Bu belgede bahsedilen bütün gömülü sistemler aşağıdaki metodoloji ile işlenecektir:

Bütün sistemler 4 ana başlık altında incelecektir: Açılış yükleyicisi, çekirdek, kök dosya sistemi ve açılış betikleri. Her başlık içinde test alt başlığı da bazen olacaktır. Ayrıca her bölümün başında genel bilgiler verilecektir.

2.1 Açılış Yükleyicisi

Açılış yükleyicisi[5] olarak u-boot[6] kullanılacaktır. U-boot sistemi, GPL lisanslı olup, neredeyse bütün ARM makineler için "sanki standard[7]" hale gelmiştir. Son derece modülerdir. Her firma kendi borduna göre sistem üzerinde çok kolay güncelleme yapabilmektedir.

U-Boot temelde 2 programdan oluşur. Birinci programa MLO[8] denir. Bu çok basit ve ufak bir koddur. Tek başına sistemin bütününü ayağa kaldıramaz. Fakat bordun ROM'unda bulunan basit bir program ile kolaylıkla yüklenebilir ve başlatılabilir. Bord tarafından doğrudan yüklenen, ama sistemin bütününü açma yeteneği olmayan programlara birinci aşama açılış yükleyicisi[9] denir.

MLO[10] kodu sistemi tamamen açmaya yetmez ama yükleyicinin ikinci parçası için gerekli adresleri atayıp, çevre birimlerinin ayağa kalkmasını sağlar.

MLO tarafından u-boot yükleyicisinin esas programı olan uboot.bin[11] yüklenir. Bu programa ikinci aşama boot yükleyicisi[12] denir.

[5] boot loader

[6] Kaynak: http://processors.wiki.ti.com/index.php/AM335x_U-Boot_User"s_Guide

[7] Decfacto Standard

[8] MmcLOder veya MasterLOder kelimelerinin kısaltmasıdır.

[9] first stage boot loader

[10] U-Boot kullanan her gömülü sistemde MLO olmayabilir. Örneğin RPi'de MLO mevcut değildir. Fakat MLO yerine geçen, farklı adla da olsa, birinci derece açılış yükleyicisi bulunur.

[11] u-boot.bin ile u-boot.img aynı programdır. Tek farkı u-boot.img'nin tepesinde özel bir başlık bulunmasıdır. U-boot kaynak kodundan derlendiğinde, aynı anda her 2 program da üretilir. Bazı bordlar u-boot.img ile bazılarıysa doğrudan u-boot.bin ile bordu açabilirler. Bütün mesele bordun ROM açılış yükleyicisindedir. u-boot.bin ile açılış yapmak daha yetenekli ROM açılış yükleyicisi gerektirir.

[12] second stage boot loader

İkinci aşama yükleyicisi eskiksiz bir yükleyicidir ve inanılmaz yeteneklidir. Ağ üzerinden ve yerel cihazlardan çekirdek, dtb, kök dosya sistemi yükleyebilir. Çekirdeğe parametre aktarabilir. Etkileşimli çalışabilir. U-boot sistemi başlı başına bir çalışma konusudur[13].

U-boot sistemi her bord için farklı komut setine sahiptir. Bir bord için kullanılan bir komut başka bir bordda bulunmayabilir. Örneğin güvenlik amacı ile POS makinelerinde bulunan u-boot sistemlerinde, ağ ile ilgili komutlar genelde çıkarılır, iptal edilir.

Bundan dolayı her makinenin u-boot sistemi birbirinden farklı ama kavramlar aynıdır. Hangi makine için olursa olsun, u-boot destekli açılışlarda her zaman 3 programa ihtiyaç duyulur. MLO, uboot.bin ve mkimage[14] programları.

MLO ve u-boot.bin programlarına birinci ve 2. aşama açılış yükleyicisi demiştik. mkimage programı ise doğrudan u-boot.bin programı için bir yardımcıdır.

u-boot.bin programı kullanacağı her dosyanın başına, ki bu dosya bir betik, bir çekirdek dosyası, bir cpio imajı veya bir kök dosya sistemi olabilir, 64 baytlık özel bir başlık[15] ister.

mkimage programı, u-boot.bin için gerekli olan 64 baytı, ilgili dosyanın başına ekler. Bu 64 baytlık bilgi olmazsa, u-boot.bin programı ilgili dosya üzerinde işlem yapamaz.

Örneğin çekirdeği derledikten sonra zImage kodunun en tepesine mkimage komutu ile 64 baytlık bilgi yazılır. Bu bilgi içinde çekirdeğin yüklenme adresi, denetim baytları, yürütme adresi gibi bilgiler bulunur. Bu bilgilerin tam listesi için bakınız Ekler 9.43. Sırası geldiğinde mkimage üzerinde çalışma yapılacaktır.

Bordu açabilmemiz için öncelikle MLO, u-boot.bin ve mkimage programlarını elde etmemiz gerekir.

Bu bölümde kısaca aşağıdaki çalışmalar yapılacaktır.

[13]Bakınız: Ekler 9.43 9.42 9.44

[14]Bakınız: Ekler 9.43

[15]Bu belge boyunca kullanılan bütün imaj başlıklarının boyu tam 64 bayttır. Aynı anda birden fazla imaj kullanılırsa başlığın boyu uzayacaktır. Fakat örnek sistemlerin tümünde tek imaj kullanılacağı için başlık boyu hep 64 bayt olacaktır. Çoklu imajlarda başlık boyu hesabı için http://www.isysop.com/unpacking-and-repacking-u-boot-uimage-files/ adresine bakılabilir. Boy hesabı son derece basittir ve şu an için konumuz dışındadır.

1. Çapraz derleyici[16] yüklenecektir.
2. U-Boot'un kaynak kodu yüklenecektir.
3. Çapraz derleyici ile u-boot derlenecek ve MLO, u-boot.bin ve mkimage programları elde edilecektir.
4. fdisk ile SD karta bölümlendirme yapılacaktır ve dosya sistemi kurulacaktır.
5. Kurulacak dosya sistemine u-boot programları ve u-boot çevre değişkenleri kopyalanacaktır.
6. Test amacı ile sistem açılacak ve u-boot seviyesinde testler yapılacaktır.
7. Mutlu olunacaktır.

Bu çalışmalar ömür boyu 1 kez yapılacaktır. Bundan dolayı katılımcının bütün adımları çok iyi kavraması gerekmektedir. En önemli madde, mutlulukla ilgili son maddedir. Çünkü u-boot seviyesinde açılış gerkçeklenirse, gömülü sistem projesinin en önemli ayağa tamamlanmış demektir. Bu adımdan sonra çekirdek, kök dosya sistemi ve açılış betiklerini test etmek için u-boot pek çok imkan sağlar.

U-Boot açılışı tamamlanmadan bir hata meydana gelirse, bu hatayı bulmak genelde çok zaman kaybettirir. Çünkü kullanıcının karşısında, genelde kapkara bir ekrandan başka bir bilgi bulunmaz.

2.1.1 Çapraz Derleyiciler

Kendi işlemci mimarisinden farklı bir işlemci mimarisi için kod üreten programlara çapraz derleyici denir. x86 makine üzerinde kuracağımız derleyici ARM mimarisi için kod üretecektir. O halde bu derleyici çapraz bir derleyicidir.

RaspPi makinesinde bulunan derleyici ARM için kod üretir. Yani kendi işlemcisi için kod üretir. O halde RaspPi içindeki derleyici native[17] bir derleyicidir.

[16] cross compiler
[17] Kendi işlemcisi için kod üreten derleyici

RaspPi içine x86 kod üreten bir derleyici yüklersek, bu derleyici çapraz derleyici olacak ve x86 için kod üretecektir.

Kod geliştirmek için sadece derleyici yeterli değildir. Debugger, linker, dump programları gibi programlara da gerek vardır. Kod geliştirmede kullanılan bu programların topluluğuna toolchain[18] denir.

Bir program derlenir ve object kod üretirlir. Object kod linker ile birleştirilir. Linker ile birleştirilen kod debug edilir vs. Kısaca bir programın çıkışı diğer programın girişi olur. Bu işlemler arka arkaya dizilirse zincir gibi gözükür. Bundan dolayı zincir içindeki programların topluluğuna, yani gcc, linker, gdb gibi programların bütününe toolchain denir. Birebir çeririrsek "araç zinciri" denilebilir. Ama bana kışın oto lastiklerine takılan zinciri hatırlatıyor.

Her bord üreticisi mutlaka berlirli bir derleyici ve hatta belirli bir derleyicinin belirli bir sürümünü tavsiye eder. U-boot, kernel ve uygulama programları derlemelerinde farklı derleyici tavsiye eden üreticiler dahi vardır. Kısaca burada bord üreticisinin tavsiye ettiği derleyici ve sürüm kullanılacaktır, her zaman da böyle yapılmalıdır. Yoksa hiç beklenilmeyen bir anda "illegal instruction" hatası alınabilir.

BBB üreticisi Linaro'nun 4.8 sürümlü derleyicisini[19] tavsiye etmektedir. Bu derleyici ve yardımcı programları aşağıdaki gibi indirilip kurulabilir ve test edilebilr.

```
$ cd toolchain

# 64 bitlik host makineler için...
# https://snapshots.linaro.org/openembedded/sources/\
#    gcc-linaro-4.8-2015.06-x86_64_arm-linux-gnueabihf.tar.xz

# 32 bitlik host makineler için...
#
$ wget http://releases.linaro.org/14.04/components/toolchain/binaries/\
gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux.tar.bz2

$ tar jxvf gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux.tar.bz2
$ ln -s gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux arm

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

[18]Bu kelime için bilişim derneğinin sözlüğünde bir karşılık bulamadık. Kulağı tırmalamayan bir karşılık bulunduğunda kullanılacaktır.

[19] Host makine 64 bit ise gcc-linaro-*x86_64_arm-linux-gnueabihf.tar.xz gereklidir.

```
# Ubuntu'da yüklü olanla karışmasın diye önce toolchain yolu yazılır.

# Aşağıdaki gibi kontrol etmeyi unutma.
$ which arm-linux-gnueabihf-gcc
$ /nk/workspace/projects/linux.egitimi/bbb4/toolchain/arm/\
bin/arm-linux-gnueabihf-gcc

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

$ ls -l `arm-linux-gnueabihf-gcc -print-sysroot`
drwxr-xr-x 2 nazim nazim 4096 Mar 21  2014 etc
drwxr-xr-x 4 nazim nazim 4096 Mar 21  2014 lib
drwxr-xr-x 2 nazim nazim 4096 Mar 21  2014 sbin
drwxr-xr-x 8 nazim nazim 4096 Nis 16  2014 usr
drwxr-xr-x 3 nazim nazim 4096 Mar 21  2014 var

# lib içinde, ARM için hazır kütüphaneler var.

$ arm-linux-gnueabihf-gcc -print-search-dirs

# Ayrıca -with-sysroot diye bir değişken de var.

# Ubuntu:
$ apt-get install gcc-arm-linux-gnueabi  veya
$ apt-get install gcc-arm-linux-gnueabihf

# Ayrıca standard lib'ler ubuntu için
# /usr/arm-linux-gnueabihf/lib altındadır.
# Fakat bunu şimdilik kullanmıyoruz.

$ /usr/bin/arm-linux-gnueabihf-gcc -v
# Ubuntu'daki ve bendeki 4.8
```

Yukarıdaki ifadelerde $ işareti terminalden girilecek komutları gösterir. # işareti ise açıklama satırlarıdır.

Aynı derleyici apt-get ile de kurulabilir. Fakat şu an için kurulmamalıdır.

Derleyici kurulduktan sonra aşağıdaki gibi bir çalışma mutlaka yapılmalıdır.

1. hello.c, hem gcc hem cross derle.
2. ls, file ile incele.
3. strip yap, sonuçları karşılaştır.

2.1.2 U-Boot'un İndirilmesi

Çapraz derleyicimiz artık hazırdır. U-Boot'un kaynak kodu indirilip, MLO, u-boot.bin ve mkimage elde edilebilir.

U-Boot'un kaynak kodu denx.de sitesinde bulunabilir. Fakat her üretici u-boot üzerinde yaptığı değişikliği u-boot'un ana kodu içinde hemen eklemez. Bundan dolayı, üreticinin göstermiş olduğu adresteki u-boot kodu indirilmelidir. U-Boot kodu aşağıdaki gibi indirilebilir.

```
$ git clone git://git.denx.de/u-boot.git
$ cd u-boot/
$ git checkout v2015.10 -b tmp
$ wget -c https://rcn-ee.com/repos/git/u-boot-patches/v2015.10/\
0001-am335x_evm-uEnv.txt-bootz-n-fixes.patch
$ patch -p1 < 0001-am335x_evm-uEnv.txt-bootz-n-fixes.patch

# Kaynak kodu içinde,
# u-boot/board/ti/am335x/README'ye bakılabilir.
#
# config dosyaları,
# u-boot/include/configs/am335x_evm.h        # Doğrudan BBB tanımları.
#                       /ti_am335x_common.h
#                       /ti_armv7_common.h   # Ti ARMv7 boards, genel tanımlar.
```

Kaynak kod git şeklinde verilmişse, mutlaka git [20] ile elde edilmelidir. tar.gz şeklinde alınmamalıdır. Çünkü git ile indirilen kaynak kod içinde çeşitli dallar vardır ve test amacı ile dallar arasında kolaylıkla geçiş yapılır. "$ git pull" komutu ile de bütün güncellemeler bir kerede indirilebilir. git ile kaynak kodu indirmenin en büyük sakıncası, bütün sürümleri barındırdığı için aşırı büyük olmasıdır.

2.1.3 U-Boot'un Derlenmesi

Çapraz derleme genelde 2 türlü yapılır.

İlk yöntemde ARCH ve/veya CROSS_COMPILE değişkenleri export ile env değişkeni olarak kaydedilirler. Örneğin,

[20] Bakınız Ekler 9.33

```
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
```

U-Boot derlemesi için ARCH değişkenine gerek yoktur. Ama sonraki bütün derlemelerde gerekli olacaktır.

İkinci yöntemde ise make komutuna argüman olarak girilirler. Örneğin,

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C $UBOOT_SRC O=$UBOOT_OUT distclean
```

Bu belge boyunca ilk yöntem kullanılacaktır. Kabuk olarak bash kullanılıyorsa, export tanımlarının kalıcı olması için ~/.bashrc dosyası kullanılabilir.

U-Boot, aşağıda verildiği gibi derlenmelidir[21].

```
$ make -C $UBOOT_SRC O=$UBOOT_OUT distclean
$ make -C $UBOOT_SRC O=$UBOOT_OUT am335x_evm_defconfig
$ make -C $UBOOT_SRC O=$UBOOT_OUT -j2

$ cp $UBOOT_OUT/tools/mkimage /usr/local/bin

$ ls -l $UBOOT_OUT/MLO
$ ls -l $UBOOT_OUT/u-boot.img
$ ls -l $UBOOT_OUT/tools/mkimage
```

Bütün bu belge boyunca *_SRC ve *_OUT isimli çevre değişkenleri[22][23] kullanılacaktır. Adından da anlaşılacağı gibi UBOOT_SRC değişkeni, u-boot'un kaynak kodunun oturduğu yeri gösterir. UBOOT_OUT ise derleme sonunda üretilen bütün dosyaları gösterir. Kaynak kodunun doğrudan içinde derleme yapılması tavsiye edilmez. Bu derleme tekniği ile kaynak kod ve Makefile[24] tarafında üretilen bütün dosyalar fiziksel olarak birbirlerinden ayrılırlar.

[21] Bu yöntemde kaynak kodları ile derlenmiş kodlar birbirlerinden C ve O seçenekleri ile ayrılırlar. Bu seçenekler olmadan da derleme yapılabilir. Bu durumda doğrudan kaynak koduna cd ile gidilmeli ve make komutu C ve O seçenekleri olmadan verilmelidir. Bu belge boyunca C ve O seçenekleri ile derleme yapılacaktır. C seçeneği zaten "change directory" demektir. Yani derleme yapmadan evvel C ile verilen yere otomatik olarak gidilir.

[22] environment variables

[23] Bakınız: Ekler 9.51

[24] Bakınız: Ekler 9.32

Derleme sonunda MLO, u-boot.bin veya u-boot.img ve mkimage elde edilir. MLO ile u-boot.bin programları SD karta yazılacaktır. mkimage ise x86 veya geliştirme tarafındaki makinede kullanılacaktır. Uygun bir yere kopyalanır. Örnekte /usr/local/bin dizini kullanılmıştır.

Derleme bittikten sonra, file, ls veya more ile şu dosyalar incelenmelidir: MLO, *.map, u-boot.*, tool/mkimage

2.1.4 SD Karta U-Boot'un Kuruluşu

BBB'nin SD karttan açılabilmesi için katın bölümlendirilip, ilk bölüme vfat dosya sisteminin kurulması gerekir.

Öncelikle SD kart, PC'ye takılarak aşağıdaki gibi bölümlendirme yapılır. OMAP3x boot rom'un çalışabilmesi için bölümlendirme tablosunda[25] 255 head ve 63 sector/track olmalıdır.

Disklerde ilk 1MB'lık alan açılış yükleyicileri için ayrılır. Burada da ilk 2048 sektör yani toplam 1MB'lık alan açılış sektörü için ayrılmıştır. Fakat BBB doğrudan vfat üzerinden MLO yükler, bu alanı kullanmaz.

Bölümlendirme tablosu, ilk sektör yani MBR içinde bulunur. Kart bölümlendirilmeden önce içinin sıfırlanması, belgelerde tavsiye edilmektedir. Fakat bu işlem bazen kartın tuğla olmasına sebep olmaktadır. Tavsiyemiz, önce sıfırlama yapmadan bölümlendirme yapmak, sistem açılmazsa sıfırlama yapmaktır. Sıfırlama işi, aşağıda /dev/zero barındıran komutla yapılır.

Aşağıda verilen MMC_DEVICE ismi genelde /dev/mmcblk0 şeklindedir. Fakat her makinede aynı olmak zorunda değildir. dmesg ile kesin bilgi elde edilebilir.

```
$ dd if=/dev/zero of=$MMC_DEVICE bs=512 count=1

$ fdisk -H255 -S63 $MMC_DEVICE << EOF
o
n
p
1
```

[25] Kaynak: http://processors.wiki.ti.com/index.php/SD/MMC_format_for_OMAP3_boot

```
+$BOOT_SIZE
a
1
t
c
n
p
2

+$ROOT_SIZE
p
w
EOF

sync
```

Artık SD kart bölümlendirilimiş[26] ve dosya sistemi kurulmaya hazır hale gelmiştir. Yukarıdaki örnekte, ilerde kullanmak için ayrıca bir bölüm daha ayrılmıştır. Nihayetinde bölümlendirme işi bittiğinde aşağıdaki gibi bölümlendirme tablosu elde edilmelidir.

```
$ fdisk /dev/mmcblk0

Command (m for help): p

Disk /dev/mmcblk0: 3904 MB, 3904897024 bytes
4 heads, 16 sectors/track, 119168 cylinders, total 7626752 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

        Device  Boot     Start         End      Blocks   Id  System
/dev/mmcblk0p1   *        2048       34815       16384    c  W95 FAT32 (LBA)
/dev/mmcblk0p2           34816      100351       32768   83  Linux
```

Bu tablodaki en hayati özellik, 1. bölümün c tipinde bir FAT bölümü olmasıdır. Ayrıca bu bölümün Boot kolonunda gösterilen boot flag değeri işaretlenmiş olmalıdır.

SD kartın ilk bölümüne aşağıdaki gibi vfat dosya sistemi kurulur.

[26] dd komutu MBR'yi tamamen sıfırlar. Her ne kadar BBB ile ilgili belgelerde bu tavsiye edilse de, SD kartın tamamen bozulmasına veya Linux tarafından görülmemesine sebep olmaktadır. Başlagıçta yapılmaması tavsiye edilir. Ayrıca fdisk ile verilen -H ve -S parametreleri Ubuntu'nun farklı dağıtımlarında farklı davranış göstermektedir. Başlangıçta bu parametreler es geçilebilir. sistem açılmazsa, parametreler verilmelidir.

```
$ mkfs.vfat -n $VOLUME_NAME ${MMC_DEVICE}p1
```

Artık SD kart üzerinde, kullanıma hazır bir dosya sistemi vardır. BBB'nin bu dosya sisteminden açılabilmesi için bu dosya sistemine aşağıdaki dosyalar kopyalanmalıdır.

1. MLO
2. u-boot.img
3. uEnv.txt

MLO mutlaka en önce kopyalanmalıdır. Aşağıdaki örnekte dosya sistemi mount edilmekte ve gerekli kopyalama işlemleri yapılmaktadır.

```
$ mount ${MMC_DEVICE}p1    $BOOT_MP

$ cp $UBOOT_OUT/MLO        $BOOT_MP
$ cp $UBOOT_OUT/u-boot.img $BOOT_MP
$ cp uEnv.txt              $BOOT_MP
```

uEnv.txt dosyasının içeriği aşağıda verilmiştir. U-Boot ayağa kalkarken bu dosyanın içeriğini okuyarak kendi değişkenlerini günceller.

```
# uEnv.txt
ipaddr=192.168.1.100

serverip=192.168.1.21

bootargs=console=ttyO0,115200n8

tftp_test=tftpboot 0x80F80000 am335x-boneblack.dtb ; \
          tftpboot 0x80007FC0 uImage ; \
          bootm 0x80007FC0 - 0x80F80000

uenvcmd=run tftp_test
```

uEnv.txt dosyası bu aşamada kullanılmayacaktır. Bilgi amacı ile verilmiştir. Yeri geldikçe uEnv.txt içeriğinden bahsedilecektir. Kısaca değişkenler üzerinden geçecek olursak,

ipaddr BBB'ye statik IP değeri.

serverip tftp sunucusunun IP değeri.

bootargs Çekirdeğe aktarılacak parametreler. U-Boot burada yazılan her bilgiyi gözü kapalı çekirdeğe aktarır. İçeriği ile ilgilenmez.

tftp_test Birkaç komuta verilen kısa bir isim. Tamamen keyfidir.

uenvcmd U-Boot açıldıktan sonra otomatik olarak çalıştırılacak komut.

Artık gömülü sistemi açmanın vakti gelmiştir. Kendi gömülü sistemimizde şu anda sadece 1. ve 2. açılış yükleyicileri vardır. Başka da bir program yoktur. Ayrıca olmayacaktır da. Çünkü çekirdek ve kök dosya sistemi ağ üzerinden yüklenecektir. Bütün bu belge boyunca bu bölüm üzerinde hemen hemen hiç güncelleme yapılmayacaktır.

Artık u-boot seviyesinde açılışa hazır bir gömülü sistemimiz vardır.

2.1.5 U-Boot'un Test Edilmesi

Hangi ortama kurulursa kurulsun, U-Boot kurulduktan hemen sonra, mutlaka açılış testi yapılmalıdır. BBB'nin kendi içinde u-boot gömülü olarak gelmektedir. Bizler de bir u-boot derledik. Sistemin hangi u-boot tarafından açıldığını tespit etmenin en basit yolu, u-boot'u derlerken .config dosyasında PROMPT değişkenini değiştirmektir. Açılışta kendi atadığımız prompt'u görürsek, BBB makinesi derlemiş olduğumuz u-boot tarafından açılmış demektir.

Aynı anda hem u-boot, hem kernel, hem kök dosya sistemi hem de açılış betiği testi yapmak şımarıklıktır.

BBB içinde bulunan u-boot istenirse silinebilir [27]. Bu yöntemde "echo 0 > /sys/block/mmcblk1boot0/force_ro" komutu ile r/o yöntemi ile korunan disk bölümü r/w hale getirilir. Sonra yedeği alınarak içi sıfırlanabilir veya istenilen açılış yükleyicisi bu bölüme dd ile yüklenebilir [28]. Ama yapılması tavsiye edilmez. Tuğla olmuş bir sistem, 9.73'de verildiği gibi kurtarılabilir.

[27] Kaynak: https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/Documentation/mmc/mmc-dev-parts.txt?id=refs/tags/v4.2.4

[28] Kaynak: http://www.jumpnowtek.com/beaglebone/Beaglebone-Black-U-Boot-Notes.html

SD karttan açılış zorlamak için önce S2 butonuna basılır sonra power verilir. S2 butonuna basmadan, doğrudan SD karttan açabilmek için, hem hard hem soft başka teknikler de mevcuttur. Karta bağlı bir özellik olduğundan bu konularla çok da ilgilenmemekteyiz.

eMMC, BBB içinde bulunan ve MMC'yi taklit eden bir donanımdır. Kısaca bu cihaza, fabrika içinde takılmış hazır bir MMC kart gözü ile bakılabilir. Cihaza gömülüdür, değiştirilemez.

Diğer kart ise micro SD kart veya kısaca SD kart dediğimiz cihazdır. Bu cihazı kendimiz takıp çıkarırız. Bütün testlerimiz de SD kart üzerinde olacaktır. SD kartı biz kendimiz bölümlendirdik. Peki eMMC nasıl bölümlendirilmiştir?

eMMC'de fabrika seviyesinde 6 bölüm mevcuttur[29]. Bunların ikisi boot için ayrılmıştır. Diğer 4 adedi genel bölümdür.

Boot bölümlerinde dosya sistemi yoktur. Bundan dolayı mount edilemezler. İkinci boot bölümünde u-boot'un çevre değişkenleri saklanır. Bu değişkenler, sadece bilgi amacıyla aşağdaki gibi listelenebilir.

```
# Aşağıdaki gibi bölüm isimleri incelenebilir.
# Bu komutlar u-boot seviyesinde girilmelidir.

> mmc dev 1
> mmc part

# Aşağıdaki komut BBB tarafında, login geldikten sonra girilir.

$ hexdump -C /dev/mmcblk0boot1 | less
```

eMMC üzerindeki bölümlerin cihaz isimleri aşağıdaki gibi incelenebilir.

```
$  ls -l /dev/mmc*
/dev/mmcblk0           # Cihazın tamamının ismi.
/dev/mmcblk0boot0      # Birinci özel boot bölümü.
/dev/mmcblk0boot1      # u-boot değişkenlerinin saklandığı özel boot bölümü.
/dev/mmcblk0p1         # Genel bölüm.
/dev/mmcblk0p2         # Genel bölüm.
```

[29]Bakınız: http://www.crashcourse.ca/wiki/index.php/EMMC_on_the_BBB

BBB üzerindeki reset düğmesi kullanılırken dikkat edilmelidir. Reset düğmesi en son boot edilen yerden sistemi açar. Power kesilip tekrar verilirse, varsayılan açılış sırasına göre açılış yapılır. Artık en son açılan cihaz dikkate alınmaz.

Bir kereliğine mahsus açılış sırasında S2 butonuna basılır ve SD karttan açılış yapılır. Daha sonraki testlerde reset butonuna basılarak açılış yapılırsa, S2 butonuna basmadan doğrudan SD üzerinden açılış yapılır.

BBB'nin açılış sırası[30] [31] [32] aşağıda verilmiştir. S2 butonuna basılmadığı kabul edilmiştir.

1. MMC1, eMMC
2. MMC0, micro SD
3. UART0
4. USB0

Eğer S2 basılı ise aşağıdaki sırada açılış denenir.

1. SPI0
2. MMC0, micro SD
3. UART0
4. USB0

SPI, UART yani seri kanal ve USB açılışları çok özel açılışlardır. Doğrudan donanıma bağlıdır ve pek nadir kullanılır. Bundan dolayı bu açılışlar üzerinde durulmayacaktır. Her açılış için genelde özel bir MLO programı derlenir. Bu MLO programı, örneğin USB'den açılış yapılacaksa, USB'yi ayağa kaldırır ve u-boot.bin programının yüklenmesi gerekli donanımları hazır hale getirir. Bu örnek "MLO niçin vardır?" sorusunun güzel bir cevabıdır.

[30] boot sequences
[31] Kaynak: http://elinux.org/EBC_Boot_Sequence
[32] Kaynak: http://dumb-looks-free.blogspot.com.tr/2014/05/beaglebone-black-bbb-boot-process.html

Yeni düzeneldiğimiz taze SD kart, BBB'de SD yuvasına takılır, S2 butonuna basılı iken power verilir. İlk mesajlar düşmeye başladığı zaman S2 butonu bırakılır. Konsola ilk mesajlar düşmeden S2 butonu bırakılmaz. Açılış mesajları seri kanal üzerinden izlenebilir.

Artık seri kanala sahip PC kalmadığı için bu iş seri/USB çeviricileri ile yapılmaktadır. Uygun bir kablo BBB'nin seri pinlerine düzgünce takılır. Yanlış takılırsa kablo veya BBB yanar. Google'den "bbb serial console" araması yapılıp uygun kablo temin edilmeli ve düzgün bağlanmalıdır.

Bizler, 3.3 TTL-232R-3V3, FTDI, TTL to USB cinsi bir kablo seçtik. Bu kabloda beyaz tel tx, yeşil tel rx ve siyah tel ground olarak kullanılır. Kırmızı tel ise besleme telidir ve asla bağlanmamlıdır. Her zaman boşta, açıkta durmalıdır.

Seri kanaldan çıkış alabilmek için, siyah tel J1'e takılır. Arada 2 boşluk bırakılır ve sonra yeşil ve beyaz tel sıra ile takılır. Kırmızı tel her zaman boşta kalmalıdır.

Seri konsol çıkışları minicom ile takip edilebilir. Örnek bir minicom komutu aşağıda verilmiştir.

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

Bu komut göz kapalı yazılmamalıdır. Kablo PC'ye takıldıktan sonra "dmesg|tail" komutu ile bağlandığı USB cihazının adı doğru tespit edilmelidir. Bizim örneğimizde bu cihaz adı ttyUSB0'dır. PC donanımına göre bu isim değişiklik gösterebilir. Cihazı takmadan önce ve taktıktan sonra ls -l /dev/ttyUSB* komutu ile de cihazın adı kolaylıkla öğrenilebilir. minicom'un seçenekleri için, bakınız "$ man minicom".

BBB açılırken bir tuşa basılarak u-boot seviyesine inilir. Böylece gömülü sistemin en önemli ayağı tamamlanmıştır. Bundan sonra yapılacak işler donanıma çok daha az bağlıdır. Aynı zamanda hata ayıklaması bundan sonraki işlerde daha kolaydır.

Eğer açılış başarısız olursa, yani u-boot ortamına düşülmezse aşağıdaki hatalar yapılmış olabilir.

1. boot flag'ın açılması unutulmuş olabilir.

2. Bölüm tipi vfat yapılmamış olabilir.
3. vfat, ilk bölüm olmalıdır.
4. MLO, ilk kopyalanmalıdır.
5. Disk geometrisi uygun verilmemiştir.
6. SD kartta bozuk sektörler olabilir, mkfs.vfat'da -c seçeneği ile tekrar dosya sistemi kurulabilir.
7. Çok çalışmış olabilirsiniz, ara vermekte fayda olabilir.

U-Boot ortamına düştükten sonra help ile komut listesi alınıp, komutlarla oynanabilir. Burası çok güzel bir oyun bahçesidir. Her komut ayrı bir eylem grubuna aittir ve sırası geldikçe her gruptan bahsedilecektir.

U-Boot seviyesinde biraz oynadıktan sonra reset ile tekrar açılış yapılır. Bu sefer açılış durdurulmamalıdır. U-Boot kendi açılışını bitirdikten sonra çekirdek arar, bulamaz ve BBB çakılır. Demek ki sonraki aşamada çekirdek derlenmelidir.

2.2 Çekirdek

Aslında Linux kelimesi doğrudan çekirdeği temsil eder. Daha sonra dağıtımların kendisini de temsil eder hale gelmiştir.

Çekirdek kernel.org adresinden indirilebilir. Fakat ARM çekirdekleri genelde biraz geriden gelir ve mutlaka cihaz üreticisinin verdiği adresten indirilmelidir. BBB için çekirdek[33] [34] aşağıdaki gibi indirilebilir.

```
$ git clone git://github.com/beagleboard/linux.git
$ git branch -a
$ git checkout 4.1
```

Çapraz derleme yapabilmek için aşağıdaki tanımların yapılması yeterlidir.

[33]Kaynak: http://wiki.beyondlogic.org/index.php/BeagleBoneBlack_Building_Kernel
[34]Kaynak: https://eewiki.net/display/linuxonarm/BeagleBone+Black#BeagleBoneBlack-LinuxKernel

```
# Çapraz derleyici tanımla.
#
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
```

Çekirdeği derlemek için öncelikle BBB'ye ait bir config dosyasına ihtiyaç vardır. Her gömülü cihazın config dosyası birbirinden farklıdır. Bord satıcıları her zaman kendi cihazlarına uygun config dosyaları kurarlar. Bu dosyalar ARM cihazları için, çekirdeğin kaynak kodu içinde arch/arm/configs/ dizini içinde bulunurlar. "ls -l" ve "more" komutu ile dosyalar incelenebilir.

BBB için config dosyasının ismi bb.org_defconfig adı ile verilmiştir. Bu dosya cp komutu ile kaynak kodunun köküne .config adı ile kopyalanabilir. Ya da aşağıdaki gibi daha formel bir biçimde .config dosyası kurulabilir.

```
$ make -C $KERNEL_SRC O=$KERNEL_OUT bb.org_defconfig
```

Bu komut KERNEL_OUT dizini altında .config dosyası yaratır. Bu dosya son derece basittir ve içinde bazı değişkenlere ait değerlere sahiptir. Bu değerler projemize tam da uygun olmayabilir. Değerlerin güncellenmesi için aşağıdaki gibi menuconfig girişi kullanılır.

```
$ make -C $KERNEL_SRC O=$KERNEL_OUT menuconfig
```

Burada uygun yanıtlar verilerek çekirdek derlemesi yapılır. Burada sadece .config dosyası güncellenir, başka bir iş yapılmaz. Çekirdek derlemesi derya deniz bir konudur. Başlangıçta aşağıdaki hususlara dikkat edilmesi faydalı olacaktır.

1. EXPERIMENTAL ve DEPRICATED özellikler seçilmemelidir.
2. Mümkünse modül kullanılmamalı, bütün özellikler çekirdeğe gömülmelidir.
3. Debugging ve profiling ile ilgili bütün özellikler kapatılmalıdır.
4. Kullanılmayan hiç bir dosya sistemi veya cihaz sürücüsü çekirdeğe dahil edilmemelidir.

5. config.gz çekirdeğe dahil edilmemelidir.
6. vs.

.config dosyası elde edildikten sonra çekirdek derlemesi aşağıdaki gibi yapılır.

```
$ rm -f $KERNEL_OUT/usr/initramfs_data.cpio.gz

$ make -C $KERNEL_SRC O=$KERNEL_OUT uImage LOADADDR=0x80008000 -j2
$ make -C $KERNEL_SRC O=$KERNEL_OUT modules                    -j2
$ make -C $KERNEL_SRC O=$KERNEL_OUT dtbs

$ file $KERNEL_UIMAGE
$ ls -l $KERNEL_UIMAGE
$ mkimage -l $KERNEL_UIMAGE
```

Bu çalışmada kök dosya sistemi çekirdeğe gömülecektir. Fakat henüz elimizde kök dosya sistemi yoktur. Yukarıdaki rm -f komutu sanki elimizde eski bir kök dosya sistemi varmış gibi temizlik yapar. Eski kök dosya sistemini siler.

Sonraki make uImage komutu çekirdeği derler ve zImage dosyasını elde eder. Sonra da zImage dosyasının tepesine 64 baytlık u-boot bilgisini yerleştirerek zImage dosyasını u-boot imajı haline getirir. Tepesinde 64 baytlık bilgi bulunan bütün dosyalara u-boot imajı denir ve genelde "u" harfi ile başlar.

make uImage sonucu aşağıda verilmiştir.

```
...
Kernel: arch/arm/boot/zImage is ready
    UIMAGE  arch/arm/boot/uImage
    Image Name:   Linux-4.1.18
    Created:      Sun Apr 10 08:54:50 2016
    Image Type:   ARM Linux Kernel Image (uncompressed)
    Data Size:    5400984 Bytes = 5274.40 kB = 5.15 MB
    Load Address: 80008000
    Entry Point:  80008000
      Image arch/arm/boot/uImage is ready
      make[1]: Leaving directory '/opt/gomsis/distro/initramfs-I/out/linux'
```

Sonraki make modules komutu, varsa modülleri derler. Modüller cihaz sürücüleridir ve çekirdekten bağımsız, ayrı bir dosya olarak bulunurlar. Yürütme zamanında yüklenip, kaldırılabilirler. Çekirdeğe inanılmaz esneklik kazandırırlar. Modüller her zaman .ko uzantısı ile biterler.

Modül derlemesi sırasında ekran çıkışı izlenmeli ve kazara işaretlenmiş olan modüller, tekrar menuconfig yapılarak çekirdekten atılmalıdır.

make help komutu ile çekirdek derlemesine ait pek çok giriş elde edilebilir. Örneğin make clean ile, .config hariç, derleme sırasında elde edilen bütün dosyalar silinir.

Menuconfig işlemleri, make xconfig ile de Qt tabanlı yapılabilir. Başka tipten menülerle de config işlemi gerçeklenebilir. Çok yaygın ve basit olduğu için bizler menuconfig girişini kullanmaktayız.

Ayrıca bütün seçimlere yes vermek veya no vermek için allyesconfig veya allno config gibi çok fazla make seçeneği, make girişi mevcuttur. Katılımcı mutlaka bunları tek tek incelemelidir.

tinyconfig özelliği çok ilginçtir. En ufak çekirdeği hazırlar. Muhtemelen bu çekirdek cihazları açmaya yetmez. Genelde tinyconfig ile çekirdek elde edildikten sonra, menuconfig ile gerekli özellikler çekirdeğe eklenir. Bu işlem çok fazla zaman alsa da, son derece küçük boyutlu bir çekirdek elde edilir.

Ayrıca destek verilen cihazlara ait defconfig[35] listesi de "$ make ARCH=arm help" ile elde edilebilir.

Bord üzerindeki donanımın özellikleri DTB denilen bir dosyada tutulur. DTB, device tree blob demektir ve çekirdek bu dosyayı okuyarak donanım hakkında bilgi sahibi olur. Bu dosya doğrudan çekirdek içine gömülebileceği gibi, açılış sırasında, ayrı bir dosya olarak da çekirdeğe verilebilir. Böylece çekirdek, borda ait pek çok donanım bilgisine, örneğin CPU sayısı, RAM'ın başlangıç adresi ve büyükülüğü, bağlı cihazlar, IRQ bilgileri gibi doğrudan donanıma bağlı bilgileri açılış zamanında elde edebilmektedir.

Yukarıda verilen make dtbs komutu işte bu DTB dosyalarını derler ve çekirdeğin anlayabileceği bir biçime getirir. DTB dosyaları ASCII formatında, hiyerarşik bir yapıdadır ve editör ile her zaman güncellenebilir.

-j ile host makinedeki CPU sayısı verilir. cat /proc/cpuinfo komutu ile, CPU sayısı tam olarak öğrenilebilir. -j ile aynı anda paralel çalışacak iş sayısı verilir. Mevcut CPU sayısından az veya fazla değer vermek derleme süresini uzatacaktır.

Derleme bittikten sonra file, ls ve mkimage komutları ile zImage ve uImage

[35]Firmalar tarafından önceden hazırlanmış .config dosyaları

imajları incelenmelidir. Bir inceleme sonucu aşağıda verilmiştir.

```
$ make -C $KERNEL_SRC O=$KERNEL_OUT menuconfig

 uImage için örnek Çıkış:
 ...
 Kernel: arch/arm/boot/zImage is ready
   UIMAGE  arch/arm/boot/uImage
   Image Name:   Linux-4.1.13
   Created:      Fri Nov 20 09:49:47 2015
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    5446336 Bytes = 5318.69 kB = 5.19 MB
   Load Address: 80008000
   Entry Point:  80008000
     Image arch/arm/boot/uImage is ready

 file çıkışı:
 uImage: u-boot legacy uImage,
         Linux-4.1.13,
         Linux/ARM,
         OS Kernel Image (Not compressed),
         5446336 bytes,
         Fri Nov 20 09:49:47 2015,
         Load Address: 0x80008000,
         Entry Point: 0x80008000,
         Header CRC: 0x0A4C126E,
         Data CRC: 0xBA080825

 ls -l çıkışı:
 -rw-rw-r-- 1 nazim nazim 5446400 Kas 20 09:49 out/linux/arch/arm/boot/uImage
```

Çekirdek elde edilmiştir. Şimdi bu çekirdek ne olacaktır, yenir mi içilir mi?

Öncelikle çekirdek tek başına yeterli değildir. Çekirdek derlendiğinde genelde aşağıdaki dosyalar üretilir.

uImage Tepesine 64 bayt eklenmiş zImage dosyasıdır.

DTB Device Tree Blob dosyası.

***.ko dosyaları** Çekirdek modülleri.

fw dosyaları Cihazlar için firmware dosyaları.

uImage hariç diğer bütün dosyalar hiç olmayabilir ya da mevcut olabilirler ama ayrı olarak değil doğrudan kernel içine gömülü olabilirler.

uImage ve DTB dosyası, her zaman u-boot tarafında yüklenir. *.ko ve fw dosyaları ise, her ne kadar çekirdeğe ait olsalar da kök dosya sistemi içinde bir yerlerde otururlar. Özel bir dizin yapısına sahiptirler. *.ko ve fw dosyaları aşağıdaki gibi uygun bir dizine kurulurlar. Uygun dizin, ROOT_FS değişkeni ile temsil edilmiştir.

```
$ make -C $KERNEL_SRC O=$KERNEL_OUT INSTALL_MOD_PATH=$ROOT_FS modules_install
$ make -C $KERNEL_SRC O=$KERNEL_OUT firmware_install
```

Firmware dosyaları INSTALL_FW_PATH ile verilen dizine yüklenir. Bu dizin verilmezse, firmware dosyaları $(INSTALL_MOD_PATH)/lib/firmware altına yüklenir[36].

Şimdilik ROOT_FS ile verilen dizinin içinde ne olduğu ile hiç ilgilenmiyoruz. make modules_install yaptıktan sonra dizin içeriği mutlaka incelenmelidir.

*.ko ve fw dosyalarını ROOT_FS'e misafir ettik. Şimdilik bu dosyaları cepte kabul edelim. Sonra kullanacağız.

Örneğimizde, firmware_install ile bütün firmware dosyaları `/lib/firmware` altında kurulur. Çekirdekte son derece fazla firmware dosyası bulunur. İhtiyaç olunan dosyalar hariç diğer bütün dosyaların silinmesi tavsiye edilir.

Ya da firmware_install komutu hiç kullanılmamalı, firmware dosyasının adı biliniyorsa, find ile KERNEL_OUT dizini altından bulunup, doğrudan `/lib/firmware` altında kopyalanmalıdır.

Asıl işimiz olan çekirdek tarafına dönecek olursak, elimizde kullanıma hazır 2 dosya vardır, uImage ve bir adet dtb dosyası. Bu dosyaların oturduğu yer gömülü sistemin bir karakterisitiğidir. Örnek sistemde bu dosyalar uzaktaki bir makinede olacaktır. Yani BBB üzerinde olmayacaktır. Şu anda BBB üzerinde sadece u-boot vardır.

BBB'nin u-boot ve dtb dosyasını alabilmesi için uzaktaki makinede bilindik bir yere yüklemesi gerekir. Genelde, uzaktaki makinede bulunan /tftpboot dizini bu iş için kullanılır. Çünkü tftp sunucusunun varsayılan kök dizini

[36]Kernel kaynak kodunda iken $ make help giriniz.

/tftpboot'tur. Aşağıdaki gibi çekirdek imajı ve ona eşlik eden dtb dosyası tftpboot altına atılır.

```
$ cp $KERNEL_UIMAGE /tftpboot
$ cp $KERNEL_OUT/arch/arm/boot/dts/am335x-boneblack.dtb \
              /tftpboot/am335x-boneblack.dtb
```

Artık herhangi bir BBB makinesi uzakta bulunan çekirdeği ve dtb dosyasını kullanarak kendini açabilir. Test başlığına geçmeden önce firmware ve XIP kavramları üzerinde iki lafın belini kıralım.

Firmware Kavramı

firmware[37] nedir? Kernel tarafından cihazlara yüklenen programlara firmware denir. Çalışma mantığı çok basittir.

Bir cihaz sürücüsü çekirdekten bir firmware dosyası talep eder. Firmware dosyası, genelde arı C kodu ile yazılmış ve doğrudan cihaz içinde çalışan bilgisayar programıdır. Eğer udev sistemi mevcutsa, çekirdek udev'e dosyanın olup olmadığını sorar. udev programı da bir betik yardımı ile çekirdeğe dosyayı gönderir. Çekirdek de dosyayı cihaz sürücüsüne verir. Cihaz sürücüsü de ilgili dosyayı cihaza yükler.

Çekirdeğin açılış mesajlarında bu işlemler "firmware: requesting ..." ve "loaded firmware ..." şeklinde belirir.

Execute-In-Place veya XIP

Execute-In-Place veya kısaca XIP tekniğinde, çekirdek RAM'a yüklendikten sonra başka bir adrese taşınmaya gerek kalmadan olduğu yerde işletilir. Böylece taşıma için geçen süre kazanılmış olur. XIP kavramı Şekil 2.2'de temsil edilmiştir.

XIP tekniği için adres hesabı çok basittir. bootm komutunda, çekirdeğin yükleme adresi olarak, mkimage adresindeki yükleme adresinin 64 bayt eksiği verilir.

[37] Türk Bilişim Derneği, TBD tarafından bellenim diye çevrilmiş. Donanım yazılımı olarak da geçiyor.

```
            u-Boot image: uImage
80007FC0    +------------+
            | 64-byte    |u-boot header, 64-byte veya 0x40-byte
80008000    +------------+
            |            |Compressed Linux kernel
            | zImage     |
            |            |
            +------------+

Load Address: 80008000  ( mkimage komutunda -a ile verilen )

> bootm 0x80007FC0 ( u-boot tarafında verilen yükleme adresi )
```

Şekil 2.2: Yerinde Yürütme

2.2.1 Çekirdeğin Test Edilmesi

Bu aşamada elimizde kök dosya sistmi ve açılış betikleri yoktur. Bunun yerine SD karta yüklü bir açılış yükleyicisi ve uzaktaki bir makinede yüklü çekirdek ve ona eşlik eden DTB dosyası vardır. Her adımda ayrı test edilmeli mantığı ile, bu adımda u-boot ve çekirdek test edilecektir. Tabii ki kök dosya sistemi olmadığı için BBB tıkanacaktır.

Testi başlatabilmek için PC tarafında tftp sunucusu yüklü ve ayakta olmalıdır[38]. Bu sunucunun config bilgileri /etc/xinetd.d/tftp içindedir. Örnek bir config dosyası aşağıda verilmiştir.

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

server_args ile verilen değerin /tftpboot olduğuna dikkat edilmelidir.

[38]Bakınız http://askubuntu.com/questions/201505/how-do-i-install-and-run-a-tftp-server

BBB'ye güç verilir ve açılış, boşluk tuşuna basılarak durdurulur, u-boot promptuna düşülür. Aşağıdaki komutlar ile çekirdek ve dtb'nin yüklenip çekirdeğin ayağa kalkması sağlanır.

```
Hit any key to stop autoboot:  0

=> setenv ipaddr 192.168.1.100              # BBB'nin IP adresi.
=> setenv serverip 192.168.1.21             # tftp'nin IP adresi.
=> ping 192.168.1.21                        # erişim var mı?
=> tftpboot 0x80F80000 am335x-boneblack.dtb # dtb'yi boş bir adrese yükle.
=> tftpboot 0x80007FC0 uImage               # çekirdeği, XIP olacak şekilde yükle.
=> setenv bootargs console=ttyO0,115200n8   # çekirdeğe geçecek argümanları tanımla.
=> bootm 0x80007FC0 - 0x80F80000            # boot from memory, bootm.
```

ipaddr ve serverip komutları ile kendi ve uzaktaki makinenin IP bilgileri u-boot'a verilir. ping ile erişimin akibeti sorgulanır.

"tftpboot 0x80F80000 am335x-boneblack.dtb" komutu ile uzaktaki makineden 0x80F8000 adresine dtb dosyası yüklenir. Bu adresin bir özelliği yoktur. Sadece adreslerdeki veriler birbirlerini ezmemelidir. dtb için genelde 0x80000000 adresinden çok uzakta, diplerde bir adres verilir.

Adres bilgileri u-boot seviyesinde iken bdinfo komutu ile elde edilebilir. BBB için örnek çıkış aşağıda verilmiştir.

```
=> bdinfo
arch_number = 0x00000E05
boot_params = 0x80000100   <-- çekirdeğe parametre aktarılacak adresin başlangıcı.
DRAM bank   = 0x00000000
-> start    = 0x80000000   <-- adresleme buradan başlar.
-> size     = 0x20000000   <-- adres uzayının genişliği.
eth0name    = cpsw
ethaddr     = 1c:ba:8c:95:ba:bc
current eth = cpsw
ip_addr     = <NULL>
baudrate    = 115200 bps
TLB addr    = 0x9FFF0000
relocaddr   = 0x9FF60000
reloc off   = 0x1F760000
irq_sp      = 0x9EF3FEC0
sp start    = 0x9EF3FEB0
=>
```

"tftpboot 0x80007FC0 uImage" komutu ile çekirdek, 0x80007FC0 adresine yüklenir.

Daha önce çekirdeği derlerken aşağıdaki gibi bir komut girmiştik.

```
$ make -C $KERNEL_SRC O=$KERNEL_OUT uImage LOADADDR=0x80008000 -j2
```

Buradaki LOADADDR ifadesi, çekirdeğin işletilmeden evvel, bu adrese taşınmasını u-boot'a söyler. Diğer bir deyişle, tftp komutu ile çekirdek nereye yüklenirse yüklensin, işletilmeden evvel u-boot tarafında LOADADDR ile verilen yere kopyalanacaktır. Aslında bu taşıma çok gereksizdir. Taşıma yapmadan da bu işlemi yapabilir miyiz?

Tabii ki çok kolay yapılabilir. tftp adresi olarak LOADADDR adresinin 64 eksiği olan adres, 0x80007FC0 verilir. "tftpboot 0x80007FC0 uImage" ifadesinde uImage çekirdeği 0x800007FC0 adresine yüklenir. u-boot, çekirdeği işletmeden evvel LOADADDR ile verilen 0x80008000 adresine taşımalıdır. Fakat yaptığımız adres hesabından dolayı 0x80008000 adresine zaten uImage vardır. Sonuçta taşıma yapılmaz. Çekirdek bulunduğu yerde işletilir, yütürülür. Daha önce bahsettiğimiz gibi buna yerine yürütme veya XIP tekniği denir.

"setenv bootargs console=ttyO0,115200n8" komutu ile u-boot hiç ilgilenmez. Burada bilgileri gözü kapalı olarak 0x80000100 adresine yazar. Çekirdek de açılırken bu adresteki bilgileri okur. Tahmin edilebileceği gibi bu bilgiler çekirdeğe geçirilmek istenen "çekirdek parametreleri"dir.

"bootm 0x80007FC0 - 0x80F80000" komutu ile yürütme 0x80007FC0 adresine geçirilir. Bu adres de bizim çekirdeği yüklediğimiz adrestir.

bootm komutu 3 argüman alır. İlk argüman çekirdeğin bellekteki başlangıç adresidir ki bu adres tftp ile yüklenirken verilmiştir. İkinci argüman ise initramfs ayrık bir dosya iken, initramfs'in başlangıç adresidir. Bu örnek sistemde initramfs doğrudan çekirdeğe gömülüdür. Bundan dolayı bu adres - ile es geçilmiştir.

Son adres ise dtb dosyasının bellek adresidir. u-boot bu adres bilgisini çekirdeğe geçer. Böylece çekirdek açılırken gerekli parametreleri 0x80000100 adresinden elde eder. dtb ağacının adresini ise u-boot doğrudan çekirdeğe verir.

Çekirdek bu şekilde açılacaktır. Tabii ki henüz kök dosya sistemi mevcut olmadığı için BBB tıkacaktır.

2.3 Kök Dosya Sistemi

Çekirdeğin açılış sırasında bağladığı dosya sistemine kök dosya sistemi[39] denir. Kök dosya sistemi birkaç farklı biçimde hazırlanır.

1. Mevcut bir kök dosya sistemi bir yerlerden alınır ve üzerinde el yordamı ile güncelleme yapılır. Zahmetsiz bir yöntemdir. Tembel hayvan veya yaprak sevenlere[40] tavsiye edilir.

2. Busybox ile kök dosya sistemi kurulabilir. Busybox içinde yeterli komut seti veya paket mevcut değildir. Fakat yine de pek çok gömülü sistem için yeterli bir kök dosya sistemi üretir. Bu bölümde busybox ile kök dosya sistemi kurulacaktır.

3. Build Root[41] ile kök dosya sistemi kurulabilir. X arabirimi dahil bütün ihtiyaçlara cevap verebilir. Çok geneldir. Fakat kurmak için peygamber sabrı gerektirir. Kuruluş ortamı pek de pratik değildir. Yatmadan önce kuruluşa başlanır, şansınız varsa sabaha bitmiş olur. Çok zahmetli olduğu için sadece tek bir gömülü sistem örneği BR ile verilecektir.

4. Debian, Slackware gibi standard dağıtımların BBB için veya ARM makineler için kök dosya sistemleri mevcuttur. Bizler bu sistemleri gömülü sistem saymadığımız için ilgilenmiyoruz. Ayrıca Yocto gibi bazı projelerde de kök dosya sistemi kurulmaktadır. Fakat bu tür projeleri gömülü sistemlere standard makineler gibi yaklaştığından pek de ciddiye almıyoruz. Ama başkaları ciddiye alıyor ki Yocto projesi almış başını gitmektedir. Hayırlısı.

Bu örnekte kök dosya sistemi İsviçre Çakısı olarak da bilinen busybox paketi ile kurulacaktır. Busybox mantığında, pek çok Linux komutunun seçenekleri azaltılmış ve hepsi tek bir dosya içine tıkışıtılmıştır. Busybox aşağıdaki mantık ile derlenebilir ve kurulabilir.

Çapraz derleme için CROSS_COMPILE tanımının yapılması yeterlidir. Bu tanım doğrudan menuconfig sırasında da yazılabilir.

[39] root file system

[40] Yaprak sevenler ya da tembel hayvanlar, tüm memeliler arasında en yavaş hareket eden hayvanlar olarak bilinirler. Dakikada en fazla yarım metre kadar hareket ettiği hesaplanmıştır. Tembel hayvanlar, günde 15 ila 18 saat uyuyarak en çok uyuyan hayvanların başında gelirler. Kaynak: https://tr.wikipedia.org/wiki/Tembel_hayvan

[41] Kısaca BR yazılacaktır.

```
# Kaynak kodu indir.
# Müşteri sistemi her zaman kararlı sürüm olmalıdır.
#
$ git clone git://busybox.net/busybox.git
$ git branch -a
$ git checkout remotes/origin/1_NN_stable ya da master'da kal.
$ git branch

# Çapraz derleyici tanımla.
#
export CROSS_COMPILE=arm-linux-gnueabihf-

# Seçim yap.
#
make -C $BUSYBOX_SRC O=$BUSYBOX_OUT clean
make -C $BUSYBOX_SRC O=$BUSYBOX_OUT menuconfig

# Derle.
#
make -C $BUSYBOX_SRC O=$BUSYBOX_OUT -j2

# Sonuçları kontrol et.
#
cd $BUSYBOX_OUT
ls -l busybox
file busybox

# Kuruluş yap.
# $BUSYBOX_OUT/_install altına temel kuruluşu yapar.
#
rm -fr $BUSYBOX_OUT/_install
make -C $BUSYBOX_SRC O=$BUSYBOX_OUT install
```

Çapraz derleyicilere ait kütüphanelerin yeri aşağıdaki gibi, çapraz gcc komutu ve print-file-name seçeneğ ile elde edilebilir.

```
$ arm-linux-gnueabihf-gcc -print-file-name=libc.so.6
/opt/gomsis/toolchain/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/ \
    bin/../arm-linux-gnueabihf/libc/lib/arm-linux-gnueabihf/libc.so.6
```

Kuruluş tamamlandıktan sonra mutlaka _install dizini altına gidip, busybox'ın ne halt yediği incelenmelidir. Burada busybox sisteminin bütün felsefesi kolaylıkla farkedilebilir. Nedir bu felsefe? Adı busybox olan, küçük boyutlu, tek bir binary dosya ve bu dosyaya atıfta bulunan pek çok sembolik link.

Busybox'ın temel amacı komutların kapladığı alanı azaltmaktır. Bu işi nasıl başarılı yaptığı aşağıdaki inceleme ile görülebilir.

```
$ ls -l /usr/bin/vim.gnome
$ ls -l busybox

$ file busybox
$ file busybox_unstripped
$ ls -l _install
```

Busybox, prensip olarak Linux komutlarını temin eder. Bunun dışında kök dosya sistemi için daha başka bir katıkda bulunmaz. Bir kök dosya sisteminde komut dışında /proc, /etc/, /tmp gibi dizinlere ve içi dolu bir /etc dizinine gerek vardır. Busybox dört başı mağrur bir kök dosya sistemi sunmaz. Bu eksiklikler el yordamı ile giderilir. Genelde bu eksiklikleri tamamlamak için iskelet bir kök dosya sistemi kullanılır. Örnek bir iskelet kök dosya sisteminin yapısı aşağıda verilmiştir.

```
$ ls -l RootFS.skel/
total 36
drwxrwxr-x 2 nazim nazim 4096 Ara  4 11:47 dev
drwxrwxr-x 2 nazim nazim 4096 Ara  6 18:11 etc
lrwxrwxrwx 1 nazim nazim   10 Tem 25  2015 init -> /sbin/init
drwxrwxr-x 2 nazim nazim 4096 Ara  4 11:37 lib
drwxrwxr-x 2 nazim nazim 4096 Ara  4 11:47 proc
drwx------ 2 nazim nazim 4096 Ara  4 11:47 root
drwxrwxr-x 2 nazim nazim 4096 Ara  4 11:47 sys
drwxrwxr-x 2 nazim nazim 4096 Ara  4 11:47 tmp
drwxrwxr-x 3 nazim nazim 4096 Ara  4 11:47 usr
drwxrwxr-x 2 nazim nazim 4096 Ara  4 11:47 var
```

İskelet kök dosya sisteminin hazır kurulu bir hali build root projesinden de elde edilebilir. Sayfa 114'de verildiği gibi build root indirilebilir. Örnek bir iskelet kök dosya sistemi, kaynak kodunda, system/skeleton altında bulunabilir.

Kernel, initramfs tipindeki kök dosya sistemini yükler yüklemez, gözü kapalı bir biçimde /init programını çalıştırır. init programı, betik dahil çalışabilir herhangi bir program olabilir. Örnekte init programı standard /sbin/init'i çalıştırmaktadır.

Yukarıdaki örnekte görülen iskelet sisteminin bütün dizinleri, /etc hariç, boştur. /etc dizinin içeriği aşağıda listelenmiştir.

```
$ ls -l etc
total 120
-rw-rw-r-- 1 nazim nazim    28 Tem 25  2015 adjtime
-rw-rw-r-- 1 nazim nazim    29 Tem 25  2015 exports
-rw-rw-r-- 1 nazim nazim   114 Ara  4 17:36 fstab
-rw-rw-r-- 1 nazim nazim    97 Tem 25  2015 group
-rw-rw-r-- 1 nazim nazim    24 Tem 25  2015 gshadow
-rw-rw-r-- 1 nazim nazim    40 Tem 25  2015 hosts
-rw-rw-r-- 1 nazim nazim    39 Tem 25  2015 inetd.conf
-rw-rw-r-- 1 nazim nazim   113 Ara  4 20:29 inittab
-rw-rw-r-- 1 nazim nazim   304 Tem 25  2015 issue
-rw-rw-r-- 1 nazim nazim     1 Tem 25  2015 ld.so.cache
-rw-rw-r-- 1 nazim nazim     1 Tem 25  2015 ld.so.conf
lrwxrwxrwx 1 nazim nazim    12 Tem 25  2015 mtab -> /proc/mounts
-rw-rw-r-- 1 nazim nazim   154 Tem 25  2015 nsswitch.conf
-rw-rw-r-- 1 nazim nazim   315 Tem 25  2015 passwd
-rwxrwxr-x 1 nazim nazim   176 Tem 25  2015 profile
-rw-rw-r-- 1 nazim nazim  1623 Tem 25  2015 protocols
-rwxrwxr-x 1 nazim nazim   501 Ara  6 18:11 rcS
-rw-rw-r-- 1 nazim nazim    24 Ara  6 18:11 resolv.conf
-rw-rw-r-- 1 nazim nazim  1615 Tem 25  2015 rpc
-rw-rw-r-- 1 nazim nazim    80 Tem 25  2015 securetty
-rw-rw-r-- 1 nazim nazim 29366 Tem 25  2015 services
-rw-rw-r-- 1 nazim nazim   251 Tem 25  2015 shadow
-rw-rw-r-- 1 nazim nazim    61 Tem 25  2015 sudoers
-rw-rw-r-- 1 nazim nazim     5 Tem 25  2015 TZ
```

Bu dosyaların hemen hemen tamamı PC'deki Ubuntu sisteminde kopyalanmıştır. Her dosyanın özel bir yeri ve anlamı vardır. En önemli dosya /etc/issue dosyasıdır, açılışta reklamımızı yapar :)

Buybox'ın sağlamış olduğu komutlar, iskelet dosya sisteminde /bin, /sbin ve /usr/bin dizinlerine kopyalanırsa elimizde açılışa hazır bir kök dosya sistemi olacaktır.

İskelet kök dosya sistemi RootFS.skel ile ve amaç kök dosya sistemi de ROOT_FS ile temsil edilmiş olsun. Nihayetinde amaç kök dosya sistemi "ROOT_FS= RootFS.skel + Busybox Komutları + LibC kütüphanesi" şeklinde elde edilebilir.

RootFS.skel dizinini zaten elimizle oluşturmuştuk. Busybox derlemesi ile de komutları oluşturduk. Dinamik kütüphane kullandığımız için busybox programı bu kütüphanelere ihtiyaç duyacaktır. Kütüphaneler her zaman çapraz derleyici içinden elde edilir. Bütün bu işlemler aşağıda verilmiştir.

```
# Iskeleti gözü kapalı amaç kök dosya sistemine kopyala.
#
$ cp -a RootFS.skel/* $ROOT_FS/

# Busybox'ın ürettiği dosya ve dizinleri taşı.
#
cp -a $BUSYBOX_OUT/_install/* $ROOT_FS

# Çapraz derleyicinin oturduğu yeri SYSROOT içinde sakla.
#
SYSROOT=‘$PROJECT_HOME/toolchain/arm/bin/arm-linux-gnueabihf-gcc -print-sysroot‘

# 32 bit toolchain için:
# Bütün dinamik kütüphanaleri kopyala.
# Aslında sadece üç kütüphane gereklidir.
# libc, libm ve ld.
# Diğerleri israftır.
#
cp -a $SYSROOT/lib/ld-linux-armhf.so.3 $ROOT_FS/lib
cp -a $SYSROOT/lib/arm-linux-gnueabihf $ROOT_FS/lib/

# 64 bit toolchain için:
# Sadece gerekli kütüphaneleri kopyala.
# Burada tutumluyuz.
#
cp -a $SYSROOT/lib/ld-linux-armhf.so.3 $ROOT_FS/lib/
cp -a $SYSROOT/usr/lib/libc.so.6       $ROOT_FS/usr/lib/
cp -a $SYSROOT/usr/lib/libm.so.6       $ROOT_FS/usr/lib/

# Gereksiz dosyaları sil.
#
rm -f $ROOT_FS/linuxrc

# Kök dosya sisteminin içeriğine ve büyüklüğüne bak.
#
ls -l $ROOT_FS
du -ks $ROOT_FS
```

Gömülü sistem kernel seviyesinde açılır ama init'i çalıştıramazsa muhtemelen init'in bağlı olduğu kütüphaneler yüklenememiştir. /sbin/init veya kısaca init programı için, busybox kullanılmıştır. Bu durumda init ve diğer bütün programlar busybox'ın bağlı olduğu kütüphanelere bağımlıdır.

Bir programın bağlı olduğu kütüphaneler native durumda ldd veya cross durumunda arm-linux-gnueabihf-ldd komutu ile tespit edilebilir. Fakat arm tarafında genelde ldd komutu bulunmaz. Bu durumda dump ile de benzer bilgi elde edilebilir. Aşağıdaki örnekte busybox'ın bağlı olduğu kütüphaneler,

PC tarafında girilecek dump komutu ile listelenebilir.

```
$ arm-linux-gnueabihf-objdump -x busybox  | grep NEEDED
  NEEDED               libm.so.6
  NEEDED               libc.so.6
```

Bu durumda busybox için libm ve libc kütüphaneleri yeterlidir. Her ne kadar burada gözükmese de her program için dinamik kütüphane yükleyicisi veya bir tür yorumlayıcıya ihtiyaç vardır. Bunun ismi arm makineler için ld-linux-armhf ile verilir. Her mimaride bu isim farklı olabilir.

Yapılan bu analize göre gerekli kütüphaneler toolchain içinden RootFS içine kopyalanır. Burada çok büyük bir incelik vardır. Kütüphaneler kopyalanırken toolchain içinde nerede otururlarsa aynı biçimde kopyalanmalıdır.

Örneğin BBB için kullandığımız 32 bir toolchain'de libc.so.6 kütüphanesi SYSROOT ile gösterilen dizinde, lib/arm-linux-gnueabi/ altında iken, aynı toolchain'in 64 bitlik sürümünde usr/lib altındadır. Kütüphane toolchain'de nerede bulunuyorsa, RootFS'de de aynı yerde olmalıdır.

Bunun sebebi çok basittir. Toolchain derleme yaparken kütüphanelerin bulunduğu yer olarak toolchain içinde bulunan konumu gcc'ye tanımlar. Derlenen kod da aynı yerde kütüphane arar. Tabii ki daha sonra LD_LIBRARY_PATH ile kütüphanelerin oturduğu yerler değiştirilebilir. Fakat kütüphaneleri toolchain ile aynı hiyerarşide tutmak çok daha pratiktir.

Bu arada ldd komutu kütüphanelerin bağlı olduğu kütüphanelerin de listesini verir. Fakat dump komutu sadece ilk seviye bağımlılıkları verir. dump komutu daha sonra kütüphaneler için işletilerek diğer bağımlılıklar bulunabilir. Aslında ldd komutu da aynı işi yapar. Bu komut genelde bash programıdır ve arka arkada dump'ı çalıştırarak bütün kütüphaneleri listeler.

Artık elimizde dört başı mağrur bir kök dosya sistemi vardır. Özetleyecek olursak, bu kök dosya sistemi aşağıdaki temel parçalardan oluşur ve bütün gömülü sistemlerde hemen hemen aynı mantıkla ama farklı tekniklerle kurulur. Örneğin BR sisteminde, kuruluş bittikten sonra benzer kök dosya sistemi elde edilir. Fakat işlemlerin tamamı arka planda yapıldığı için daha az emek verilir. Katılımcı buradaki kök dosya sisteminin genel özelliklerini mutlaka çok iyi kavramış olmalıdır. Kök dosya sisteminin temel parçaları aşağıda verilmiştir.

1. Önce içleri tamamen boş, /bin, /sbin, /usr, /tmp, /proc, /etc vs gibi dosyalar yaratılır.
2. Busybox tarafından elde edilen komutlar /bin, /sbin ve /usr/bin içine kopyalanır.
3. /etc içine gerekli config dosyaları kopyalanır.
4. /init dosyası tanımlanır ki çekirdek bu dosyayı, initramfs yüklenir yüklemez çalıştırsın.
5. Çekirdek modülleri varsa /lib/modules altına kopyalanır.
6. Firmware varsa /lib/firmware altına kopyalanır.
7. Dinamik kütüphaneler, çapraz derleyiciden elde edilir ve /lib altında kopyalanır.
8. /etc/rcS gibi bir açılış betiği yazılır. Bu betik sistemi login'e kadar getirir.

ROOT_FS dizini altında bulunan kök dosya sistemi artık pek çok teknikle çekirdek tarafından yükelenebilir. Biz bu örnek sistemde, kök dosya sistemini doğrudan çekirdek içine atacağız. Böylece çekirdeği yüklediğimizde kök dosya sistemi de ona yapışık olarak gelecektir.

Kök dosya sistemi ROOT_FS dizinde verildiği gibi çekirdek tarafından kullanılamaz. Bu dosya sistemini çekirdeğin anlayacağı hale getirmek gerekir. Bunun için birkaç yol vardır. Bu bölümdeki örnek Linux sisteminde, ROOT_FS dizinini çekirdek derlemesi ile çekrideğe ekleyeceğiz. 2 farklı teknik daha vardır ve bunlar ileriki bölümlerde işlenecektir.

Örnek kök dosya sistemi, yani ROOT_FS dizininde bulunan kök dosya sistemi aşağıdaki şeklide çekirdek derlemesi ile çekirdeğe yapıştırılır.

```
$ rm -f $KERNEL_OUT/usr/initramfs_data.cpio.gz
$ make -C $KERNEL_SRC O=$KERNEL_OUT menuconfig
$ make -C $KERNEL_SRC O=$KERNEL_OUT uImage LOADADDR=0x80008000 -j2
$ ls -l $KERNEL_UIMAGE
```

Çekirdek, derleme sırasında, ROOT_FS'i cpio arşivi haline getirir ve çekirdek çıkışının kökünde, /usr dizini altında initramfs_data.cpio.gz adı ile saklar.

ROOT_FS'i çekirdeğe yapıştırmadan veya gömmeden önce bu dosya mutlaka silinmelidir. Eğer silinmezse, "dosya zaten var" denilerek ROOT_FS'de yapılan güncellemer çekirdeğe eklenmez.

Sonraki "$ make menuconfig" komutu ile config soruları ekrana gelir. ROOT_FS'in çekirdeğe eklenebilmesi için aşağıdaki gibi girişler ve seçimler yapılmalıdır.

```
General setup --->
[*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
      (/gomsis/boards/bbb/RootFS) Initramfs source file(s)
(1000)  User ID to map to 0 (user root)
(1000)  Group ID to map to 0 (group root)
Built-in initramfs compression mode (Gzip)  --->
```

Sonra "make uImage" komutu ile derleme yapılır. Derleme çıkışında aşağıdaki gibi bir satır gözlenmelidir.

```
   GEN     usr/initramfs_data.cpio.gz
```

Bu bilgi satırı, ROOT_FS ile verilen kök dosya sisteminin, cpio arşivi haline getirildiğini tespit eder. Derleme işi bittikten sonra aşağıdaki gibi bir çıkış elde edilir.

```
Image Name:   Linux-3.8.13-UL23052014-00744-g5
Created:      Sat May 24 09:38:15 2014
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    4765360 Bytes = 4653.67 kB = 4.54 MB  <-- Boy büyüdü.
Load Address: 80008000
Entry Point:  80008000
```

"Data Size" ile gösterilen çekirdek boyunu büyüdüğü açıkça gözlemlenebilir. Buradan da hemen tahmin edeleceği gibi, kök dosya sistemi veya buradaki kullanım tekniğine göre initramfs sistemi, çok büyükse bu yöntem verimli olmayacaktır. Çünkü kök dosya sistemi bellekte büyük yer işgal edecektir.

Şimdi elimizde bir çekirdek ve ve içine initramfs tekniğine göre gömülmüş bir kök dosya sistemi arşivi vardır. Çekirdek /tftpboot/ altında uImage ismi ile kopyalanarak kullanıma hazır hale getirilir.

Sonraki bölümde açılış betiği ile birlikte login'e kadar gelinecektir.

2.4 Açılış Betiği

/etc/rcS ismi ile aşağıdaki gibi çok basit bir açılış betiği yazılmıştır. İleriki bölümlerde daha kullanışlı betiklerden bahsedilecektir. Fakat mantık hiç ama hiç değişmemektedir. /sbin/init programı göz kapalı bir biçimde /etc/rcS betiğini işletir.

/sbin/init'in çalışma mantığını anlamak için /etc/inittab dosyası incelenmelidir. Bu dosyanın bir örneği aşağıda verilmiştir.

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

Bu dosya standard bir inittab dosyası değildir. Busybox için feci biçimde sadeleştirilmiştir. /sbin/init programı çalışır çalışmaz, /etc/inittab içindeki sysinit satırını arar ve karşısında bulunan programı işletir. Diğer satırların anlamları için busybox belgeleri incelenmelidir.

Örnek bir /etc/rcS betiği aşağıda verilmiştir. Genel bir açılış betiği değildir ama bizi login'e kadar getirir. Bundan iyisi can sağlığı.

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

mount -t proc  proc  /proc
mount -t sysfs sysfs /sys

# mount ile de yapılabilir!
echo /sbin/mdev >/proc/sys/kernel/hotplug
mdev -s

mkdir /dev/pts
mount -t devpts devpts /dev/pts

mkdir /dev/shm
mount -t tmpfs  tmpfs  /dev/shm

hostname UcanLinux

syslogd
```

```
klogd

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

ifconfig eth0 192.168.1.100 up
route add default gw 192.168.1.1

telnetd
```

Buradaki her bir satır üzerinde mutlaka ayrıntılı bir çalışma yapılmalıdır. 20 satırdan daha az bu açılış betiği, bize aşağıdaki özellikleri sağlar.

1. PATH satırı, komutların nerede aranacağını tespit eder.
2. Pusedo dosya sistemlerini bağlar.
3. /dev dosya sistemini sıfırdan üretir.
4. /dev/shm dosya sistemini bağlar ki RAM belleği disk bellek gibi kullanabilelim.
5. hostname tanımlar.
6. Sabit IP tanımlar.
7. Telnet sunucusunu başlatır.

2.5 Nihai Test

"minicom -s" girişi ile "ttyUSB0, 115200, 8N1, NoFlow" tanımları yapılabilir. U-Boot seviyesinde açılış durdurulur ve aşağıdaki komutlar girilerek Login'e kadar gelinir.

```
=> setenv ipaddr 192.168.1.100
=> setenv serverip 192.168.1.21
=> ping 192.168.1.21
=> tftpboot 0x80F80000 am335x-boneblack.dtb
=> tftpboot 0x80007FC0 uImage
=> setenv bootargs console=ttyO0,115200n8
=> bootm 0x80007FC0 - 0x80F80000
```

Sistem tekrar açılır ve U-Boot seviyesinde açılış durdurulur ve aşağıda verilen çalışma yapılabilir.

```
=> mmc rescan

=> mmc list
OMAP SD/MMC: 0 (SD)
OMAP SD/MMC: 1

=> mmc dev
switch to partitions #0, OK
mmc0 is current device

=> ls mmc 0:1             # SD kartta var olan dosyalar.
    66388   mlo           # VFAT için büyük/küçük harf ayrımı yoktur.
   319216   u-boot.img
      138   uenv.txt

3 file(s), 0 dir(s)

=> ls mmc 0
    66388   mlo
   319216   u-boot.img
      138   uenv.txt

3 file(s), 0 dir(s)


MMC'deki env'yi içeri almak:

=> fatload mmc 0 0x81000000 uEnv.txt
=> env import -t 0x81000000 $filesize

=> print ipaddr
ipaddr=192.168.1.100

=> print test1
test1=tftpboot 0x80F80000 am335x-boneblack.dtb ; \
      tftpboot 0x80007FC0 uImage ; \
      bootm 0x80007FC0 - 0x80F80000
```

Login geldikten sonra, minicom'dan root/root kullanıcısı ve şifresi ile veya telnet ile uzaktan root/root ile sisteme giriş yapılabilir. U-Boot tarafında, bootm girişinden hemen sonra, Power-on'dan login'e kadar geçen aşamalar aşağıda kısacaca verilmiştir. Bu aşamalar initramfs destekli açılış için geçerlidir. Katılımcı tarafından bu açılış yordamı mutlaka eksiksiz anlaşılmış olmalıdır.

1. Çekirdek ve DTB tftp ile RAM'a yüklenmiştir. U-Boot yütümeyi çekirdeğe aktarır ve çekirdek açılmaya başlar.
2. Çekirdek açılırken DTB'yi okuyarak donanım hakkında bilgiye sahip olur.
3. Açılış bitince cpio arşivini RAM bellekte açar ve /dev/root'a mount eder.
4. Sonra /dev/root ile gösterilen kök dosya sisteminde /init programını çalıştırır.
5. init programı /sbin/init'i gösterir. /sbin/init programı ayağa kalkar /etc/inittab dosyasına göre yürümeye başlar.
6. /etc/inittab dosyasında bulunan sysinit satırını arar ve bu satırda gösterilen /etc/rcS dosyasını çalıştırır.
7. /etc/rcS dosyası da pusedo dosya sistemlerini bağlar, /dev dizinini kurar, çeşitli programları başlatır.
8. /etc/inittab içinde bulunan getty programı ise seri konsoldan login promptunu getirir.
9. login ekranından sisteme girildiğinde kabuğa düşülür ve Linux altında çalışma başlar.

Başka açılış örnekleri aşağıda verilmiştir.

```
# TEST 1
# Sadece ip bilgileri otomatik alınır.
# Diğer bütün yükleme işlemleri tftp ile yapılır.
#

=> setenv bootargs console=ttyO0,115200n8
=> setenv autoload no  # ip aldıktan sonra yükleme yapma.
=> setenv autostart no # bootcmd'yi çalıştırma.

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

=> print serverip
## Error: "serverip" not defined
```

```
# BBB'nin ip bilgisi ve tftp sunucusunun ip bilgisi mevcut değil.
# ipaddr, bbb'nin IP bilgisini barındırır.
# serverip ise tftp sunucusunun IP bilgisini barındırır.
# dhcp ile her iki ip bilgisi de elde edilir.

=> dhcp
link up on port 0, speed 100, full duplex
BOOTP broadcast 1
DHCP client bound to address 192.168.1.100 (3 ms)

=> print ipaddr
ipaddr=192.168.1.100

=> print serverip
serverip=192.168.1.33

=> tftp 80f80000 am335x-boneblack.dtb
=> tftp 80007fc0 uImage
=> bootm 80007fc0 - 80f80000

Login gelir.


# TEST 2
# dhcp ile yarı otomatik açılış.
# ip ve çekirdek dhcp ile yüklenir.
# dtb tftp ile yüklenir.

=> setenv bootargs console=ttyO0,115200n8
=> setenv autoload yes  # filename yükle.
=> setenv autostart no

=> dhcp 80007fc0  # ip al, uImage yükle.

# setenv loadaddr 80007FC0
# dhcp

=> tftp 80f80000 am335x-boneblack.dtb
=> bootm 80007fc0 - 80f80000

## Booting kernel from Legacy Image at 80007fc0 ...
   Image Name:   Linux-4.1.18
   Created:      2016-04-17  12:13:50 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    11392048 Bytes = 10.9 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 80f80000
```

```
   Booting using the fdt blob at 0x80f80000
   XIP Kernel Image ... OK
   Loading Device Tree to 8ffee000, end 8ffffcd2 ... OK

Starting kernel ...

Login gelir.


# TEST 3
# Autoboot
#
# Yukarıdaki açılış komutları uenvcmd değişkenine ";"
# ile yan yana yazılırsa sistem otomatik olarak boot eder.
```

Birçok iş el yordamı ile yapılmıştır. Yapılan işleri farkında olarak yapmak esas amacımız idi. Bütün konular iyice kavrandıktan sonra, hızlı ilerleyebilmek için komutlar betiklerin içinde toplanabilir.

Tembelin kuralı: İki kereden fazla yazılan her komut betiğe girmeye mahkumdur :)

Örnek bir betik parçası aşağıda verilmiştir.

```
PROJECT_HOME=/nk/workspace/projects/linux.egitimi/bbb4/
SYSROOT=‘$PROJECT_HOME/toolchain/arm/bin/arm-linux-gnueabihf-gcc -print-sysroot‘

KERNEL_SRC=$PROJECT_HOME/linux
KERNEL_OUT=$PROJECT_HOME/out/linux
KERNEL_UIMAGE=$PROJECT_HOME/out/linux/arch/arm/boot/uImage
ROOT_FS=/tmp/RootFS

...

export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-

uimage(){
# Varsa eski cipo arşivini sil.
#
rm -f $KERNEL_OUT/usr/initramfs_data.cpio.gz

make -C $KERNEL_SRC O=$KERNEL_OUT uImage LOADADDR=0x80008000 -j2 || exit 1
make -C $KERNEL_SRC O=$KERNEL_OUT modules                     -j2 || exit 1
make -C $KERNEL_SRC O=$KERNEL_OUT dtbs                            || exit 1
```

```
file $KERNEL_UIMAGE
ls -l $KERNEL_UIMAGE
$PROJECT_HOME/bin/mkimage -l $KERNEL_UIMAGE
}
...
```

Host tarafı için çok basit dhcp sunucu ayarları dosyası aşağıda verilmiştir. Ayarlar yapıldıktan sonra "$ service isc-dhcp-server restart" ile dhcpd tekrar başlatılabilir.

DHCP'nin esas görevi IP ve diğer ağ yapılandırma bilgilerini dağıtmaktır. Ayrıca DHPC sunucusu bootp gibi de davranır. Bunun için "allow bootp" yazmak yeterlidir. Bu sayede aynı anda hem dhcp hem de bootp sunucusu çalıştırmaya gerek kalmaz.

DHCP sunucusu filename ile belirtilen dosyayı, örnekte uImage dosyasını açılış dosyası olarak talep edilen yere gönderir. Yani sunucu her zaman tek dosya gönderir.

Açılış için dtb ve initrd dosyalarına gerek olabilir. dhcp sunucusu aynı anda tek dosya sağlayabildiği için diğer dosyaları gönderemez.

Bu sorun iki türlü çözülebilir. İlk çözümde dtb veya initrd dosyaları tftp ile elde edilir. Ki bu çözüm örneklerde kullanılmıştır.

Diğer çözümde ise mkimage ile birden fazla dosya birleştirilir ve tek dosya haline getirilir. Örneğin uImage, dtb ve initrd dosyaları mkimage ile tek dosya şekline getirilir[42][43]ve filename seçeneğine bu dosya ismi verilir.

```
# /etc/dhcp/dhcpd.conf

ddns-update-style none;
allow bootp;
allow booting;

subnet 192.168.1.0 netmask 255.255.255.0{
}

# u-boot seviyesinde, bdinfo ile BBB'nin MAC
# adresi elde edilir.
```

[42]Kaynak: http://www.denx.de/wiki/pub/U-Boot/Documentation/multi_image_booting_scenarios.pdf
[43]Kaynak: http://www.denx.de/wiki/bin/view/DULG/CombiningKernelAndRamdisk

```
host bbb{
      hardware ethernet 1c:ba:8c:95:ba:bc;
      fixed-address 192.168.1.100;
      filename "uImage";
      next-server 192.168.1.33;
      option host-name "bbb-test";
      option domain-name "bizimsirket.com";
      option routers 192.168.1.1;
      option root-path "/tftpboot";
}
```

Bu config tanımlarına göre BBB ağ tanımlarını aşağıdaki gibi elde eder.

```
DHCPDISCOVER from 1c:ba:8c:95:ba:bc via wlp2s0
DHCPOFFER on 192.168.1.100 to 1c:ba:8c:95:ba:bc via wlp2s0
DHCPREQUEST for 192.168.1.100 (192.168.1.27) from 1c:ba:8c:95:ba:bc via wlp2s0
DHCPACK on 192.168.1.100 to 1c:ba:8c:95:ba:bc via wlp2s0
```

Bu mesajlar, DHCP server makinesinde /var/log/syslog dosyasından görülebilir. Az yer kaplaması için satırların başındaki zaman bilgileri kırpılmıştır. IP'nin elde edilmesi aşağıdaki gibi yorumlanabilir.

u-boot, BBB'nin MAC adresini içine katarak, "MAC:?" biçiminde bir DISCOVER mesajı yayınlar. U-Boot seviyesinde, bdinfo komutu ile, BBB'nin MAC adresi 1c:ba:8c:95:ba:bc şeklinde görülebilir.

DISCOVER mesajını alan DHCP server, MAC adresine 192.168.1.100 değerini atar. Bu IP değerinin config dosyasında fixed-address olarak atandığını hatırlatalım. DHCP server, "MAC:192.168.1.100" bilgisini barındıran OFFER paketini ağda yayınlar.

Bu MAC adresine sahip olan BBB makinesi, paketin kendine gelidiğini anlar ve hemen IP değerini alır. Kabul ettiğini belirtmek için tekrar ağa REQUEST mesajı gönderir.

DHCP server, REQUEST mesajın alır ve ACK mesajı yayınlar ve protokol tamamlanır.

Dikkat edilirse son iki mesajda BBB, bir IP değeri elde etmesine rağmen, sunucu ile konuşmak için bu IP değerini kullanmaz. Her seferinde MAC adresi

ile yayın[44] yapar. Bunun çok basit bir sebebi vardır.

Ağda birden fazla DHCP sunucusu varsa, her sunucu bütün haberleşmeden haberdar olur. Böylece bir DHCP sunucusu, kendi IP değerinin kabul edilip edilmediğini kolaylıkla anlar.

DHPC sunucusu, bu örnekte bir IP havuzundan IP dağıtmaz, doğrudan MAC'a göre IP dağıtır. Gömülü sistem geliştirirken, bu örnekteki gibi, MAC'a sabitlenmiş IP kullanmak daha pratiktir.

Reklam öncesi son söz olarak, kernel ve busybox'ın .config dosyaları farklı bir dizinde veya ortamda yedeklenmelidir.

Reklam: `$ figlet "UcanLinux.Com"`

[44]broadcast

Bölüm 3

Initramfs Destekli Gömülü Sistem-II

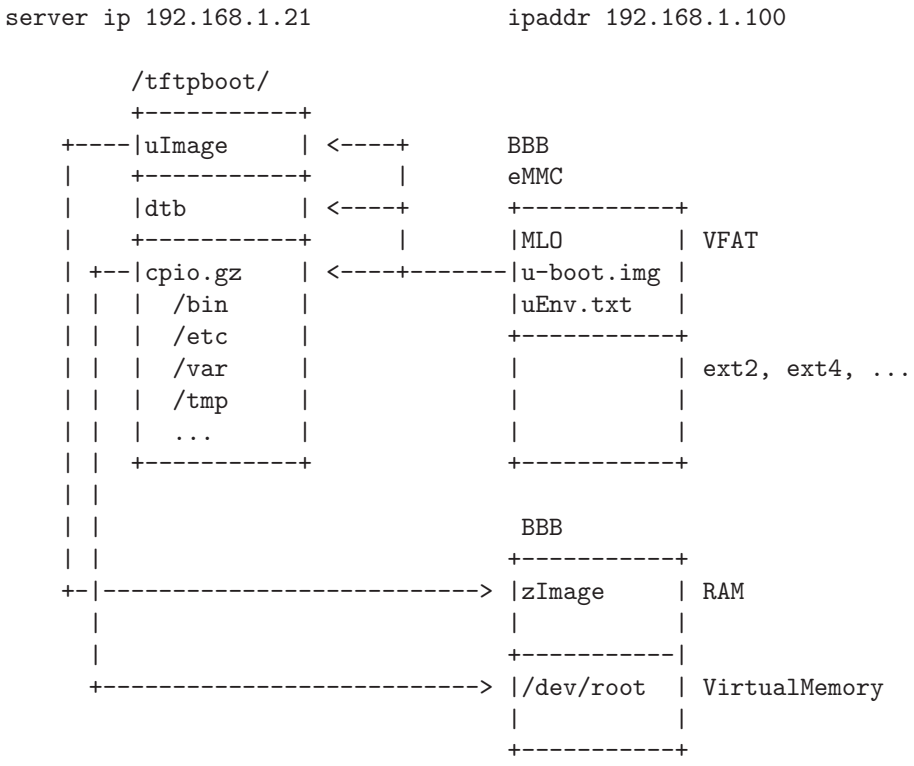
En basit gömülü sistemlerden birisidir. Kök dosya sistemi ayrıktır, yani çekirdek içinde değildir. Sistemin blok şeması Şekil 3.1'de verilmiştir. Çizim çok karıştığı için dtb'den BBB'de ok çıkmamıştır, tabii ki dtb da BBB'ye yüklenir.

Sistemin açılış mantığı çok basittir. BBB'nin ROM'da oturan açılış yükleyicisi MLO'yu yükler. MLO'da u-boot.img'yi yükler. u-boot.img ise, uEnv.txt dosyası içinde uenvcmd satırında ne varsa işletir. Bu satır uImage, dtb ve cpio.gz dosyasını ftp ile belleğe yükler ve bootm ile açılışı başlatır. Login'e kadar geliş, Sayfa 46'de incelenebilir. Arada hemen hemen hiç bir fark yoktur.

Çekirdek ve kök dosya sistemi ağ üzerinde oturur. Çekirdek ve kök dosya sistemi, u-boot tarafından ayrı ayrı yüklenir. Sonra çekirdek kök dosya sistemini RAM içinde açar ve bağlar. Kök dosya sisteminin RAM içinde kurulmasına initramfs tekniği denir. Bir önceki bölümde üzerinde çalışılan gömülü sistemle tamamen aynıdır. En büyük farkı ise kök dosya sistemi çekirdeğe gömülü değildir, ayrıktır.

Daha önce initramfs tekniğinin 2 farklı biçimde uygulandığından bahsedilmişti. Kök dosya sistemi, ya önceki bölümde gösterildiği gibi gibi çekirdek içine gömülür ya da bu bölümde gösterileceği gibi çekirdekten bağımsız yüklenir. Bu bölümde çekirdeten ayrık initramfs tekniği üzerinde çalışılacaktır.

Bir önceki bölümde bahsedilen bütün iyi ve sakıncalı taraflar burada da geçerlidir. Farklı olarak kök dosya sistemi çekirdeğe gömülü olmadığı için,



Şekil 3.1: Initramfs destekli gömülü sistem, tftp.tftpcpio

RootFS içinde bulunan uygulama programlarının kaynak kodlarını açmak gerekmez. Ayrıca çekirdek ve kök dosya sistemi daha kolay güncellenir. Test, kurtarma veya müşteri sistemi olabilir.

Eğer gömülü sistem projesi bir ağ ortamında çalışıyorsa, çekirdek, kök dosya sistemi veya uygulamanın ağ üzerinden yüklenmesi tavsiye edilir. Merkezi bir yerden tftp ve/veya nfs ile gerekli dosyalar alınabilir. BBB tarafında, u-boot, illa ki eMMC veya SD kartta olmalıdır. Fakat PXE benzeri bir boot tekniği ile BBB üzerinde u-boot dahil hiç bir programa gerek olmayabilir. Fakat BBB ile PXE uygulamak pek de yaygın değildir.

3.1 Açılış Yükleyicisi

Bölüm 2.1'deki ile tamamen aynıdır.

3.2 Çekirdek

Çekirdek derlemesi Bölüm 2.2'de verilen teknikle hemen hemen aynıdır. Bölüm 2.2'de cpio arşivi çekirdeğin içine gömülü idi. menuconfig sırasında, cpio arşivinin çekirdeğin içine gömülmesini engellemek için aşağıdaki gibi güncelleme yapılır. Diğer bir deyişle "() Initramfs source file(s)" satırında, parantez içinde verilen ve ROOT_FS'in oturduğu yeri gösteren ifadenin silinmesi yeterlidir.

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

Derleme aşamaları tamamen aynı şekilde yapılır. cpio arşivi çekirdeğe gömülmediği için, çekirdeğin boyu kısalacakatır. "$ ls -l uImage" ile boy kontrolü yapılmalıdır.

3.3 Kök Dosya Sistemi

Bölüm 2'de verildiği gibi kurulur. Daha sonra da RootFS'in içindeki bütün dosya ve dizinlerin sahiplikleri[1] aşağıdaki gibi root kullanıcısı yapılır.

```
$ chown -R root:root RootFS
```

Bir önceki initramfs tekniğinde buna gerek duyulmadı. Çünkü, make menuconfig sırasında Bölüm 2.3'de aşağıdaki gibi bir giriş yapılmıştı.

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

Örnek ROOT_FS dizini altında bulunan user ve group dizinlerinin sahibi nazim:nazim idi. nazim:nazim sahipliğinin rakam değeri 1000 ve 1000'dir. id komutu ile görülebilir. Yukarıdaki girişte 1000 ve 1000 olan bütün user ve group'ların otomatik olarak root yapılması söylenmiştir. Böylece ayrıca chown komutu kullanılmamıştır.

Bu özellik niye vardır? Yani bir önceki örnekte de chown ile RootFS altındaki bütün dosya ve dizinler root yapılabilirdi, ama yapılmadı. Sebebi çok basittir. Gömülü sistem kuruluşu ile uğraşan kişinin root erişimine yetkisi olmayabilir. root yetkisi yoksa "chown root:root ..." komutu kullanılamaz. Yukarıdaki menuconfig girişi bu sakıncayı ortadan kaldırır.

Herkes kendi PC veya notebook'unda çalıştığı için root yetkisine sahiptir. Ama üniversite veya şirketlerde bulunan çalışanlar root yetkisine sahip olmayabilirler. Çünkü muhtemelen başkalarının makineleri üzerinde çalışmaktadırlar.

Bu bölümde cpio arşivi ayrık kurulacaktı. Elimizde daha önce kurulmuş RootFS olduğuna göre, arşiv aşağıdaki gibi adım adım kurulabilir.

```
$ cd RootFS
```

[1] Bakınız: Ekler 9.64

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

$ mkimage -A arm            # tek satırda yazılmalıdır.
          -T ramdisk
          -C none
          -n "ucanlinux, cpio test imaji"
          -d /tftpboot/rootfs.cpio.gz
          /tftpboot/uramfs.img

$ ls -l /tftpboot/rootfs.cpio.gz
$ ls -l /tftpboot/uramfs.img

$ file /tftpboot/uramfs.img
```

Yukarıda yapılan işlemleri özetlersek, find ile dosya ve dizin isimleri elde edilir. cpio komutu newc formatına göre arşiv yapar. gzip komutu cpio arşivini sıkıştırır ki diskte az yer kaplasın. Sonunda arşiv /tftpboot altına atılır.

Daha önce de belirtmiştik. U-Boot imajı olmayan hiç bir dosyayı u-boot programı tanıyamaz, kullanamaz. cpio arşivini u-boot programı işletecektir. O halde bu arşivi, u-boot imajı haline getirmek gerekir. Bir dosyayı u-boot imajı haline getirmek için, dosyanın en tepesine 64 baytlık u-boot başlığını yapıştırmak yeterlidir. Bu başlıkta dosya hakkında bilgiler bulunur. Bu bilgiler mkimage komutu ile yukarıdaki gibi argümanlar yardımı ile verilir.

Kolay okunması için mkimage komutu çok satırda yazılmıştır, aslında tek satır olmalıdır. Nihayetinde ls ve file komutları ile dosyalar incelenebilir.

u-boot tarafından çalıştırılabilecek ayrık bir kök dosya sistemine sahip olundu. Bu kök dosya sistemi, initramfs yöntemi ile yüklenecektir. Yani çekirdek RAM'da kurduğu bir dosya sistemi içine bizim cpio arşivini açıp doğrudan mount ederek kullanacaktır.

3.4 Açılış Betiği

Bölüm 2.4'de verildiği gibi kurulur. Hiç bir farklılığı yoktur.

3.5 Nihai Test

Örnek sistemin açılış sırası Bölüm 2.5'de verilen açılış sırası ile tamamen aynıdır. Farklı olarak cpio imajı çekirdekten ayrı olarak yüklenir. Diğer bir deyişle tftp komutu ile uImage, dtb ağacı ve cpio arşivi tek tek yüklenir. Sonra yükleme adresleri bootm'ye sıra ile verilir. Çeşitli test teknikleri aşağıda verilmiştir.

```
=> setenv ipaddr 192.168.1.100
=> setenv serverip 192.168.1.21
=> ping 192.168.1.21

=> tftpboot 0x80007FC0 uImage
=> tftpboot 0x81000000 uramfs.img
=> tftpboot 0x80F80000 am335x-boneblack.dtb

=> iminfo 80007fc0
=> iminfo 81000000
=> iminfo 80F80000

=> setenv bootargs console=ttyO0,115200n8
=> bootm 0x80007FC0 0x81000000 0x80F80000

## Booting kernel from Legacy Image at 80007fc0 ...
   Image Name:   Linux-4.1.13-UL.2015.12.05.19.56
   Created:      2015-12-05  18:01:03 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    5386992 Bytes = 5.1 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK

## Loading init Ramdisk from Legacy Image at 88080000 ...
   Image Name:   ucanlinux, cpio test imaji
   Created:      2015-12-06  16:12:30 UTC
   Image Type:   ARM Linux RAMDisk Image (uncompressed)
   Data Size:    6272849 Bytes = 6 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK

## Flattened Device Tree blob at 80f80000
   Booting using the fdt blob at 0x80f80000
   XIP Kernel Image ... OK
   Loading Ramdisk to 8fa04000, end 8ffff751 ... OK
   Loading Device Tree to 8f9f2000, end 8fa0379e ... OK
```

```
Starting kernel ...
```

Testler her zaman el yordamı ile yapılmaktadır. Girilen komutların aralarına ';' konularak uenvcmd içine yazılırsa gömülü sistem otomatik olarak açılacaktır.

Elimde bulunan ve u-boot imajı haline getirilmiş kök dosya sistemi aşağıdaki gibi tekrar elde edilebilir. Buradaki bütün incelik dosyanın tepesinde bulunan 64 baytı kaldırmaktan ibarettir.

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

$ dd if=uramfs.img of=ramfs.img bs=64 skip=1
$ ls -l ramfs.img
$ file ramfs.img

$ mv ramfs.img ramfs.img.gz
$ gunzip ramfs.img.gz
$ ls -l ramfs.img

$ file ramfs.img

$ mkdir /tmp/RootFS
$ cd /tmp/RootFS
$ sudo cpio -i -d < $PROJECT_HOME/ramfs.img
$ ls -l
```

Bölüm 4

İdeal Test Sistemi

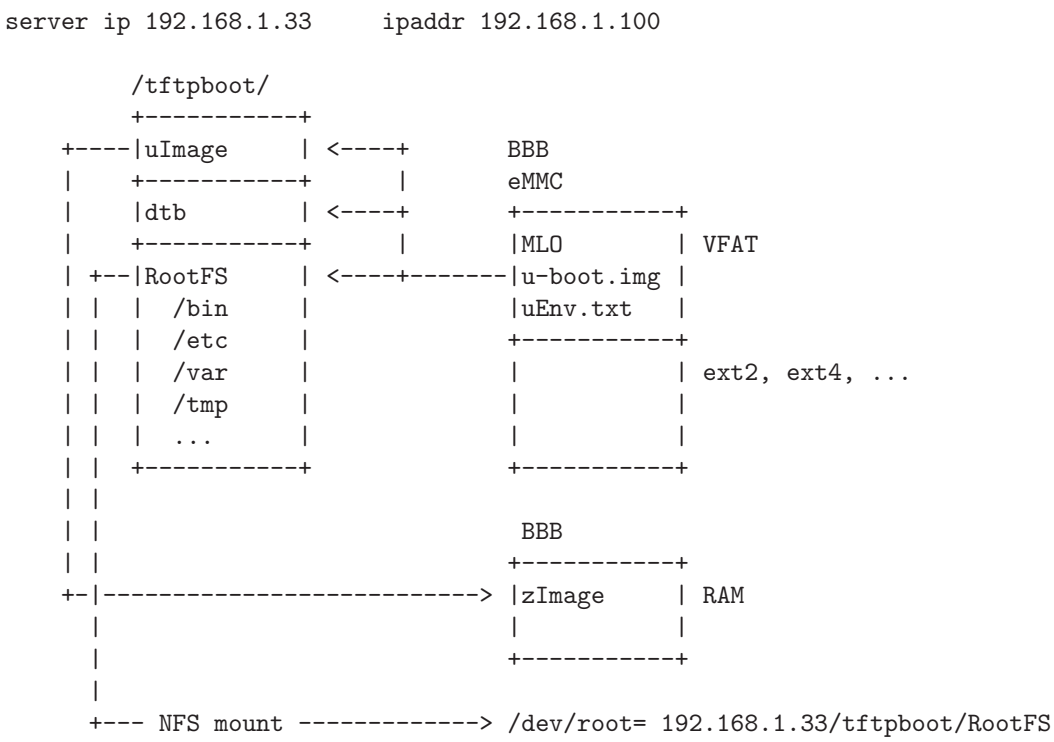
Bu bölümde kurulacak olan ve Şekil 4.1'de resmedilen örnek gömülü sistemde çekirdek tftp ile uzaktan yüklenir. Kök dosya sistem NFS yardımı ile uzaktan mount edilir. BBB sisteminde olabilecek en ideal test ortamıdır. Çünkü kök dosya sisteminde değişiklik yapıldığı an BBB tarafında hissedilir. Ayrı bir yükleme vs yapmaya gerek yoktur.

Bu teknikle kurulan gömülü sistem, eğer gömülü sistem projesi ağ ortamında çalışıyorsa, ideal bir müşteri sistemi de olabilir.

Burada BBB tarafında sadece u-boot mevcuttur. Eğer BBB sistemi PXE gibi ağ tabanlı bir yükleme sistemine yaygın bir destek vermiş olsaydı, BBB tarafında u-boot dahil hiç bir program olmayacaktı ki bu da eksiksik bir test sistemi olacaktı. Bu kadar kusur kadı kızında da olur deyip sistemi standard metodolojimiz ile inşa edelim.

4.1 Açılış Yükleyicisi

Sayfa 12 Bölüm 2.1'deki ile tamamen aynıdır.



Şekil 4.1: İdeal test sistemi, tftp.nfs

4.2 Çekirdek

Bölüm 3.2'de verilen yöntemle aynıdır. Farklı olarak uzaktan kök dosya sistemini bağlamak için aşağıdaki özellikler çekirdeğe katılmalıdır.

```
 File systems  --->
    [*] Network File Systems  --->
        <*>   NFS client support
        <*>     NFS client support for NFS version 2
<*>     NFS client support for NFS version 3
[*]       NFS client support for the NFSv3 ACL protocol extension
<*>     NFS client support for NFS version 4
[*]     Provide swap over NFS support
[*]   NFS client support for NFSv4.1
[*]     NFS client support for NFSv4.2
        (kernel.org) NFSv4.1 Implementation ID Domain
        [ ]     NFSv4.1 client support for migration
[*]   Root file system on NFS
```

Katılımcının da tahmin edeceği gibi, "Root file system on NFS" seçimi, buradaki can alıcı seçimdir. Bu seçim kök dosya sisteminin ağ üzerinden ve NFS desteği ile bağlanacağını çekirdeğe söyler. Bu da tam bizim gömülü sistemin ihtiyaç duyacağı özelliktir.

Uzun yıllar /dev dosya sistemi[1], gömülü sistemler için sorun olmuştur. Açılış sırasında /dev içindeki cihaz isimlerine ve özelliklerine gerek vardır. Fakat henüz kök dosya sistemi bağlanmadığı için /dev sistemindeki cihaz isimlerine de erişim çok zordu. Bu sorun için zaman içinde çeşitli çözümler denenmiştir. Şu anda en ideal çözüm bulunmuş gibidir.

Şöyle ki, çekirdek açılış sırasında mevcut bütün cihazların tanımını kendi içindeki bir dosya sisteminde saklamaktadır. Bu dosya sistemine devtmpfs dosya sistemi[2]denir. Öncelikle çekirdek içindeki bu özellik aşağıdaki örnekteki gibi açılmalıdır.

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

[1]Bakınız: Ekler 9.47
[2]Bakınız: Ekler 9.21

Tahmin edileceği gibi, yukarıdaki 1. seçim, çekirdek içinde bir device dosya sisteminin kurulmasını ve cihaz isim ve özelliklerinin bu dosya sisteminde biriktirilmesini sağlar.

İkinci seçimde ise kök dosya sistemi mount edildikten hemen sonra, /sbin/init çalıştırılmadan da evvel, device dosya sisteminin /dev dizinine otomatik olarak bağlanmasını sağlar. Gömülü sistemlerle uğraşanalar için bu özellik cihaz isimleri konusundaki sorunları kökten çözmüştür.

Pek bu kadar güzel bir özellik varken Initramfs destekli her iki gömülü sistemde de niçin kullanılmamıştır? Initramfs kullanan gömülü sistemlerde yukarıdaki 2. özellik, yani "Automount devtmpfs at /dev, after the kernel mounted the rootfs" özelliği geçerli değildir. Sebebi çok basittir. Initramfs, gerçek bir kök dosya sistemi olarak kabul edilmez. Genelde initramfs işini bitrdikten sonra esas, gerçek kök dosya sistemi bağlanır. İşte çekirdek bu esas kök dosya sisteminden sonra /dev'i otomatik mount eder. Bizde esas kök dosya sistemi olmadığı için bahsedilen otomatik bağlama özelliği olmayacaktır.

Bunun yerine açılış betiğinde doğrudan "mount -t devtmpfs devtmpfs /dev" komutunu yazarsak, çekirdeğin merhametine kalmadan kendi /dev sistemimizi kendimiz bağlamış oluruz.

Ya da busybox sisteminin mdev komutu ile aynı işi yapabiliriz. mdev komutu /sys dosya sistemini gezerek, sistemde mevcut bütün cihazları tanımakta ve /dev sistemini, açılışta otomatik olarak kurmaktadır.

Bu durumda yukarıdaki 2 çekirdek seçeneğine de gerek yoktur. Ama mdev'in mount işlemine göre daha yavaş olduğunu hatırlatmak isteriz.

Derleme ve kuruluş işlemleri, aşağıda topluca verilmiştir.

```
$ export ARCH=arm
$ export CROSS_COMPILE=arm-linux-gnueabihf-

$ make -C $KERNEL_SRC O=$KERNEL_OUT bb.org_defconfig
$ make -C $KERNEL_SRC O=$KERNEL_OUT menuconfig

$ make -C $KERNEL_SRC O=$KERNEL_OUT uImage LOADADDR=0x80008000 -j2
$ make -C $KERNEL_SRC O=$KERNEL_OUT modules                    -j2
$ make -C $KERNEL_SRC O=$KERNEL_OUT dtbs

$ rm -fr $ROOT_FS/lib/modules
$ rm -fr $ROOT_FS/lib/firmware
$ make -C $KERNEL_SRC O=$KERNEL_OUT INSTALL_MOD_PATH=$ROOT_FS modules_install
```

```
$ cp $KERNEL_UIMAGE /tftpboot
$ cp $KERNEL_OUT/arch/arm/boot/dts/am335x-boneblack.dtb  \
          /tftpboot/am335x-boneblack.dtb
```

4.3 Kök Dosya Sistemi

Kök dosya sistemi Bölüm 2.1'deki ile hemen hemen tamamen aynıdır. Initramfs destekli sistemlerde, çekirdek ilk olarak /init programını işletir. NFS veya eMMC veya SD karta gömülü kök dosya sistemlerinde /sbin/init programı işletilir. Örnek sistemimizde, init isimli ayrı bir kod olmadığından, init programı olarak /sbin/init gösterilmiştir. Bunun için sembolik link kullanılmıştır. Bu link'in burada bir önemi yoktur, kaldırılabilir. Ayrıca açılış betiği değişmiştir.

Kök dosya sistemi uzaktan mount edileceği için, cp -a komutu ile /tftpboot altında doğrundan kopyalanablir. Bütün dosya ve dizin sahipliklerinin root:root olmasına dikkat edilmelidir.

4.4 Açılış Betiği

Açılış betiğinde çok ufak değişiklikler yapılmıştır. Initramfs sistemlerindeki rcS ile aşağıdaki rcS'nin karşılaştırılması tavsiye edilir.

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

mount -t tmpfs  tmpfs  /tmp
mount -t sysfs  sysfs  /sys
mount -t proc   proc   /proc

mkdir /dev/pts
mount -t devpts devpts /dev/pts

mkdir /dev/shm
mount -t tmpfs  tmpfs  /dev/shm
```

```
mount -t tmpfs tmpfs /var
mkdir /var/log
mkdir /var/run
mkdir /var/tmp
mkdir /var/cache

hostname UcanLinux

syslogd
klogd

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

telnetd
```

4.5 Nihai Test

Çekirdek, kök dosya sisteminin nerede oturduğunu bilmez, bilemez. Bunu her zaman biz söyleriz. Çekirdek derlemesi sırasında, "[*] Root file system on NFS" seçeneği işaretlenerek, çekirdeğe NFS üzerinden kök dosya sistemi bağlayabilme yeteneği kazadırılmıştı. Kök dosya sisteminin hangi makineden mount edileceği ve diğer bazı ağ parameterleri ve IP alma yöntemi, çekirdeğe parametre olarak aktarılmalıdır. Böylece çekirdek, hem üzerinde çalışıtığı makine için IP talep eder, hem de uzaktaki kök dosya sistemini kendine bağlayabilir.

NFS parameterleri çekirdeğe aşağıdaki formatla verilir.

```
ip=<client_ip>:<server_ip>:<gw_ip>:<netmask>:<hostname>:
     <device>:<autoconf>:<dns1>:<dns2>
```

Karışık gibi görünen bu formatın kullanımı aşağıdaki açılış örnekleri üzerinde verilmiştir.

```
# TEST
#
# Açılış u-boot seviyesinde durdurulur ve her
# değişken tek tek tanıtılır.
# En genel ve en uzun test tekniğidir.
```

```
# En az 1 kez mutlaka denenmelidir.
#
# Aşağıda özet çıkış verilmiş olup, kolay okunması
# için aralara boşluk veya yeni satır eklenmiştir.

Hit any key to stop autoboot:  0

=> setenv ipaddr   192.168.1.100

=> setenv serverip 192.168.1.33

=> ping 192.168.1.33
link up on port 0, speed 100, full duplex
Using cpsw device
host 192.168.1.33 is alive

=> tftp 0x80007FC0 uImage

link up on port 0, speed 100, full duplex
Using cpsw device
TFTP from server 192.168.1.33; our IP address is 192.168.1.100
Filename 'uImage'.
Load address: 0x80007fc0
Loading: #################################################################
         ...
         ########################################
         1.5 MiB/s
done
Bytes transferred = 4863480 (4a35f8 hex)

=> tftp 0x80F80000 am335x-boneblack.dtb
link up on port 0, speed 100, full duplex
Using cpsw device
TFTP from server 192.168.1.33; our IP address is 192.168.1.100
Filename 'am335x-boneblack.dtb'.
Load address: 0x80f80000
Loading: ############
         1.5 MiB/s
done
Bytes transferred = 60627 (ecd3 hex)

=> iminfo 80007fc0

## Checking Image at 80007fc0 ...
   Legacy image found
   Image Name:   Linux-4.1.18
   Created:      2016-04-21  21:08:01 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    4863416 Bytes = 4.6 MiB
```

```
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK

=> setenv nfs_root ip=192.168.1.100:192.168.1.33:192.168.1.33:255.255.255.0:\
     ideal_test:eth0:off root=/dev/nfs rw nfsroot=192.168.1.33:/tftpboot/RootFS

=> setenv bootargs console=ttyS0,115200n8 ${nfs_root}

=> print bootargs
bootargs=console=ttyO0,115200n8 ip=192.168.1.100:192.168.1.33:\
       192.168.1.33:255.255.255.0:ideal_test:eth0:off \
       root=/dev/nfs rw nfsroot=192.168.1.33:/tftpboot/RootFS

=> bootm 0x80007FC0 - 0x80F80000

## Booting kernel from Legacy Image at 80007fc0 ...
   Image Name:   Linux-4.1.18
   Created:      2016-04-21  21:08:01 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    4863416 Bytes = 4.6 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK

## Flattened Device Tree blob at 80f80000
   Booting using the fdt blob at 0x80f80000
   XIP Kernel Image ... OK
   Loading Device Tree to 8ffee000, end 8ffffcd2 ... OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 4.1.18 (nazim@nkoc) (gcc version 4.8.3 20140401 (prerelease)
ool-NG linaro-1.13.1-4.8-2014.04 - Linaro GCC 4.8-2014.04) ) #12 Fri Apr 22 00:07:52 EEST
[    0.000000] CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7), cr=10c5387d
...
[    0.000000] Machine model: TI AM335x BeagleBone Black
...
[    0.000000] Kernel command line: console=ttyO0,115200n8
                ip=192.168.1.100:192.168.1.33:192.168.1.33:255.255.255.0:
                ideal_test:eth0:off root=/dev/nfs rw nfsroot=192.168.1.33:/tftpboot/RootFS
...
[    0.093950] devtmpfs: initialized
...
[    1.381859] mmc0: new high speed SDHC card at address 1234
[    1.388058] mmcblk0: mmc0:1234 SA08G 7.21 GiB
[    1.393817] omap-sham 53100000.sham: hw accel on OMAP rev 4.3
[    1.400806] hidraw: raw HID events driver (C) Jiri Kosina
```

```
[    1.406617]  mmcblk0: p1 p2
...
[    1.583538] mmc1: new high speed MMC card at address 0001
[    1.589410] mmcblk1: mmc1:0001 MMC04G 3.66 GiB
[    1.594135] mmcblk1boot0: mmc1:0001 MMC04G partition 1 1.00 MiB
[    1.600196] mmcblk1boot1: mmc1:0001 MMC04G partition 2 1.00 MiB
...
[    1.667823]  mmcblk1: p1 p2 p3
...
[    5.112515] IP-Config: Complete:
[    5.115775]      device=eth0, hwaddr=1c:ba:8c:95:ba:bc,
                    ipaddr=192.168.1.100,
                    mask=255.255.255.0,
                    gw=192.168.1.33
[    5.126108]      host=ideal_test, domain=, nis-domain=(none)
[    5.131791]      bootserver=192.168.1.33, rootserver=192.168.1.33, rootpath=
[    5.147312] VFS: Mounted root (nfs filesystem) on device 0:15.

[    5.153754] devtmpfs: mounted

[    5.157104] Freeing unused kernel memory: 284K (c0863000 - c08aa000)


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

Welcome to UcanLinux
http://ucanlinux.com

UcanLinux login: root
Password:
Linux UcanLinux 4.1.18 #12 Fri Apr 22 00:07:52 EEST 2016 armv7l GNU/Linux

root@UcanLinux:~ # ifconfig
eth0      Link encap:Ethernet  HWaddr 1C:BA:8C:95:BA:BC
          inet addr:192.168.1.100  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::1eba:8cff:fe95:babc%3204370024/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1585 errors:0 dropped:0 overruns:0 frame:0
          TX packets:699 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1959074 (1.8 MiB)  TX bytes:107938 (105.4 KiB)
          Interrupt:175

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
```

```
          inet6 addr: ::1%3204370024/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@UcanLinux:~ # cat /proc/cmdline

console=ttyO0,115200n8 ip=192.168.1.100:192.168.1.33:192.168.1.33:255.255.255.0:
        ideal_test:eth0:off root=/dev/nfs rw nfsroot=192.168.1.33:/tftpboot/RootFS

root@UcanLinux:~ # mount
192.168.1.33:/tftpboot/RootFS on / type nfs (rw,relatime,vers=2,rsize=4096,wsize=4096,naml
hard,nolock,proto=udp,timeo=11,retrans=3,sec=sys,mountaddr=192.168.1.33,mountvers=1,
mountproto=udp,local_lock=all,addr=192.168.1.33)
devtmpfs on /dev type devtmpfs (rw,relatime,size=241564k,nr_inodes=60391,mode=755)
...
root@UcanLinux:~ #
```

Komutlardaki farkındalığı artırmak için, yukarıdaki örnekte bütün adımlar tek tek girilmiştir. Tabii ki buna gerek yoktur. Her zaman otomatik açılış kolaylıkla yapılabilir. Otomatik açılışta dikkat edilmesi gereken en önemli nokta, a=b,${c}'deki ${c}'nin yerine, mevcut tanımın geçebilmesi için mutlaka "run a" demek gereklidir.

Açılış sırası bu teknikte son derece basittir.

tftp ile çekirdek ve dtb yüklenir. U-Boot aynı zamanda bootargs ile verilen çekirdek parametrelerini, çekirdeğin okuyabilmesi için uygun adrese yerleştirir. Çekirdek açılırken bu parametreleri okur ve root tanımına bakar. root değeri /dev/nfs verildiği için ki aslında böyle bir cihaz yoktur, kök dosya sisteminin NFS üzerinden bağlanacağını anlar ve mount için gerekli parametreleri nfsroot=192.168.1.33:/tftpboot/RootFS tanımından elde eder. rw seçeneği sayesinde de kök dosya sistemi read/write modunda bağlanır.

eMMC veya SD kart üzerinde bulunan dosyalar ise her zaman read/only bağlanmalıdır ki ani kapanmalarda sistemi kaybetmeyelim, üzülmeyelim. NFS üzerindeki kök dosya sistemleri genelde r/w bağlanır. Çünkü sunucunun ani kapanması nadir bir olaydır.

Bu açılış tekniğinde çok büyük bir sorun vardır. BBB'nin mount işlemini ya-

pabilmesi için kendisinin bir IP değerine ve ağ tanımlarına sahip olması gerekir. İşte "ip=192.168.1.100:192.168.1.33:192.168.1.33:255.255.255.0:test2:eth0:off" parametresi, BBB'ye doğrudan statik bir IP ve ağ parametleri atanmasını sağlar.

Bu tanım sayesinde IP ve ağ bilgilerine sahip olan BBB sistemi, nfs üzerinden kök dosya sistemini mount eder, sonra hemen /dev sistemini mount eder. Sonra da /sbin/init programını başlatır ve açılış devam eder.

BBB için yukarıda verilen IP tanımı ve ağ bilgileri statik idi. Bu işlem dhcp kullanılarak otomatik hale getirilebilir.

Açılış testlerini yapabilmek için PC veya host tarafında NFS server, tftp server ve dhcp server çalışıyor olmalıdır. DHPC sunucu tanımları Bölüm 2.5'de verilmişti. tftp sunucu tanımları ise Bölüm 2.2.1'de verilmişti. NFS için host tarafının tanımları da aşağıda verilmiştir.

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

$ service nfs-kernel-server start  # Daha önce çalışmıyorsa.
$ exportfs -avr
```

Config dosyasında güncelleme yapıldığı zaman bunu DHCP sunucuna haber vermek gerekir. Bunun için exportfs komutu kullanılır.

/etc/exports dosyası çok basit bir yapıya sahiptir. Yazım formatı için man sayfasında inceleme yapılabilir. Muhtemelen kimse man sayfasına bakmayacak, teşebbüs bile etmeyecektir. Man sayfası okumak ayrı bir alışkanlıktır ve en iyi stackoverflow cevabından daha faydalıdır. Hep ifade ederim, internete erişim kapansa, pek çok yazılımcı "merhaba dünya" kodu dahi yazamaz.

DHCP destekli çeşitli örnekler aşağıda verilmiştir.

```
# TEST
#
# Çekideğin otomatik yüklenmesi.
# ipaddr, netmask, serverip tanımları verilmez.
# Bu tanımlar dhcp ile u-boot tarafından alınır.
# Çekirdek otomatik yüklenir.
#
```

```
=> setenv autoload yes
=> setenv autostart no

=> setenv kernel_addr 80007fc0

=> dhcp ${kernel_addr}
link up on port 0, speed 100, full duplex
BOOTP broadcast 1
DHCP client bound to address 192.168.1.100 (3 ms)
Using cpsw device
TFTP from server 192.168.1.33; our IP address is 192.168.1.100
Filename 'uImage'.
Load address: 0x80007fc0
Loading: #################################################################
         ...
         ########################################
         1.5 MiB/s
done
Bytes transferred = 4863480 (4a35f8 hex)

=> setenv fdt_addr 80F80000

=> tftp ${fdt_addr} am335x-boneblack.dtb
link up on port 0, speed 100, full duplex
Using cpsw device
TFTP from server 192.168.1.33; our IP address is 192.168.1.100
Filename 'am335x-boneblack.dtb'.
Load address: 0x80f80000
Loading: #############
         1.5 MiB/s
done
Bytes transferred = 60627 (ecd3 hex)

=> setenv nfs_root ip=192.168.1.100:192.168.1.33:192.168.1.33:255.255.255.0:ideal_test:eth
    root=/dev/nfs rw nfsroot=192.168.1.33:/tftpboot/RootFS

=> setenv bootargs console=ttyS0,115200n8 ${nfs_root}

=> bootm ${kernel_addr} - ${fdt_addr}

## Booting kernel from Legacy Image at 80007fc0 ...
   Image Name:   Linux-4.1.18
   Created:      2016-04-21  21:08:01 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    4863416 Bytes = 4.6 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 80f80000
```

```
   Booting using the fdt blob at 0x80f80000
   XIP Kernel Image ... OK
   Loading Device Tree to 8ffee000, end 8ffffcd2 ... OK

Starting kernel ...
...
Login:


# TEST
#
# Çekirdeğin ip tanımlarını dhcp ile alması.
# Bu yeni durumda hem u-boot hem de çekirdek dhcp'yi kullanarak IP
# tanımlarını elde edecektir.
#
=> setenv autoload yes
=> setenv autostart no
=> setenv kernel_addr 80007fc0
=> dhcp ${kernel_addr}
=> setenv fdt_addr 80F80000
=> tftp ${fdt_addr} am335x-boneblack.dtb
=> setenv nfs_root ip=dhcp root=/dev/nfs rw nfsroot=192.168.1.33:/tftpboot/RootFS
=> setenv bootargs console=ttyS0,115200n8 ${nfs_root}
=> bootm ${kernel_addr} - ${fdt_addr}

## Booting kernel from Legacy Image at 80007fc0 ...
   Image Name:   Linux-4.1.18
   Created:      2016-04-21  21:08:01 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    4863416 Bytes = 4.6 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 80f80000
   Booting using the fdt blob at 0x80f80000
   XIP Kernel Image ... OK
   Loading Device Tree to 8ffee000, end 8ffffcd2 ... OK

Starting kernel ...
...
[    5.072276] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[    5.092380] Sending DHCP requests ., OK
[    5.112700] IP-Config: Got DHCP answer from 192.168.1.33, my address is 192.168.1.100
[    5.120716] IP-Config: Complete:

[    5.123993]      device=eth0, hwaddr=1c:ba:8c:95:ba:bc, ipaddr=192.168.1.100,
                    mask=255.255.255 .0, gw=192.168.1.1

[    5.134223]      host=bbb-test, domain=bizimsirket.com, nis-domain=(none)
```

```
[    5.141039]      bootserver=192.168.1.33, rootserver=192.168.1.33, rootpath=/tftpboot
[    5.148736]      nameserver0=192.168.1.33
[    5.159564] VFS: Mounted root (nfs filesystem) on device 0:15.
[    5.166016] devtmpfs: mounted
[    5.169373] Freeing unused kernel memory: 284K (c0863000 - c08aa000)
...
Login:
```

NFS destekli kök dosya sisteminin çok ilginç bir sorunu vardır. Aynı anda birden fazla makine aynı kök dosya sistemini kullanırsa ne olacaktır?

Birden fazla makine varsa aynı RootFS'in kullanılması sakıncalı olabilir. Bu sakıncayı gidermenin bir kaç yolu vardır. En yaygın olanı aşağıda kısaca verilmiştir.

/tftpboot altında birden fazla RootFS varsa, ya nfsroot= tanımı her bir makine için ayrı ayrı yapılır. Ya da bu tanım IP'yi alacak şeklide genişletilebilir. nfsroot verilmezse /tftpboot/client_ip veya /tftpboot/hostname kabul edilir.

Birden fazla makinenin NFS üzerinden farklı kök dosya sistemlerini kullanması, aşağıdaki örnek ile gösterilmiştir.

```
# Bordun IP değerini RootFS olarak alan açılış.
# En sondaki %s ifadesine dikkat edilmelidir.
# Çekirdek, otomatik olarak %s ifadesi yerine bordun IP bilgisini veya
# hostname bilgisini yazacaktır.

=> setenv bootargs console=ttyO0,115200n8 \
          ip=dhcp \
          root=/dev/nfs \
          rw \
          nfsroot=192.168.1.33:/tftpboot/%s
```

Yukarıdaki örnekte, otomatik olarak /tftpboot/RootFS yerine /tftpboot/bbb-test kullanılacaktır. Çünkü BBB'nin hostname değeri bbb-test olarak verilmiştir. Hostname bilgisi yoksa doğrudan IP kullanılır, /tftpboot/192.168.1.100 gibi.

Login geldikten sonra, df komutu ile bağlantı noktaları incelenirse, kök dosya sisteminin "192.168.1.33:/tftpboot/bbb-test" gibi bir bağlantı noktasına[3] sa-

[3]mount point

hip olduğu görülebilir.

Açılıştan sonra girilen bazı komutların örnek çıkışları aşağıda verilmiştir. Katılımcı mutlaka bu komutları kendi makinesinde girmeli ve sonuçları incelemelidir.

```
root@UcanLinux:~ # df -a

Filesystem           1K-blocks      Used Available Use% Mounted on
192.168.1.33:/tftpboot/RootFS
                      57541628  34399268  20196364  63% /
devtmpfs                240772         0    240772   0% /dev
tmpfs                   253236         0    253236   0% /tmp
sysfs                        0         0         0   0% /sys
proc                         0         0         0   0% /proc
devpts                       0         0         0   0% /dev/pts
tmpfs                   253236         0    253236   0% /dev/shm
tmpfs                   253236        40    253196   0% /var

root@UcanLinux:~ # mount

192.168.1.33:/tftpboot/RootFS on / type nfs (rw,relatime,vers=2,rsize=4096,
       wsize=4096,namlen=255,hard,nolock,proto=udp,timeo=11,retrans=3,sec=sys
,mountaddr=192.168.1.33,mountvers=1,mountproto=udp,local_lock=all,addr=192.168.1.33)
devtmpfs on /dev type devtmpfs (rw,relatime,size=240772k,nr_inodes=60193,mode=755)
tmpfs on /tmp type tmpfs (rw,relatime)
sysfs on /sys type sysfs (rw,relatime)
proc on /proc type proc (rw,relatime)
devpts on /dev/pts type devpts (rw,relatime,mode=600,ptmxmode=000)
tmpfs on /dev/shm type tmpfs (rw,relatime)
tmpfs on /var type tmpfs (rw,relatime)

root@UcanLinux:~ # ifconfig

eth0      Link encap:Ethernet  HWaddr 1C:BA:8C:95:BA:BC
          inet addr:192.168.1.100  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::1eba:8cff:fe95:babc%3202170472/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1635 errors:0 dropped:0 overruns:0 frame:0
          TX packets:733 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1996086 (1.9 MiB)  TX bytes:113318 (110.6 KiB)
          Interrupt:170

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1%3202170472/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
```

```
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@UcanLinux:~ # cat /proc/cmdline

console=ttyS0,115200n8
   ip=192.168.1.100:192.168.1.33:192.168.1.33:255.255.255.0:net_net_test:eth0:off
   root=/dev/nfs rw nfsroot=192.168.1.33:/tftpboot/RootFS

# Veya

root@UcanLinux:~ # cat /proc/cmdline
console=ttyS0,115200n8 ip=dhcp root=/dev/nfs rw nfsroot=192.168.1.33:/tftpboot/RootFS

root@UcanLinux:~ #
```

Otomatik açılış için, u-boot komutları aralarına ';' konularak girileblir. SD kartı hiç yerinden çıkarmadan, aşağıdaki gibi, canlı canlı güncelleme yapılabilir.

```
$ cat /proc/partitions
major minor  #blocks  name
...
 179        0    7563264 mmcblk0
 179        1      65536 mmcblk0p1
 179        2     131072 mmcblk0p2
 179        8    3833856 mmcblk1
 179        9      98304 mmcblk1p1
 179       10      31296 mmcblk1p2
 179       11    3703232 mmcblk1p3
 179       24       1024 mmcblk1boot1
 179       16       1024 mmcblk1boot0

$ mkdir /tmp/boot
$ mount /dev/mmcblk0p1 /tmp/boot
$ vi /tmp/boot/uEnv.txt

bootargs=console=ttyS0,115200n8 ip=dhcp root=/dev/nfs rw \
         nfsroot=192.168.1.33:/tftpboot/RootFS
kernel_addr=80007fc0
fdt_addr=80F80000
nfs_test=setenv autoload yes ; \
         setenv autostart no ; \
         dhcp ${kernel_addr} ; \
         tftp ${fdt_addr} am335x-boneblack.dtb ; \
         bootm ${kernel_addr} - ${fdt_addr}
uenvcmd=run nfs_test
```

```
$ sync
$ umount /dev/mmcblk1p1
$ reboot

# Login, dışarıdan müdahale olmadan gelecektir.
# İdeal test sistemi işte budur.
#
```

Burada hata yapılırsa, muhtemelen sistem tıkanır. Bu durumda yine u-boot'a düşüp, komutlar el yordamı ile girilir. Ya da SD kart BBB'den çıkarılır ve gerekli güncellemeler PC tarafında yapılır.

Son kapanış notları: Açılış sırasında "stale file handle" denirse, geçersiz bir dosya var demektir. Örneğin nfs bir dizinde ama dizin silinmiş gibi. Ya da NFS'in otomatik aldığı ip, rcS'de değiştirilmiş olabilir.

Login'den hemen önce sistem tıkanırsa, muhtemelen NFS açık değil veya /etc/exports dosyası düzgün ayarlanmamıştır.

Bölüm 5

Tamamı MMC'de Olan Sistem

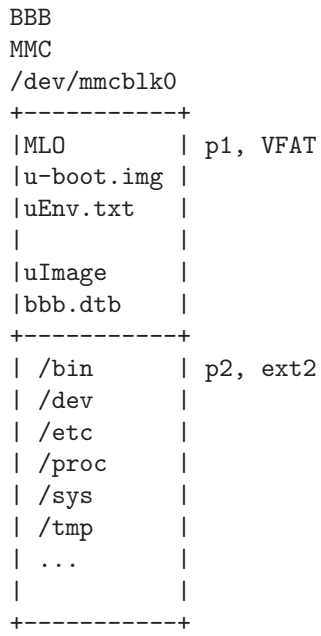
Bu bölümde kurulacak olan ve Şekil 5.1'de resmedilen örnek gömülü sistem çok yaygın bir kullanıma sahiptir. Açılış yükleyicisi, çekirdek ve kök dosya sistemi MMC üzerindedir. Müşteri sistemi olablir. Geliştirme veya test sistemi olarak kullanılması tavsiye edilmez. Host tarafında yapılan güncellemelerin SD karta yazılması gerekir. Bu durumda SD kart sürekli olarak bir host tarafına bir de bord tarafına takılıp çıkartılacaktır. Bu son derece sıkıcı ve vakit alan bir işlemdir.

Gömülü sistem mühendisi her zaman Bölüm 4'de verilen ideal test ortamını kurmalıdır. Bütün sistem istenildiği gibi çalıştıktan sonra MMC'ye taşıma yapılır.

5.1 Açılış Yükleyicisi

Bölüm 3.1'deki hemen hemen aynıdır. Özet olarak, aşağıdaki işlemler yapılacaktır.

1. SD kart, fdisk komutu ile bölümlere ayrılacaktır.
2. İlk bölüme vfat dosya sistemi kurulacaktır.
3. u-boot, kaynak kodundan derlenecektir. Daha önce derlenmişse, tekrar derlemeye gerek yoktur.
4. MLO, u-boot.img, uEnv.txt dosyaları birinci bölüme kopyalanacaktır.



Şekil 5.1: Tamamı MMC'de olan sistem, mmc.mmc

Linux altında çok gelişmiş bölümlendirme programları mevcuttur. Ayıptır söylemesi, bazılarının GUI sistemi dahi vardır. Alışık olduğumuz için bizler bölümlendirme için fdisk kullanmaktayız. Hem inanılmaz derecede basittir, hem betik içinde kullanılabilir hem de işimizi fazlasıyla görmektedir. Katılımıcı, ne yaptığını bildiği sürece farklı bölümlendirme programları kullanabilir.

fdisk ile MMC[1] bölümlendirmesi aşağıda verilmiştir. Önceki yapılanlardan bir farkı yoktur.

```
$ dd if=/dev/zero of=$MMC_DEVICE bs=512 count=1

$ fdisk -H255 -S63 $MMC_DEVICE << EOF
o
n
p
1
```

[1] Aslında bu MMC değil SD karttır. U-Boot sistemi SD kart için de MMC terimini kullandığı için biz de MMC diyoruz, bazen de SD diyoruz.

```
+$BOOT_SIZE
a
1
t
c
n
p
2

+$ROOT_SIZE
p
w
EOF

$ sync
$
```

sync komutu verilmeden asla SD kart yuvasında çekilmemelidir. Yazım işlemi yarım kalabilir. sync komutu ara bellekleri tamamen boşaltmadan bitmez. sync komutunun işini bitirmesi beklenmelidir.

Yukarıda ROOT_SIZE olarak verilecek değer tamamen kök dosya sisteminin büyüklüğüne bağlıdır. 32MB'lık büyüklük şimdilik bize yeterlidir. Çünkü sistemin tamamı busybox destekli olacaktır. Bu tür sistemler çok az yer kaplarlar.

Bölümlendirme bittiğine göre aşağdaki gibi her iki bölüme de dosya sistemleri kurulabilir.

```
$ mkfs.vfat -c -n $VOLUME_NAME  ${MMC_DEVICE}p1
$ mkfs.ext2 -c -L $VOLUME_LABEL ${MMC_DEVICE}p2
```

Şu anda ikinci bölüm gerekli değildir. Hazırlık babından kurulmuştur.

Bizdeki geliştirme ortamında MMC_DEVICE değeri "/dev/mmcblk0" şeklinde tanıtılmıştır. "-c" seçeneği SD karttaki arızalı blokları[2], dosya sistemini kurmadan evvel araştırır. Arızalı blok varsa, dosya sistemi kuruluşunda bu blokları kullanmaz.

SD veya MMC kartlar son derece nazik cihazlardır. Çok çabuk bozulurlar. Özellikle veri tabanı sistemleri veya çok sıklıkla yapılan yazım işlemlerinde

[2]bad blocks

aşırı derecede çabuk pes ederler. Bu tür işlemler için farklı çözümlere gidilmeli, SD/MMC üzerindeki kök dosya sistemleri asla read/write bağlanmamalı ve log'lu dosya sistemi kullanılmamalıdır. Yoğun kullanımda derhal bozulacaklardır.

Şimdiye kadar bölümlendirme yapıldı ve bölümlere dosya sistemleri kuruldu. u-boot sisteminin daha önce derlendiğini kabul edersek, açılış için gerekli dosyalar ilk bölüme aşağıdaki gibi kopyalanabilir.

```
$ mount ${MMC_DEVICE}p1 $BOOT_MP

# MLO, ilk kopyalanmalıdır.
$ cp $UBOOT_OUT/MLO         $BOOT_MP
$ cp $UBOOT_OUT/u-boot.img  $BOOT_MP

$ cat <<EOF > /tmp/uEnv.txt
bootargs=console=ttyS0,115200n8 root=/dev/mmcblk0p2 ro rootfstype=ext2 rootwait fixrtc
mmc_test=load mmc 0:1 80007fc0 uImage ; \
         load mmc 0:1 0x80F80000 bbb.dtb ; \
 bootm 0x80007FC0 - 0x80F80000
uenvcmd=run mmc_test
EOF

$ cp /tmp/uEnv.txt $BOOT_MP/

$ sync
$ df $MP
$ umount $BOOT_MP
```

uEnv.txt içindeki "bootargs=console=ttyS0,..." ifadesi ilk bakışta kafa karışıklığı yaratabilir. Bu ifade "bootargs=(console=ttyS0,...)" şeklinde düşünülebilir. Birinci "=" ifadesinden sonra bütün ifade çekirdeğe aittir. Çekirdek ifadesi de "console=..." şeklinde başladığı için çift "=" işareti meydana çıkmıştır.

MP, "mount point" anlamındadır. Kendi geliştirme sistemimizde MP=/mnt/boot olarak tanıtılmıştır.

mount komutu ile ilk bölüm bağlanır ve açılış için gerekli 3 dosya sıra ile bu bölüme atılır. MLO, mutlaka ilk önce kopyalanmalıdır, yoksa sistem açılmaz. uEnv.txt dosyası yukarıdaki gibi önce /tmp altında kurulmakta sonra dosya sistemine atılmaktadır.

Daha önceki sistemlerden farklı olarak birkaç yeni parametre gelmiştir. Bunları kısaca inceleyecek olursak,

root=/dev/mmcblk0p2 ile, çekirdeğe kök dosya sisteminin nerede oturduğu söylenir. Unutulmamalıdır ki bootargs=... olarak verilen ifadenin sağ tarafının u-boot ile bir ilgisi yoktur. İfadenin sağ tarafı, çekirdeğe geçirilecek parametreleri tanımlar. Yani u-boot sistemi, bu ifadeleri tek bağlı liste haline getirip çekirdeğin bildiği bir adrese yerleştirir. Çekirdek de açılırken bu bilgileri bağlı liste içinden okur. Diğer bir deyişle, u-boot sistemi bootargs ile verilen ifadeyi gözü kapalı biçimde çekirdeğe aktarır.

ro ile verilen çekirdek parametresi, kök dosya sisteminin read/only bağlanacağını söyler. Açılış betiklerinde kök dosya sistemi istenirse read/write modunda bağlanabilir.

rootfstype=ext2 parametresi, kök dosya sisteminin ext2 tipinde bir dosya sistemi üzerine kurulduğunu çekirdeğe haber verir. Çekirdek bu parametreyi bulamazsa, içinde tanımlı olan, modüller hariç, dosya sistem tiplerini tek tek dener. Sonuçta rootfstype tanımının yapılması yavsiye edilir, açılış hızını artırır.

rootwait parametresi çok önemlidir. Unutulursa, her iş düzgün yapılmış olsa bile sistem açılmayabilir. Çekirdek yüklendikten berlirli bir süre sonra MMC/SD kartı tespit eder. Hemen tespit edemez. İşte bu parametre ile, kart tespit edilene kadar, ya da daha genel ifade ile, kök dosya sisteminin üzerinde oturduğu cihaz tespit edilene kadar, kök dosya sisteminin mount edilmesi geciktirilir.

BBB sistemlerinde gerçek zaman saati[3] devresi mevcuttur. Yani BBB sistemi, kolumuzdaki saat gibi, duvar saati gibi, zamanı tutan bir entegre devreye sahiptir. Fakat RTC'nin beslemesi mevcut değildir. Ucuz olsun diye BBB içine yassı pil konulmamıştır. Güç gidince zaman bilgisi kaybolmakta ve tarih 1.Ocak.2000'e geri dönmektedir. Cihazın ağa bağlı olacağı ve bir yerlerden zamanı temin edebileceği kabul edilmiştir.

Linux, açılırken kök dosya sistemi içine mount zamanını yazar. fixrtc parametresi çekirdeğe, sistem saati olarak, en son mount zamanını almasını söyler. Bu kaba bir zamandır, hassas değildir ama en azından 1.Ocak.2000[4]gece yarısını görmekten daha iyidir.

Bilgi babından bahsedelin, bir de Unix'in başlangıç zamanı vadır. 1.Ocak.1970 Gece yarısına, Unix Başlangıç Zamanı[5] denir. Zaman sayıcıları, yani bil-

[3]real-time clock

[4] BBB için epoch time UTC 1.Ocak.2000 tam gece yarısıdır.

[5]Unix Epoch Time

gisayar içindeki gerçek zaman saatleri, genelde 1'er saniyelik arlıklarla zamanı sayarlar. 32 bitlik bir alanda, milattan sonraki bütün saniyeleri tutmak mümkün değildir. Bit sayısı yetmez. 1.Ocak.1970 gece yarısı başlangıç kabul edilip, sayılan saniyeler bu başlangıç zamanı üzerine eklenerek gerçek zaman elde edilebilmektedir. Ama yine de 32 bitlik alanda tutulababilecek saniye sayısı 2038'de tükenecektir.

Eğer gömülü sistemin ağ bağlantısı varsa, ya da GPS gibi hassas bir zaman kaynağına sahipse, NTP protokolü ile çok hassas zaman bilgisi elde edilebilir.

mmc_test satırında özel bir bilgi mevcut değildir. Bütün komutlar daha önce kullanılmıştı. Farklı olarak "mmc 0:1" ifadesi gelmiştir.

"mmc 0" ifadesi, birinci MMC kartı gösterir. U-Boot tarafı için, birinci MMC kartı, dışarıdan takılan SD karttır.

"mmc 1" ifadesi 2. MMC kartı ifade eder. U-Boot tarafı için, ikinci MMC kartı, içerde gömülü olan eMMC kartı ifade eder. Kernel tarafında bu tanımların tamamen ters olduğunu hatırlatım.

"mmc 0:1" ifadesi de SD kartın 1. bölümü, yani vfat bölümüdür. "mmc 0:2" ifadesi de benzer şekilde SD kartın 2. bölümüdür. Örneğimizde ext2 formatlanmış bölümdür.

uenvcmd komutunun sağındaki ifade gözü kapalı işletilir. Her zaman "run X" denildiğinde X'in içindeki ifadeler u-boot tarafında işletilir.

Şimdi SD kart BBB'ye takılır ve güç verilirse, u-boot açılacak ama çekirdek olmadığı için sistem çakılacaktır.

5.2 Çekirdek

Çekirdek aşağıdaki gibi derlenir.

```
$ export ARCH=arm
$ export CROSS_COMPILE=arm-linux-gnueabihf-

$ make -C $KERNEL_SRC O=$KERNEL_OUT bb.org_defconfig
$ make -C $KERNEL_SRC O=$KERNEL_OUT menuconfig
$ make -C $KERNEL_SRC O=$KERNEL_OUT uImage LOADADDR=0x80008000
```

```
$ make -C $KERNEL_SRC O=$KERNEL_OUT modules
$ make -C $KERNEL_SRC O=$KERNEL_OUT dtbs
```

Daha önce yapılanlardan bir farkı yoktur. Kök dosya sisteminde ext2 seçildiği için, menuconfig sırasında ext2 seçeneği çekirdeğe katılmalıdır. Kök dosya sistemi desteği çekirdeğe modül olarak eklenemez.

Derleme bitince uImage ve dtb dosyası elde edilir. Bu iki dosya, aşağıdaki gibi, vfat bölümüne kopyalandığında çekirdek u-boot tarafından açılacaktır.

```
$ mount ${MMC_DEVICE}p1 $BOOT_MP

$ cp $KERNEL_UIMAGE  $BOOT_MP
$ cp $KERNEL_OUT/arch/arm/boot/dts/am335x-boneblack.dtb $BOOT_MP/bbb.dtb
```

Kopyalama sırasında "am335x-boneblack.dtb" yerine kısaca bbb.dtb yazılmıştır. İsmin bir önemi yoktur. Bu isim sadece eEnv.txt içinde "load mmc 0:1 0x80F80000 bbb.dtb" şeklinde kullanılır. Başka bir yerde kullanılmaz.

BBB'ye tekrar güç verilirse, hem u-boot hem de çekirdek açılacaktır. Ama çekirdek kök dosya sistemini bulamadığı için sistem tıkanacaktır. Tıkanma sırasında çekirdeğin verdiği mesajları incelemek çok faydalı olacaktır.

Şimdi tek eksiğimiz kök dosya sistemi ve kök dosya sistemi içinde oturan açılış betiğidir.

5.3 Kök Dosya Sistemi

Yine busybox destekli kök dosya sistemi kurulacaktır. Önceki örneklerden bir farkı yoktur. İşlemler topluca aşağıda verilmiştir.

```
$ export ROOT_MP="/mnt/root"
$ export MMC_PART=${MMC_DEVICE}p2
$ export SYSROOT=‘$PROJECT_HOME/toolchain/arm/bin/arm-linux-gnueabihf-gcc -print-sysroot‘

# Eski RootFS'i tamamen sil.
#
$ rm -fr $ROOT_FS
$ mkdir $ROOT_FS
```

```
# Iskeleti kopyala.
#
$ cp -a RootFS.skel/*  $ROOT_FS/

# Busybox'ı kopyala.
# Daha önce install edilmiş olmalıdır.
#
$ cp -a $BUSYBOX_OUT/_install/* $ROOT_FS

# busybox'ın ihtiyaç duyduğu kütüphaneleri kopyala.
#
$ cp -a $SYSROOT/lib/ld-linux-armhf.so.3 $ROOT_FS/lib/
$ cp -a $SYSROOT/lib/arm-linux-gnueabihf $ROOT_FS/lib/

# Varsa, kernel modüllerini ve firmware dosyalarını kopyala.
# Çekirdek ve modülleri daha önce derlenmiş olmalıdır.
#
$ make -C $KERNEL_SRC O=$KERNEL_OUT INSTALL_MOD_PATH=$ROOT_FS modules_install

# Gereksiz dosyaları sil.
#
$ rm -f $ROOT_FS/linuxrc

# issue dosyasını kur.
# Çok gerekli :)
#
$ figlet UcanLinux | head -5 > $ROOT_FS/etc/issue
$ sed -i 's/\\/\\\\/g' $ROOT_FS/etc/issue
$ echo "\nWelcome to UcanLinux\nhttp://ucanlinux.com\n" >> $ROOT_FS/etc/issue

# Sahliplikler root:root olmalı.
#
$ chown -R root:root $ROOT_FS/*

# Incele bakalım, ne kadar yer işgal edilmiş.
#
$ ls -l $ROOT_FS
$ du -ks $ROOT_FS
```

Yukarıda yapılan işlemler, ROOT_FS ile temsil edilen dizine, kök dosya sistemini kurmuştur. Bu kök dosya sistemi olduğu gibi SD'nin 2. bölümüne kopyalanır. Böylece çekirdek tarafında mount edilebilecek hale gelir. Aşağıda, ROOT_FS'in içeriği SD'nin 2.bölümüne kopyalanmaktadır.

```
# SD'nin 2. bölümünü bağla.
#
$ mount $MMC_PART $ROOT_MP

# Host tarafında kurulan ROOT_FS/ dizinini MMC'ye kopyala.
# MMC, host tarafındaki kart okuyucuya takılmalıdır.
#
$ cp -a $ROOT_FS/*  $ROOT_MP

# İncele.
$ ls -l $ROOT_MP
$ df $ROOT_MP

$ sync
$ umount $MMC_PART
```

5.4 Açılış Betiği

/etc/rcS isimli örnek açılış betiği aşağıda verilmiştir.

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

mount -t sysfs  sysfs  /sys
mount -t proc   proc   /proc

mkdir /dev/pts
mount -t devpts devpts /dev/pts

mkdir /dev/shm
mount -t tmpfs  tmpfs  /dev/shm

mount -t tmpfs tmpfs /tmp    # <-- RO'dan RW'a geçiş.

mount -t tmpfs tmpfs /var    # <-- RO'dan RW'a geçiş.
mkdir /var/log
mkdir /var/run
mkdir /var/tmp
mkdir /var/cache

hostname UcanLinux

syslogd
```

```
klogd

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

telnetd
df
```

Açılış betikleri birbirlerinden çok az farklılık gösterir. Buradaki en büyük yenilik, /tmp ve /var dosyalarının tmpfs dosya sistemine bağlanmış olmalarıdır.

Kök dosya sisteminde /var, /tmp gibi yazılabilir dizinlerin olması zorunludur. Ani kapanmalarda tedbir amaçlı, kök dosya sistemimiz read/only durumundadır. Yazılabilir dizin ihtiyacını karşılamanın en basit yollarından birisi, yazılabilir dizinleri /tmpfs dosya sistemine bağlamaktır. tmpfs dosya sistemi ram'da açılan çok özel disk gibi düşünülebilir. Çekirdek derlemesinde seçilmese bile, tmpfs dosya sistemi desteği her çekirdekte bulunur.

Güç gidince bilgiler de gider. Kalıcı bilgileri saklamak için MMC üzerinde yeni bir bölüm fdisk ile yaratılıp, üzerine dosya sistemi kurularak, mount edilebilir.

read/write bağlanan bu gibi bölümlerde ext2 dosya sistemi kurulmamalıdır. ext2 dosya sistemi ani kapanmaya karşı aşırı duyarlıdır.

Genelde ext3 veya ext4 gibi log tabanlı bir sistem kurulabilir. Hangi dosya sistemi kurulursa kurulsun, gerçek cihazlar üzerinde read/write bağlanan her dosya sistemi, açılışta read/only bağlanır, sonra "file system check" yapılır ve sonra read/write bağlanır. Tek istisna, UBIFS gibi doğrudan NAND cihazlara kurulan dosya sistemleridir. Bu dosya sistemleri doğrudan bağlanır ve hatalara karşı inanılmaz dayanıklıdırlar. Fakat yine de kök dosya sistemleri her zaman r/o bağlanmalıdır, bu altın kuraldır.

ext3 ve ext4 gibi log tabanlı dosya sistemleri salt okunur[6] bağlansa bile, diske yazım yapılma ihtimali her zaman vardır. ext2 için bu geçerli değildir. Salt okunur bağlanan dosya sistemileri için ext2 biçilmiş kaftandır. r/w bağlanan sistemlerde ise kullanılmamalıdır.

[6] r/o veya read/only

5.5 Nihai Test

SD kart PC'den çıkarılır, BBB'ye takılır. Her işi düzgün yaptıysak ki ilk denemelerde mümkün değildir, sistem MMC'den açılacaktır. Bu tür sistem kuruluşu çok yaygındır.

Açılışta u-boot seviyesinde durdurulup, uEnv.txt içindeki uenvcmd satırı tek tek el ile girilip test edilebilir.

Örnek bir çıkış aşağıda verilmiştir.

```
...
VFS: Mounted root (ext2 filesystem) readonly on device 179:2.   <-- RO

devtmpfs: mounted

Freeing init memory: 228K

$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/root                63461     10647     49538  18% /
devtmpfs                256092         0    256092   0% /dev
tmpfs                   256204         0    256204   0% /dev/shm
tmpfs                   256204         0    256204   0% /tmp
tmpfs                   256204        40    256164   0% /var
                                            ^
                                          toplamı, sanal bellekten fazla!
```

tmpfs dosya sistemi her zaman RAM belleğin yarısı kadar kapasiteye sahiptirler. Yukarıdaki df çıktısına göre toplam dosya sistemi kapasitesi, tmpfs sistemleri için 1GB kadardır. Ama ram belleğimiz 512MB'dir. Bu nasıl olur?

tmpfs dosya sistemi mevcut kapasiteyi, kurulduğu an RAM'dan çalmaz. İhtiyaç oldukça büyür, içeriği silindikçe küçülür. Ayrıca ihtiyaç oldukça swap alanı da kullanır. Bu arada yeri gelmişken, gömülü sistemlerde swap alanı kullanılmamalıdır. Aşırı derecede swap'a düşme olursa, MMC/SD gibi cihazlar çok çabuk bozulacaklardır.

Bölüm 6

MMC'de CPIO Kullanan Sistem

Bu bölümde kurulacak olan ve Şekil 6.1'de resmedilmiştir. Bu gömülü sistemde hemen her zamanki gibi vfat bölümünde birinci ve ikinci aşama yükleyicileri olan MLO ve u-boot.img bulunmaktadır. Ayrıca u-boot.img'nin sistemi nasıl açacağına dair yapılandırma dosyası uEnv.txt ismi ile ilk bölümde oturmaktadır.

Şekilden de görüleceği gibi çekirdek imajı ve imaja eşlik eden dtb dosyası yine birinci bölümde oturmaktadır. Tekrar etmek gerekirse, dtb dosyası, çekirdeğin donanımla ilgili yapılandırma bilgilerini barındıdır.

Buraya kadar daha önce üzerinde çalışılan sistemlere benzeşim vardır. Esas farklılık ise kök dosya sisteminin cpio arşivi halinde birinci bölümde, yani çekirdekle birlikte aynı yerde oturmasıdır.

Tahmin edileceği gibi bu sistem bir initramfs sistemidir. Tekrar edecek olursa, initramfs destekli sistemlerde kök dosya sistemi çekirdek tarafında doğrudan RAM içinde kurulur.

Eğer kök dosya sistemi çok büyük değilse, bu tür bir sistem çok idealdir. Initramfs sistemlerinin bütün iyi taraflarını barındırır. Daha önce de bahsettiğimiz gibi, cpio arşivi çekirdeğe gömülü olmadığından, uygulamaların kaynak kodunu açamaya gerek yoktur. En önemli özelliklerden birisi, ani kapanmalarda kök dosya sisteminin bozulma ihtimali yoktur ve açılış son derece hızlıdır.

```
BBB
MMC
/dev/mmcblk0
+-----------+
|MLO        | p1, VFAT
|u-boot.img |
|uEnv.txt   |
|           |
|uImage     |
|bbb.dtb    |
|           |
|uramfs     | uramfs = (u-boot header) + (cpio.gz)
+-----------+
```

Şekil 6.1: MMC'de initramfs kullanımı, mmc.mmccpio

Bu örnek sistemde kök dosya sistemini güncellemek için yeni bir uramfs dosyasının yüklenmesi yeterli olacaktır. Bu yükleme uzaktan dahi, son derece kolay bir biçimde rsync, scp veya ssh komutları ile kolaylıkla yapılabilmektedir.

Bir insansız hava aracı çalışmasında böyle bir sistem kullanılmıştır. GPS'den verilerin okunması, ppp ile gönderilmesi, uzaktan cihaza bağlanma, otomatik kontrol sistemi gibi, C ve Bash ile yazılan pek çok program ve komut, kök dosya sistemine eklenmiştir.

MMC'de arta kalan kapasite, kalıcı verilerin ve log'ların saklanması için kullanılabilir.

Daha önceki bölümlerde kurulan dosyalarla, bu bölümdeki örnek sistem çok az bir çaba ile kurulabilecektir.

6.1 Açılış Yükleyicisi

fdisk veya bölümlendirme yapan bir programla, daha evvelki örneklerde olduğu gibi bölümlendirme yapılır. Tek bir bölüm olması yeterlidir. fdisk ile örnek bölüm kuruluşu aşağıda verilmiştir.

```
dd if=/dev/zero of=$MMC_DEVICE bs=512 count=1

fdisk -H255 -S63 $MMC_DEVICE << EOF
o
```

```
n
p
1

+$BOOT_SIZE
a
1
t
c
p
w
EOF

sync
```

Elimizde daha önce bölümlendirilmiş bir MMC varsa, o da kullanılabilir. Ayrıca fdisk ile yeniden bölümlendirme yapmaya gerek yoktur. MMC'de fazladan bölüm olmasının, açılışa bir etkisi yoktur.

uEnv.txt dosyası aşağıdaki gibi olabilir.

```
bootargs=console=ttyS0,115200n8
uenvcmd=load mmc 0:1 80007fc0 uImage;    \
        load mmc 0:1 0x80F80000 bbb.dtb; \
        load mmc 0:1 0x88080000 uramfs;  \
        bootm 0x80007FC0 88080000 0x80F80000
```

MLO, u-boot.img ve uEnv.txt dosyaları 1. bölüme kopyalanır. SD kart BBB'deki yuvasına takılarak u-boot'un açılıp açılmadığı kontrol edilebilir.

6.2 Çekirdek

Bölüm 3.1'de verildiği biçimde çekirdek derlenir. Elde edilen uImage ve dtb dosyaları 1. bölüme kopyalanır. SD kart BBB'deki yuvasına takılarak çekirdeğin yüklenişi izlenebilir. Tabii ki kök dosya sistemi olmadığı için çakılacaktır.

6.3 Kök Dosya Sistemi

Kök dosya sistemi Bölüm 3.1'de verildiği biçimde kurulur. Bu bölümde elde edilen cpio arşivi /tftpboot altına atılıp, oradan u-boot tarafından belleğe yükleniyordu. Sonra da bootm komutu ile açılış yapılıyordu. Şimdi aynı arşiv, diskin 1. bölümüne kopyalanır. Artık 1. bölümde gerekli bütün dosyalar mevcuttur.

cpio arşivi her zaman u-boot imajı olarak kullanılır. Özetle, örnek ROOT_FS içindeki dizin ve dosyalar cpio arşivi haline getirilir. Arşiv, kabaca bütün dosyaların alt alt dizilmesinden başka bir şey değildir. Bu hali ile arşiv çok büyüktür. gzip ile sıkıştırılır ve sıkıştırılmış arşiv elde edilir. Sonra da bu sıkıştırılmış arşivi u-boot'un işleyebilmesi için, arşiv tepesine 64 baytlık bilgi eklenir. Nihayetinde uramfs adını verdiğimiz bir dosya oluşur.

6.4 Açılış Betiği

Açılış betiği aşağıda verilmiştir.

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

mount -t proc  proc  /proc
mount -t sysfs sysfs /sys

echo /sbin/mdev >/proc/sys/kernel/hotplug
mdev -s

mkdir /dev/pts
mount -t devpts devpts /dev/pts

mkdir /dev/shm
mount -t tmpfs  tmpfs  /dev/shm

hostname UcanLinux

syslogd
klogd

ifconfig lo 127.0.0.1 up
```

```
route add -net 127.0.0.0 netmask 255.0.0.0 gw 127.0.0.1 lo

ifconfig eth0 192.168.1.100 up
route add default gw 192.168.1.1

telnetd
```

/dev kuruluşu için mdev[1] kullanılmıştır. Bir donanım eklendiği veya çıkarıldığı zaman Linux çekirdeği /proc/sys/kernel/hotplug'da bulunan programı çağırır. Standard dağıtımlarda bu program için udev programı kullanılır. Gömülü sistemlerde daha çok busybox tarafından sağlanan mdev programı kullanılır.

Çekirdeğe mdev programının kullanıldığını söylemek için "echo /sbin/mdev >/proc/sys/kernel/hotplug" satırı kullanılır. Yani programın tam ismi hotplug içine, echo ile yazılır. Böylece yeni bir cihaz eklendiğinde veya çıkarıldığında, çekirdek mdev programını çağırır.

mdev kodu, cihaza ait /dev isimlerini dinamik olarak yaratır veya siler. Ayrıca bu cihazlarla ilişkili sürücüleri yükler, mod ve sahiplik bilgilerini [2] atar.

Örneğin borda bir usb/seri çeviricisinin takıldığını kabul edelim. Çekirdek bunu anlayacak ve mdev programını çağıracaktır. /mdev programı da /dev/ttyUSB0 gibi bir cihaz ismi üretecektir. Kablo çekildiğinde cihaz ismi otomatik olarak mdev tarafından silinecektir.

mdev programı, çekirdeğe, gerekli firmware dosyalarını da yükler. mdev sisteminin yapılandırma bilgileri /etc/mdev.conf içinde bulunabilir. Gömülü sistemde çok özel bir cihaz yoksa, varsayılan tanımlamalar yeterlidir.

Bu arada /proc dosya sisteminin sözde dosya sistemi[3][4] olduğunu hatırlatalım. Bu dosya sisteminin görevi user ve kernel space arasında bilgi alışverişini sağlamaktır.

Çekirdek ile user space arasında bir kaç şekilde bilgi alışverişi yapılabilir. En yaygın ve basit olan /proc dosya sistemidir. Örneğin yukarıda "echo /sbin/mdev > /proc/sys/kernel/hotplug" komutu ile doğrudan çekirdeğe /sbin/mdev bilgisi gönderilmektedir.

[1] Kaynak: https://git.busybox.net/busybox/plain/docs/mdev.txt
[2] Bakınız: Ekler 9.64
[3] pseudo file system
[4] Bakınız: Ekler 9.70

ls -l komutu ile /proc altındaki dosya sistemlerinin mod bilgileri incelenebilir. Mod içinde "r" barındıran dosyalardan veri okunabilir, yani çekirdekten veri alabiliriz. Örneğin cpu bilgilerini veren dosya /proc/cpuinfo içindedir. "ls -l" ile aşağıdaki gibi incelendiğinde bu dosyanın sadece read modunda işlenebileceği görülebilir.

```
$ ls -l /proc/cpuinfo
-r--r--r-- 1 root root 0 Mar 26 16:53 /proc/cpuinfo

$ cat /proc/cpuinfo
```

Bu dosya cat ile okunduğunda, çekirdeğin sağladığı verileri görmüş oluruz. Benzer şekilde hotplug dosyası da aşağıdaki gibi incelenebilir.

```
$ ls -l /proc/sys/kernel/hotplug
-rw-r--r-- 1 root root 0 Mar 26 16:42 /proc/sys/kernel/hotplug
```

Çıkştan da görüleceği gibi, /proc/sys/kernel/hotplug dosyasına, root kullanıcısı write yapabilir. Çünkü bu dosya root kullanıcısı için "rw-" moduna sahiptir. Ki biraz önce echo komutu ile bu dosyaya veri yazılmış, yani çekirdeğe bilgi gönderilmişti.

Aynı şekilde "rw-" ifadesinde "r" harfi de vardır. Yani bu dosyadan okuma yapılabilir. cat ile okuma yaparsak içinde bilgi bize sunulur, yani çekirdeğin hazırladığı bilgi bize gelir.

/proc ve diğer sözde dosya sistemlerinde bulunan dosyaların boyu her zaman 0'dır. Çünkü böyle bir dosya yoktur. Yani disk gibi bir ortamda mevcut bir bilgi yoktur. Peki bu bilgiler gökten zembille mi inmiştir? Çekirdeğe zembil dersek doğru bir benzetme yapmış oluruz.

Dikkat edilirse dosya ve dizinlerin yaratılma zamanı ya tam komutun girildiği zaman ya da sistemin açıldığı zamandır. ls ile bakıldığında ya da cat ile dosyalar okunduğunda, tam o anda, çekirdek bilgileri üretir ve gönderir. Bundan dolayı her ls komutunda veya her echo komutunda dosya güncelleme zamanları tam komutun işlendiği zamandır.

Kuantum mekaniğinde garip bir tanım vardır. Siz bakana kadar parçacık aslında mevcut değildir, tam baktığınızda orada oluşur. Ya da buna benzer bir ifade vardı.

Sözde dosya sistemleri de tam böyledir. Aslında yokturlar. ls, cat, open, echo vb. komutlar kullanıldığı anda içerikleri çekirdek tarafından üretilir. Bundan dolayı dosya boyları her zaman 0 ve güncelleme zamanları her zaman komutun işlendiği zamandır.

/proc dosya sistemi pek çok bilgi barındırsa da, esas itibari ile çalışan prosesler içindir. /proc kelimesi de processes kelimesinin kısaltmasıdır.

Benzer şekilde bir de donanım bilgilerini tutan /sys dizinine bağlı başka bir sözde dosya sistemi daha vardır. /proc'dan farklı olarak proses değil, donanım bilgilerini tutar. Ayrıca bu iki dosya sistemi bazen birbirlerine link'ler ile atıfta bulunurlar.

/sys sistemi, bordda bulunan bütün cihazların bilgilerini barındırır. İşte "mdev -s" komutu, /sys dosya sistemini gezer ve mevcut cihazlar için /dev dosya sistemini kurar. Tahmin edileceği gibi, mdev sistemi /sys dizinini gezdiği için, "mdev -s" komutundan evvel /sys dosya sistemi bağlı olmalıdır.

6.5 Nihai Test

SD kart cihaza takılıp güç verildiğinde, çok kısa sürede login gelecektir. U-boot sistemi açılış için, uEnv.txt[5] içinde bulunan, uenvcmd komutunu işletir. Bu komut aşağıda verilmiştir.

```
uenvcmd=load mmc 0:1 80007fc0 uImage;    \
        load mmc 0:1 0x80F80000 bbb.dtb; \
        load mmc 0:1 0x88080000 uramfs;  \
        bootm 0x80007FC0 88080000 0x80F80000
```

Katılımcı, her test sırasında, u-boot seviyesinde açılışı durdurmalı ve eunvcmd içindeki komutları el ile tek tek vermelidir. Çok faydasını görecektir.

Daha önce tftp ile ağ üzerinden yüklenen dosyalar artık "load mmc" ile yüklenmektedir. Diğer bir deyişle, bütün mesele bir biçimde ram belleğe bir dosya yüklemektir. Dosyalar ram belleğe yüklendikten sonra bootm komutu benzer şekilde açılışı başlatır. Önceki örneklerden bir farkı yoktur. Her

[5]VFAT sistemlerinde büyük/küçük harf ayrımı yoktur. uEnv.txt ile uenv.txt aynı anlamdadır.

yükleme işinin "mmc 0:1" den yani SD kartın 1. bölümünden yapıldığına dikkat edilmelidir.

Sistem tamamen açıldıktan sonra SD kart yerinden çıkarılabilir. Sistem yine de çalışmaya devam eder. Çünkü kök dosya sistemi artık tamamen RAM içinde kuruludur. Tekrar edersek, RAM içinde kurulan kök dosya sistemine initramfs tekniği denir.

Bölüm 7

Tamamı eMMC'de Olan Sistem

Bu bölümde kurulacak sistem Şekil 7.1'de resmedilmiştir. Daha önce Bölüm 5.1'de kurulan, tamamı MMC'de olan sistem ile tamamen aynıdır. Farklı olarak bütün sistem SD kartta değil, BBB içinde gömülü olan eMMC içinde olacaktır.

BBB için, tamamı eMMC'de olan sistem, ideal bir müşteri sistemidir. Geliştirme veya test sistemi olarak kullanılması tavsiye edilmez. Daha önce belirtildiği gibi, tftp.nfs biçiminde, ideal test sistemi ile kuruluş, geliştirme ve test yapılmalıdır. Sonra bütün sistem eMMC üzerine kolaylıkla aktarılabilir. Mevcut metodolojiniz ile sistemi adım adım inşa edelim.

7.1 Açılış Yükleyicisi

Daha önce kurulan sistemlerin herhangi biri ile BBB açılır. Mevcut eMMC sisteminin bölümlendirme tablosu aşağıdaki gibi incelenebilir.

```
$ cat /proc/partitions

major minor  #blocks  name
 179        0    1921024 mmcblk0     <-- SD (dışarıdan takılan)
 179        1      32768 mmcblk0p1
 179        2      65536 mmcblk0p2

 179        8    1875968 mmcblk1     <-- eMMC (borda sabitlenmiş)
```

```
BBB
MMC
/dev/mmcblk1
+-----------+
|MLO        | p1, VFAT
|u-boot.img |
|uEnv.txt   |
|           |
|uImage     |
|bbb.dtb    |
+-----------+
| /bin      | p2, ext2
| /dev      |
| /etc      |
| /proc     |
| /sys      |
| /tmp      |
| ...       |
|           |
+-----------+
```

Şekil 7.1: Tamamı eMMC'de olan sistem, emmc.emmc

```
 179        9      72261 mmcblk1p1
 179       10    1799280 mmcblk1p2
 179       24       1024 mmcblk1boot1
 179       16       1024 mmcblk1boot0

$ fdisk /dev/mmcblk1

Disk /dev/mmcblk1: 1920 MB, 1920991232 bytes
255 heads, 63 sectors/track, 233 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

        Device Boot      Start         End      Blocks  Id System
/dev/mmcblk1p1   *           1           9       72261   c Win95 FAT32 (LBA)
/dev/mmcblk1p2              10         233     1799280  83 Linux
```

Yukarıdaki bölümlendirme tablosu, eMMC içinde standard gelen bölümlendirme tablosudur. eMMC bu bölümlendirme şekli ile kullanıma uygun değildir. Kök dosya sistemi nadasa bırakılacak kadar büyüktür. Daha ufak bir yerde oturmalıdır. Uygulamaların da farklı bir bölümde bulunması tavsiye edilir. Ayrıca yedek bir çekridek+açılış sisteminin bulunması faydalı olacaktır. Nihayetinde 2GB'lık bir diskin 2 parçadan fazlaya bölünmesi gereklidir.

eMMC'yi katletmeden önce bazı dosyaların yedeklerinin alınması faydalı olacaktır. dd komut ile imaj yedeği bir bütün olarak alınabilir. Ya da NFS üzerinden sadece bazı dosyaların yedeği alınabilir.

NFS tekniğinde, host'un dizini, BBB tarafından mount edilir. BBB'deki dizine yazılan her bilgi otomatik olarak host'a yazılaccaktır. Aşağıdaki örnekte, NFS üzerinden BBB'nin nasıl yedekleneceği verilmiştir.

```
# BBB içindeyken, host'un /tftpboot dizinini bağla.
# BBB tarafında /tmp/nfs'e yazılan her bilgi, doğrudan host tarafında
# /tftpboot altına taşınacaktır.
#
$ mkdir /tmp/nfs
$ ifconfig eth0 192.168.1.100 netmask 255.255.255.0 up
$ mount -t nfs -o nolock 192.168.1.33:/tftpboot /tmp/nfs
```

BBB kendi kurduğumuz sistemle açıldığıdan, eMMC bölümleri otomatik olarak bağlanmamaktadır. Çünkü bağlanması gerekli rcS komutları veya /etc/fstab bilgisi sisteme girilmemiştir. Aşağıdaki gibi eMMC'nin 2 bölümü bağlanabilir. eMMC'nin 2 bölümü olduğunu fdisk ile tespit edilmişti.

```
$ mkdir /tmp/boot
$ mount /dev/mmcblk1p1 /tmp/boot

$ mkdir /tmp/root
$ mount -t ext4 /dev/mmcblk1p2 /tmp/root

$ df

Filesystem            1K-blocks      Used Available Use% Mounted on

/dev/root                 63461     10647     49538  18% /         <-- SD üzerinde oturan kök dosya sistemi.
devtmpfs                 256092         0    256092   0% /dev
tmpfs                    256204         0    256204   0% /dev/shm
tmpfs                    256204         4    256200   0% /tmp      <-- RootFS, RO olduğundan.
tmpfs                    256204        44    256160   0% /var      <-- RootFS, RO olduğundan.
192.168.1.33:/tftpboot 65924864  12594688  49958400  20% /tmp/nfs  <-- Host makinede, uzaktaki bir dizin.
/dev/mmcblk1p1            71133     67748      3386  95% /tmp/boot <-- eMMC'nin boot bölümü.
/dev/mmcblk1p2          1738184   1419852    228368  86% /tmp/root <-- eMMC'nin root bölümü.
```

/tmp/boot ve /tmp/root dizinlerinin içine girilerek mevcut içerikler incelenebilir. Bazı dosyalar, NFS üzerinden aşağıdaki gibi yedeklenebilir.

```
$ mkdir /tmp/nfs/bbb.yedek
$ cp MLO u-boot.img uEnv.txt /tmp/nfs/bbb.yedek/
```

```
$ cp uImage-3.8.13 /tmp/nfs/bbb.yedek/

umount /mnt/boot
umount /mnt/root
```

Hangi dosyaların yedeği alınmalıdır? Elimize bir bord geçtiğinde, ilk yaptığımız iş bazı dosyaların hemen yedeklerini almaktır. Asıl amaç BBB'yi yedeklemek değil, ilk elden işletim sistemi hakkında bilgi toplamaktır. Bu dosyalar veya komut çıkışları aşağıda kısaca incelenmektedir.

1. Boot yedeği mutlaka alınmalıdır. Boot içinde genelde, MLO, u-boot.img veya u-boot.bin, uEnv.txt, zImage, initramfs imajı gibi kök dosya sistemi hariç çok değerli dosyalar bulunmaktadır. zImage içinden kernel config bilgisi elde edilebilir.

   uEnv.txt içinden uenvcmd komutu ve bu komutun çağırdığı değişkenler elde edilebilir. Ya da örnek RAM adresleri bulunabilir.

   Hepsinden önemlisi uEnv.txt'nin nasıl daha profesyonel yazıldığı görülebilir. initramfs imajları loop cihazlar ile mount edilebilir ya da cpio arşivi şeklinde ise cpio komutu ile açılarak dosya sisteminin içeriği incelenebilir. Elimizde acemisi olduğumuz bir bord varsa, sadece bu dosyaların incelenmesiyle dahi, çok hızlı bilgi edinilebilir.

2. Çekirdeğin .config dosyası yedeklenebilir. BBB makinesi, üzerindeki orjinal Linux sistemi ile açılıp, çekirdeğin config dosyası olan /proc/config.gz saklanabilir. Daha sonra çekirdek derlemesi sırasında, gunzip ile bu dosya açılıp, çekirdeğin kaynak kodunun köküne .config adı ile kopyalanabilir. Böylece defconfig ve menuconfig yapmadan doğrudan çekirdek derlemesi yapılabilir. Elde edilen çekirdek, sistemi açan çekirdek ile birebir aynı özellikte olacaktır.

   Config dosyası her zaman /proc içinde olmayabilir. O zaman basit betiklerle çekirdek içinden elde etmek gerekir. Bu dosya çekirdek içinde de olmayabilir. Çünkü derleme sırasında "config dosyasını çekirdeğe ekle" seçeneği kapatılmış olabilir. Bu durumda mevcut çalışan sistemin config dosyası elde edilemez. make defconfig ile örnek .config dosyası, doğrudan çekirdeğin kaynak kodundan elde edilecektir. Müşteriye verilen sistemlerde, az yer kaplasın diye config dosyaları çekirdeğe eklenmez.

3. "$cat /proc/partitions" çıkışı faydalı olacaktır. Sistemdeki bölümlendirme bilgisini verir.

4. df çıktısı da faydalı olacaktır. Bağlı bölümler, bölümlerin tipleri ve kapasite bilgileri faydalı olacaktır.

5. Her cihazın fdisk çıktsını "$ fdisk /dev/mmcblk0" örneğinde gibi saklanmalıdır. Disk geometrisi, boot sektörün başlangıcı, bölüm tipi çok hayati bilgileri bizlere sağlar. BBB'de olmasa da bazı bordlar disk geometrisine veya ilk sektörün başlangıç numarasına aşırı duyarlıdır. Her iş düzgün yapılsa da makine açılmaz. Bu durumda, mevcut fdisk çıkışından kopya çekilebilir.

6. free komutu ile bellek kullanımı bilgisi elde edilmelidir. Tabii ki BBB, üzerindeki orjinal Linux ile açılmalıdır. Daha sonra kendi sistemimiz ile orjinal sistem arasındaki bellek kullanım oranları karşılaştırılabilir. Eğer üzerinde çalıştığımız gömülü sistem, orjinal sistem kadar RAM harcamışsa, boşuna uğraşmız olabiliriz. Hazırlanan sistem projeye özel olduğu için, her zaman daha az RAM harcamalıdır. Ayrıca swap kullanılmamalıdır.

7. /etc dosyası olduğu gibi yedeklenmelidir. Daha sonra wifi, bluetooth, eternet gibi cihazları ve sistemi yapılandırırken buradaki dosyalardan kopya çekilebilir. Ayıp değildir.

8. "$ ifconfig -a" çıkışı da faydalı olacaktır. En azından cihazların MAC adresleri elde edilebilir.

9. Orjinal sistem ile açtıktan sonra "$ ps -ef" çıkışı saklanmalıdır. Çalışan proses sayısı ve proseslerin argümanları açıkça bellidir. Proje için üretilen gömülü sistemde her zaman çok daha az sayıda proses olmalıdır. Orjinal sistemin ps çıktısı incelendiğinde niçin proje bazlı gömülü sistem yapılması gerektiği hemen anlaşılır. Bord ile gelen orjinal dağıtımlarda, projemizle ilgisi olmayan, çer çöp bir sürü çalışan proses olacaktır.

10. Orjinal sistem açılırken üretilen çekirdek mesajları "$ dmesg" komutu[1] ile elde edilmelidir. Özellikle ilk defa üzerinde çalışılan bir sistem için dmesg çıkısı yoğun bir bilgi aktarımı sağlayacaktır. Verilen bilgiler arasında, Linux sürümü, kullanılan derleyici, derlenme zamanı, CPU bilgileri, mevcut donanım, çeşitli adresler, bütün bölümlendirme tabloları, kapasiteler vb bulunmaktadır. En önemli bilgilerden birisi "Kernel command line satırıdır." Bu satır çekirdeğe dışarıdan verilen bütün parametreleri kapsar ve çok değerlidir. Tahmin edileceği gibi bu satır uEnv.txt içinde yazılan "bootargs=..." satırından başka bir şey değildir.

[1] display messages'den üretilmiştir.

11. BBB üzerindeki orjinal sistem, u-boot seviyesinde açılarak, print komutu ile bütün u-boot değişkenleri ve tanımları elde edilmelidir. Burada u-boot'un bütün yetenekleri sergilenir. Çok hızlı açılan bir sistem istenirse uEnv.txt içinde yazılan komutlar doğrudan kaynak koduna kazınabilir. O zaman uEnv.txt dosyasına gerek olmayacaktır. Yani u-boot açıldıktan sonra printenv denirse, kaynak kodunda varsayılan tanımlar olarak girilen tanımlar gelecektir. Fakat bu sistem çok statik olacaktır, tavsiye edilmez.

    Varsayılan u-boot değişkenleri çok geneldir ve hiç bir gömülü sistem bu kadar farklı boot tekniğine aynı anda sahip olamaz. Sahip olsa da böyle bir sistem pratik değildir. Bu eğitimde hiç bahsi geçmeyecek olsa da NAND bölümlendirmesi, NAND ve SPI'dan da boot teknikleri de printenv çıkışından incelenebilir. Bu çıkışta if/else şeklinde koşullu dallanmalar da vardır. Fare[2] ile copy/paste yapılarak veya başka bir yol ile ekran çıktısı yedeklenebilir. Ekran çıktısı katılımcı tarafından mutlaka incelenmelidir. İç yapısı son derece basittir.

12. top komutunun da en az birkaç periyotluk çıktısı elde edilmeli ve saklanmalıdır. top komutu sistem ve proseslerin bütünü hakkında çok ayrıntılı bilgi vermektedir. Daha sonra kendi gömülü sistemimiz ile karşılaştırma yapılabilir. Proje bazlı gömülü sistemin, CPU ve bellek harcaması ile proses adedi orjinal sisteme göre daha az olmalıdır.

13. Gerekli olabilecek başka dosyalar veya çıkışlar ile bu liste uzatılabilir.

Bu bu bölümde mmc.mmc'de yaptılanların neredeyse aynısı yapılacaktır. fdisk ile eMMC 3 bölüme ayrılacaktır. İlk bölüm her zamandaki gibi vfat olacaktır ve u-boot ile çekirdeği taşıyacaktır. İkinci bölüm kök dosya sistemi olacaktır. Üçüncü bölümse çeşni olsun diye kurulacaktır. Bu bölüm read/write bağlanarak kalıcı bilgiler saklanabilir. Kurulacak bölüm sayısı tamamen proje ile ilgilidir.

fdisk ile eMMC'deki mevcut bölümle silinir ve aşağıdaki gibi yeni bölümler kurulur.

```
Device Boot              Start         End      Blocks  Id System
/dev/mmcblk1p1   *           1           9       72261   c Win95 FAT32 (LBA)
/dev/mmcblk1p2              10          18       72292+ 83 Linux
/dev/mmcblk1p3              19          27       72292+ 83 Linux
```

[2]Bu kelimeye çok zor alıştım. Azeriler sıçan diyorlarmış. Sıçan farenin büyüğüdür.

Çok tekrar olduğu için ara adımlar verilmemiştir. Birinci bölüme vfat dosya sistemi kurulmuş ve boot flag aktif hale getirilmiştir. Her bölüm 64MB büyüklüğündedir. fdisk ile çalışırken "w" ile çıkılmadığı sürece yeni bölümlendirme tablosu eMMC'ye yazılmaz.

fdisk programı varsayılan birim olarak sektör kullanır. Fakat -u seçeneği ile silindir birimi de kullanılabilir. Sektör ve silindir arasındaki ilişki, eMMC parametreleri dikkate alınarak, aşağıdaki gibi hesaplanabilir.

```
Bir silindir= 4 head * 16 sektör= 64 sektör.
Her head'de 16 sektör var.
Her sektör 512 bayt.
64 sektör * 512 byte= 32K
Bir silindir= 32 K

İlk silindir numarası 33.
0 halde ilk 32 silindir boş.
32 silindir* 32K= 1M
```

Bazı belgelerde ilk bölümün başlangıcının bazen 33 bazen de 2048 gözükmesinin sebebi kullanılan birimdeki farklılıktır. Boot sektörlerinde bulunan ilk 1M'lık alan açılış yükleyicileri için ayrılmıştır. Bu yüzden genelde ilk bölümler her zaman 1M içerden başlar. Fakat bu bir zorunluluk değildir. fdisk'in expert menüsünden bu değer değiştirilebilir.

BBB'nin içinde gömülü olan ROM açılış yükleyicisi[3] çok yetenekli değildir. Sistemi eMMC'den açabilmesi için bölümlendirme tablosunda ve dosya sisteminde bazı parametrelerin sağlanması gerekir.

1. ilk dosya sisteminin başlangıç ve bitiş silindirleri[4] 1-9 olmalıdır. Bu değeler fdisk ile girilir.
2. İlk bölüm, bootable olarak işaretlenmelidir.
3. Dosya sisteminin FAT tipi 32-bit olmalıdır. mkfs.vfat komutunda -F32 seçeneği ile sağlanır.
4. MLO ilk dosya olmalıdır.

[3]ROM boot loader

[4]Bu şart sağlanamasa da BBB açılmaktadır. Ama bir çok yerde bu silindir uyarısı vardır.

Aşağıdaki gibi, her üç bölüme de dosya sistemleri kurulabilir.

```
$ mkfs.vfat -n BOOT -F32 /dev/mmcblk1p1
$ mkfs.ext2 -L ROOT /dev/mmcblk1p2
$ mkfs.ext2 -L CONF /dev/mmcblk1p3
```

-n ve -L seçenekleri ile "volume label" tanıtılır. Zorunlu bir parametre değildir. -F32 seçeneği ile de vfat sisteminin 32 bitlik FAT tablosu kurması sağlanır.

uEnv.txt dosyası Bölüm 5.1'de verildiği gibi kullanılamaz. Ama 0:1 ifadeleri 1:1 yapılırsa sorunsuz olarak eMMC'den açılış gerçeklenir.

Daha genel olara aşağıdaki gibi güncelleme yapılırsa hem Bölüm 5.1'de hem de bu bölümdeki sistemde hiç değişiklik yapılmadan uEnv.txt dosyası kullanılabilir.

```
bootargs=console=ttyS0,115200n8 root=/dev/mmcblk0p2 ro \
         rootfstype=ext2 rootwait fixrtc

mmc_test=load mmc ${mmcdev}:${mmcpart} 80007fc0 uImage ;    \
         load mmc ${mmcdev}:${mmcpart} 0x80F80000 bbb.dtb ; \
         bootm 0x80007FC0 - 0x80F80000

uenvcmd=run mmc_test
```

uEnv.txt içinde sadece "mmc 0:1" ifadeleri yerine "mmc ${mmcdev}:${mmcpart}" ifadesi getirilmiştir. Açılış sırasında u-boot programı, mevcut mmc değerini mmcdev değişkenine atar. Eğer SD kartı takılı ise mmcdev=0 atanır. Takılı değilse mmc=1 atanır. u-boot açılışı sırasında, print denildiğinde atamaları yapan ilgili satırlar incelenebilir.

mmcpart değeri de her zaman 1'dir. Böylece aynı açılış dosyası hem SD hem de eMMC'den yapılan açılışları destekler.

u-boot ve çekirdeğin Bölüm 7'deki gibi derlendiğini kabul edelim. Açılış için gerekli dosyalar aşağıdaki gibi ilk bölüme kopyalanır.

```
$ mount /dev/mmcblk1p1 /tmp/boot

# MLO, u-boot.img,... dosyaları NFS üzerinden
# BBB'ye kopyalabilir.
```

```
$ cp /tmp/nfs/MLO        /tmp/boot
$ cp /tmp/nfs/u-boot.img /tmp/boot
$ cp /tmp/nfs/uEnv.txt   /tmp/boot

$ cp /tmp/nfs/uImage     /tmp/boot
$ cp /tmp/nfs/bbb.dtb    /tmp/boot

$ df /tmp/boot
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mmcblk0p1           71146      3760     67386   5% /tmp/boot

$ umount /tmp/boot
$ umount /tmp/nfs
```

Test amacı ile SD kart yerinden çıkarılır ve BBB reset edilirse açılış çekirdek seviyesine kadar gelir ve kök dosya sistemi olmadığı için cihaz tıkanır.

7.2 Çekirdek

Bölüm 5.1'de derlenen çekirdek yeniden derleme yapmadan burada kullanılabilir. Zaten Bölüm 7.1'de çekirdek ve dtb doğrudan eMMC'nin ilk bölümüne kopyalanmıştı.

7.3 Kök Dosya Sistemi

Bölüm 5.1'de elde edilen ve RootFS/ dizini altında bulunan kök dosya sistemi, NFS üzerinden veya başka bir yol ile eMMC'nin 2. bölümüne kopyalanır. NFS üzerinden taşıma örnek olarak gösterilecektir.

Bunun için daha önceki bölümlerde elde edilen herhangi bir sistem ile BBB açılır. NFS mount edilir ve "cp -a" komut ile NFS üzerinden RootFS/ içindeki bilgiler eMMC'deki ikinci bölüme aşağıdaki gibi kopyalanabilir.

```
# BBB içindeyiz.
# Uzaktaki /tftpboot dizinini bağla.

$ mkdir /tmp/nfs
```

```
$ mount -t nfs -o nolock 192.168.1.33:/tftpboot /tmp/nfs

# eMMC'deki 2. bölümü bağla.

$ mkdir /tmp/root
$ mount /dev/mmcblk0p2 /tmp/root

# Uzaktaki kök dosya sistemini eMMC'ye kopyala.

$ cp -a /tmp/nfs/RootFS /tmp/root

$ umount /tmp/root
$ umount /tmp/nfs
```

Örnek kök dosya sistemi eMMC üzerine envai yolla taşınabilir. Özellikle az tecrübeli katılımcılar, çok basit olduğu ve akla gelen ilk çözüm olduğu için, taşıma işini SD kart ile yapmaya meyillidirler. Çok zahmetli bir yöntemdir, tavsiye edilmez.

7.4 Açılış Betiği

Bölüm 5.1'de verilen betik burada değiştirilmeden kullanılabilir.

7.5 Nihai Test

Sistemde SD kart varsa yuvasından çekilir. Böylece açılış eMMC üzerinden yapılır. Açılış mantığında hiç bir değişiklik yoktur. Önce MLO sonra u-boot.img yüklenir ve açılış uEnv.txt'deki uenvcmd komutuna göre devam eder.

Açılış tamamen eMMC'den yapılır. Login'e kadar olan bazı satırlar aşağıda verilmiştir.

```
...
VFS: Mounted root (ext2 filesystem) readonly on device 179:2.
devtmpfs: mounted
Freeing init memory: 228K
```

```
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/root                69995      9366     57015  14% /
devtmpfs                256092         0    256092   0% /dev
tmpfs                   256204         0    256204   0% /dev/shm
tmpfs                   256204         0    256204   0% /tmp
tmpfs                   256204        36    256168   0% /var


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


Welcome to UcanLinux
http://ucanlinux.com

UcanLinux login: root
Password: root

root@UcanLinux:~ # df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/root                69995      9366     57015  14% /
devtmpfs                256092         0    256092   0% /dev
tmpfs                   256204         0    256204   0% /dev/shm
tmpfs                   256204         4    256200   0% /tmp
tmpfs                   256204        44    256160   0% /var

root@UcanLinux:~ #
```

/dev/root ile gösterilen cihaz, aslında /proc/cmdline içinde gösterilen root tanımıdır. Örneğimizde root=/dev/mmcblk0p2'dir. Fakat açılışta hem initramfs hem de gerçek kök dosya sistemi varsa, kök dosya sisteminin oturduğu cihaz bu şekilde tespit edilemez. Bunun için aşağıdaki gibi rdev komutu kullanılabilir.

```
$ rdev
/dev/mmcblk0p2 /
```

Çıkıştan da görüleceği gibi "/" ile gösterilen kök dosya sistemi /dev/mmcblk0p2 üzerinde oturmaktadır.

eMMC'den açıldıktan sonraki cihaz isimleri ile MMC'den açıldıktan sonraki cihaz isimleri farklıdır. BBB, eMMC den açılırsa eMMC'nin cihaz numarası 0'dan başlar. Bu durumda bölüm isimleri de aşağıdaki gibi değişecektir.

```
179        0    3833856 mmcblk0
179        1      98304 mmcblk0p1
179        2      31296 mmcblk0p2
179        3    3703232 mmcblk0p3

179       16       1024 mmcblk0boot1
179        8       1024 mmcblk0boot0
```

Açılışı hızlandırmak için console tanımı iptal edilir ve bootdelay 0 yapılırsa[5], açılış süresi, poweron'dan login'e kadar ortalama 3.5 saniyedir.

[5]Bakınız: Ekler 9.42

Bölüm 8

En Genel Gömülü Sistem

Bu bölümde genel amaçlı, dört başı mağrur bir gömülü sistem kurulacaktır. Şu ana kadar kurulan bütün gömülü sistem örneklerinde kök dosya sistemi ve açılış betiği son derece basitti. Tamamı busybox destekli böyle bir gömülü sistem pek çok proje için yeterli olmasına rağmen, kütüphane ve bazı paketler bakımından son derece zayıftır.

Kütüphaneler veya paketler tek tek el ile derlenip sisteme eklenebilir. Açık kaynak kodlu programlar, genelde "$ configure && make && make install" komutları ile derlenir ve kurulurlar. Fakat program sayısı arttıkça veya programların birbirlerine olan bağımlılıkları arttıkça el yordamı ile derlemek bazen çok zahmetli veya imkansız hale gelmektedir.

İşte tam bu anda "build root" projesi veya kısaca BR, imdadımıza yetişir.

BR sistemi, çapraz derleyici, açılış yükleyicisi, çekirdek, kök dosya sistemi ve açılış betikleri ihtiyacının tamamını karşılar. Diğer bir deyişle, bizim metodolojik olarak takip ettiğimiz 4 ayağın tamamını tek bir paket içinde birleştirmişlerdir. Üstelik bu iş yapılırken, sadece betik programlama kullanılmıştır. Diğer bir deyişle, paketlerin indirilmesi, configure edilmesi, bağımlıklıkların bulunması, çapraz derlenmesi, kök dosya sistemine kurulması gibi işlerin tamamı betikler yardımı ile yapılmaktadır.

Şu ana kadar bahsi geçen bütün konular es geçilip sadece BR desteği ile, baştan sona açılan gömülü sistemler kurulabilirdi. Yapılan işin farkındalığını artırmak için BR'nin yapmış olduğu pek çok iş el yordamı ile yapılmıştır. Bu işler kısaca aşağıda listelenmiştir.

1. Çekirdek, u-boot veya busybox gibi programlar internetten, ftp, wget veya git ile indirilmiştir. U-Boot'da aynı şekilde indirir.

2. Çapraz derleyici el yordamı ile kurulmuştur. BR doğrudan bu derleyiciyi kullanabilir veya yeni bir derleyici indirilebilir. BR belgelerinde "derleyiciyi siz indirin biz kullanalım" önerisi yapılmaktadır. Ama tecrübelerimiz göstermiştir ki, derleyici u-boot'a indirtmek her zaman daha az hata ile ilerlemeyi sağlamaktadır. El ile indirilen derleyicilerde, özellikle kütüphanelerin yeri konusunda BR tarafında sorunlar çıkmaktadır.

3. BR, u-boot sistemini indirip, aynen bizlerin yaptığı gibi derleyebilir. Fakat kesinlikle tavsiye edilmez. Geçmiş bölümlerde anlatıldığı gibi u-boot el yordamı ile indirilmeli ve derlenmelidir.

4. BR aynı zamanda çekirdek de indirip derler. Çekirdek derlemesinin de BR tarafında yapılması hiç ama hiç tavsiye edilmez.

5. Aynen bizim önceki örneklerde oludğu gibi BR sistemi örnek bir kök dosya sistemi iskeleti kullanır. BR'nin kök dosya sisteminin kullanılması tavsiye edilir. Bizim örnek kök dosya sistemi de zaten BR'nin sistemine çok benzemektedir. BR sistemine ait iskelet sistem $BR_SRC/system/skeleton altında incelenebilir.

6. BR sistemi kök dosya sistemi kurmadan önce ve sonra istenirse özel bir betik işletebilir. Bu şekilde bir kullanım tarzı hiç tavsiye edilmez. Hiç pratik değildir. Bunun yerine BR, kök dosya sistemini kurduktan sonra yazılan bir betik gerekli güncellemeleri yapmalıdır.

7. BR sistemi envai çeşit kök dosya sistemi formatı kullanır. ext2, ext3, jffs2, ubifs vs. Bunların hiç biri kullanılmamalı bunun yerine sadece tar şeklinde kök dosya sistemi kurulmalıdır. Böylece bu dosya sistemi, RootFS/ gibi bir dizin altına açılarak, buraya kadar yapılan bütün gömülü sistemlerde kolaylıkla kullanılabilir.

8. Onu kullanma, bunu kullanma dedik, ne kullanılacaktır? BR, sadece kök dosya sisteminin tar halini elde etmek için kullanılacaktır. Kök dosya sistemi içinde çok basit bir açılış mekanizması da mevcut olacaktır. Özetle, u-boot ve çekirdek el yordamı ile derlenecek, kök dosya sistemi ve açılış betiği BR yardımı ile elde edilecektir.

BR sistemi ile elde edilen kök dosya sistemi her zaman çok daha fazla yer kaplar. Busybox destekli bir kuruluş, gömülü sistem projesinde işimizi görüyorsa,

BR sistemi kullanılmamalıdır.

Bu bölümde kurulacak gömülü sistem için herhangi bir şekil çizilmemiştir. Zaten şekiller de pek dandik durmaktaydı.

BR ile elde edilen kök dosya sistemi, mevcut bütün projelerde, hatta initramfs destekli projelerde dahi rahatlıkla kullanılabilir. Örnek olması için kök dosya sistemi ve açılış betikleri, Bölüm 4'de verilen ideal test sistemi üzerinde kullanılacaktır. Katılımcı iki sistemin kuruluşu arasında, içerik hariç hiç bir fark olmadığını görecektir.

Yeni kök dosya sisteminde maalesef, Qt ve GUI mevcut olacaktır. Piyasada, bu konuda inanılmaz bir talep vardır.

8.1 Açılış Yükleyicisi

Bölüm 4'de bahsediliği gibi derlenir ve SD üzerine kurulur Bu kısım da BR yardımı ile elde edilebilir. Ama hiç tavsiye edilmez.

8.2 Çekirdek

Bölüm 4'de bahsediliği gibi derlenir ve /tftpboot altında çekirdek ve dtb dosyası kopyalanır. Çekirdek de BR yardımı ile elde edilebilir. Ama zerre kadar tavsiye edilmez.

8.3 Kök Dosya Sistemi

Esas işimiz burada başlamaktadır. BR ile kök dosya sistemi kurulacaktır. Aslında BR, kök dosya sistemi kuruluşu dışında da kolaylıkla kullanılabilir.

Örneğin sadece git server kuracak olalım. BR içinde sadece php ve git'i seçmek yeterlidir. Sonra kendi iskelet kök dosya sistemine elde edilen dosyalar kopyalanabilir.

Ya da libxml kütüphanesi gerekli oldu. Sadece bu kütüphane BR[1] ile derlenebilir. Özetle, bütün bir kök dosya sistemi kurmak yerine tek tek paket de derlenebilir.

BR sistemi git ile aşağıdaki gibi indirilir.

```
# br adı ile klon yap.
#
$ git clone git://git.buildroot.net/buildroot br
```

BR'nin kaynak kodu her zaman git ile elde edilmelidir. Böylece arada bir pull ile kolaylıkla güncelleme yapılabilir. Ayrıca aynı anda pek çok BR sürümü de gelir. Bazen eski sürümlere dönmek gerekli olabilir. Bu durumda sadece git chkecout komutu ile başka bir sürüme anında geçilebilir. git'ten şaşmamak gerekir.

Çekirdek ve Busybox için kullanılan derleme ortamının aynısı BR için de kullanılmıştır. Genel mantık hep aynıdır. Örnek bir config dosyası .config olarak köke kopyalanır. make menuconfig ve make ile derleme yapılır. Derleme sırasında aşağıdaki seçimlere dikkat edilmelidir.

1. u-boot ve çekirdek derlemesi boş geçilmelidir. Bu derlemeler el yordamı ile yapılmalıdır.

2. İskelet dosya sistemi olarak, BR'nin varsayılan kök dosya sistemi kullanılmalıdır.

3. Kök dosya sisteminin formatı tar seçilmelidir. Üretilen diğer dosya sistemleri üzerinde güncelleme yapmak "tar" kadar rahat değildir.

4. pre ve post script girişleri boş bırakılmalıdır. Kök dosya sistemi üzerinde, tar oluştuktan sonra güncelleme yapılmalıdır.

5. Bazı paketler derlenirken kütüphanelerle ilgili sorunlar çıkabilmektedir. Bir çözüm olarak çapraz derleyicinin BR tarafından kurulması sağlanabilir.

6. c-cache kullanılmamalıdır. C-cache mekanizması, kaynak koddaki güncellemeleri fark edemez. Hatalı derlemeler üretebilir.

[1]Kaynak: https://buildroot.uclibc.org/downloads/manual/manual.html

Daha önce belirtildiği gibi tek paketler de derlenebilir. Aşağıda zlib paketinin, native olarak, tek başına nasıl derleneceği örneklenmiştir. Çapraz derleme için CROSS_COMPILE değişkeninin make komutunda tanımlanması yeterlidir.

```
$ cd out/br/build           # Derlenmeye hazır paket buradadır.
$ cp -a zlib-1.2.8/ /tmp    # Orjinal sistemi bozmamak için dışarı al.
$ cd /tmp
$ cd zlib-1.2.8/
$ ./configure --prefix=/tmp/test # Paket /tmp/test altına kurulacak.
$ make
$ make install
$ cd ..
$ cd test         # Her şey burada.
```

BR ile kök dosya sistemi kurabilmek için öncelikle örnek bir config dosyası yardımı ile menuconfig işletilir ve uygun paketler seçilir. Qt için de seçim yapılabilir. Bir paket seçildiği zaman o paketin bağlı olduğu bütün paket ve kütüphaneler de otomatik olarak seçilir. Aşağıda menuconfig işleminin aşamaları verilmiştir.

```
# Her sıfır kuruluşta clean yapılmalı, yoksa eskiler silinmiyor.

# menuconfig
$ make -C $BR_SRC O=$BR_OUT clean
$ make -C $BR_SRC O=$BR_OUT list-defconfigs
$ make -C $BR_SRC O=$BR_OUT beaglebone_defconfig
$ make -C $BR_SRC O=$BR_OUT menuconfig
```

menuconfig işlemi bittiğinde .config dosyası kurulmuş olur. Bu dosya içinde seçilen paketler ve derleme için gerekli değişkenler mevcuttur. menuconfig bittikten sonra aşağıdaki gibi derleme aşamasına geçilir.

```
$ T1=‘date‘
$ make -C $BR_SRC O=$BR_OUT -j2
$ T2=‘date‘

$ echo "başlangıç zamanı: $T1"
$ echo "bitiş zamanı: $T2"
```

Seçilen paketlerle orantılı olarak BR'nin derlenmesi inanılmaz uzun sürebilir. Özellikle X ve Qt derlenecekse çok sabırlı olmak gerekir. Ayrıca ilk derlemenin hatasız olması çok nadir bir durumdur, illa ki bir hata çıkacaktır. Muhtemelen ilk çıkacak hatalar, host makinede bulunan eksik paketlerdir.

Eğer -jN ile derleme yapılıyorsa ve N>1 verilmişse, hata mesajları bazen birkaç sayfa geride bulunabilir. N>1 verildiği için derleme birden fazla thread ile yapılır. Bu durumda bazen hatasız ilerleyen thread'in, hata veren thread yüzünden durması zaman almakta ve hatasız thread'in mesajları, hata mesajlarından sonra çıkabilmektedir. Bundan dolayı BR derlemesi hatalı biter ve ekranda hiç hata mesajı gözükmezse, Shift+PageUP veya faredeki scroll düğmesi ile birkaç sayfa geri gidilmeli ve hata mesajları aranmalıdır.

Yukarıdaki örnekte derlemenin başlangıç ve bitiş zamanı bilgi amacı ile yazılmıştır. Süre inanılmaz uzundur, ilk derlemenin gece yatmadan önce verilmesi tavsiye edilir.

Sadece Qt4'ün derlenmesi 25 dakika sürmektedir. Tabii benim makine biraz eski, ondan da süre uzamaktadır.

İlk defa derleme yapıldığında, derlenecek bütün programların kaynak kodları tek tek otomatik olarak indirilir. Yüksek hızlı bir ağ ortamında bu işin yapılması tavsiye edilir. İndirilen bütün dosyalar dl/ dizini[2] altına atılır.

Her program derlenmeden önce açılır ve sonra çapraz derlenir. Tabii ki en önce çapraz derleyici derlenir. Sonra paketler derlenir.

Derleme bitince $BR_OUT/images/ altında rootfs.tar adlı, nur topu gibi bir dosya sistemi üretilmiş olacaktır. Bu dosya sistemi tar formatındadır ve doğrudan /tftpboot/RootFS dizini altına açılmalıdır. Tabii ki NFS destekli kök dosya sistemi için bu geçerlidir. Katılımcı elindeki RootFS'i geçmiş bütün gömülü sistem örneklerinde kullanabilir.

BR'de derleme bittikten sonra bazen tek bir paketin yeniden derlenmesi istenebilir. Tek bir paketin derlenmesi için BR aşağıdaki gibi zorlanabilir.

```
$ make -C $BR_SRC O=$BR_OUT ${PACKAGE}-dirclean
$ make -C $BR_SRC O=$BR_OUT -j2
```

Daha sonra make yapıldığında sadece bu paket derlenecektir. Bu arada BR'de paket silme kavramı yoktur. menuconfig sırasında seçilen bir paket daha sonra yine menuconfig yapılarak seçim iptal edilirse, bu paket kök dosya sisteminden çıkarılmaz. Paket el ile silinmelidir. BR'nin kılavuz sayfalarında bu konuda çok faydalı bilgiler bulunabilir.

[2]dl: download

Bazen busybox da tek başına derlenmek istenebilir. BR, kök dosya sistemi kurarken busybox sistemini de arka planda derler fakat kullanıcıya busybox'ın menuconfig sistemini göstermez. Diğerbir deyişle, BR'nin menuconfig girişlerinde yapılan seçimlere göre arka planda busybox'ın da seçimleri otomatik olarak yapılır. Kullanıcının bundan haberi olmaz. Ama yine de aşağıdaki gibi busybox tek başına derlenebilir.

```
$ make -C $BR_SRC O=$BR_OUT busybox-menuconfig
$ make -C $BR_SRC O=$BR_OUT -j2
```

Derleme bittikten sonra BR'nin ürettiği kök dosya sistemi aşağıdaki gibi /tftpboot altında kopyalanır.

```
$ export ROOT_FS=$BR_OUT/images/rootfs

# Garanti olması için varsa eski RootFS'i sil.
#
$ rm -fr /tftpboot/RootFS
$ mkdir /tftpboot/RootFS

# BR'nin RootFS'ini /tftp altına aç.
#
$ tar -C /tftpboot/RootFS -xvf ${ROOT_FS}.tar

# Açılış için gerekli dosyaları kopyala.
#

# Qt örneğini kopyala.
#
$ cp hello /tftpboot/RootFS/root/hello

# Son güncellemeleri yap.
#
$ ./update.rootfs
```

Ayrıca daha önceki bölümlerde bahsedildiği gibi, çekirdek modülleri ve firwmare dosyaları, çekirdeğin kaynak kodu içinde iken modules_install ve firmware_install girişleri ile yeni üretilen kök dosya sistemine yüklenir. Burada INSTALL_MOD_PATH olarak /tftpboot/RootFS'in verileceği aşikardır.

Yukarıdak "tar -C" satırına kadar yapılanlar açık, net ve sıradandır. BR'nin kök dosya sistemi /tftpboot/RootFS/ altına açılarak kopyalanmaktadır.

Qt programı RootFS/root altına kopyalanır. Böylece BBB açıldıktan sonra hello programı test amacı ile çalıştırılacaktır.

Daha sonra update.rootfs betiği çalıştırılmakta ve /tftpboot/RootFS altında çeşitli güncellemeler yapmaktadır. Bu betiğin bir kısmı, özet şeklinde aşağıda verilmiştir, eksiksiz değildir.

```
#!/bin/sh +x
#
# Bu betik, /tftpboot/RootFS altına kurulan sistemi özelleştirir.
# RootFS genelde BR ile kurulmuştur.
# Bu tür betikleri BR içinde çalıştırmak çok kullanışsızdır.
# Bundan dolayı ayrıca işletilmektedir.
#

TFTP_ROOTFS="/tftpboot/RootFS"

# /etc/securetty sonuna, telnet ile uzaktan bağlanabilmek için
# pts/0 pts/1 pts/2 ekle.
#
secure_tty(){
echo "ttySAC0\nttyO0\npts/0\npts/1\npts/2\n" > $TFTP_ROOTFS/etc/securetty
}

root_user(){
cd $TFTP_ROOTFS/root
rm -f .bash* .ash*
ln -f -s /tmp/ash_history .ash_history

# profile güncelle.
#
cat <<EOF >> $TFTP_ROOTFS/etc/profile

PS1="\u@\h:\w \$PS1"
export PS1

alias ls="ls --color"

echo
df
echo

ifconfig eth0 | head -n2
echo

uname -a
echo
```

```
EOF

# ssh'da root girişine izine ver.
#
printf "\nPermitRootLogin yes\n" >> $TFTP_ROOTFS/etc/ssh/sshd_config

# /home yarat.
#
mkdir $TFTP_ROOTFS/home

# DEBUG için strace kopyala.
#
cp $PROJECT_HOME/bin/strace $TFTP_ROOTFS/sbin
}

issue(){
figlet -C utf8 UCanLinux > $TFTP_ROOTFS/etc/issue
sed -i 's/\\/\\\\/g' $TFTP_ROOTFS/etc/issue
echo "\nWelcome to UcanLinux\nhttp://ucanlinux.com\n" >> $TFTP_ROOTFS/etc/issue
}

version(){
echo "UCANLINUX_ID=`date +%F-%H-%M`">> $TFTP_ROOTFS/etc/os-release
}

rm_unneeded(){
rm -f $TFTP_ROOTFS/linuxrc
}

secure_tty
root_user
issue
version
rm_unneeded
```

Bu betiğin içeriği tamamen projeye bağlıdır. Felsefesinin bilinmesi yeterlidir. İstenirse bu veya benzeri betikler, BR girişinde ayrıca tanıtılabilir. Bu durumda rootfs kurulmadan önce ve kurulduktan sonra, betiklerin otomatik olarak BR tarafından çalıştırılması sağlanabilir. Kulağa hoş gelse de kesinlikle pratik değildir. rootfs kurulduktan sonra dışarıda çalıştırmak daha mantıklıdır. Katılımcı aynı betiği BR içine de ekleyerek kullanabilir. Sadece betiğin path bilgisini vermesi yeterlidir.

Qt tarafında yazılan "hello world" programı[3] aşağıda verilmiştir.

```
/* main.cpp */
#include <QtGui/QApplication>
#include <QtGui/QPushButton>

int main(int argc, char **argv){

   QApplication app (argc, argv);

   QPushButton button ("Hello world !");
   button.show();

   return app.exec();
}
```

Qt programını çapraz derlemek için öncelikle qmake hazırlamak gerekir[4]

```
# Önce toolchain için qmake yapılmalıdır.
# Bunun için mevcut olanın kopyasını al ve qmake.conf içine girerek
# gnueabi yerlerine gnueabihf yaz.
# Örnek:
$ cd out/br/host/usr/arm-ucanlinux-linux-gnueabihf/sysroot/usr/mkspecs
$ cp -a linux-arm-gnueabi-g++ linux-arm-gnueabihf-g++
$ vi qmake.conf
```

qmake ömür boyu bir kez yapılır. main.cpp programı aşağıdaki gibi derlenir.

```
QMAKE=$BR_OUT/host/usr/bin/qmake

# .pro kur.
#
$QMAKE -project

# Makefile kur.
#
$QMAKE -spec linux-arm-gnueabihf-g++

# Makefile'ı toolchain'e göre güncelle.
#
```

[3] Bütün yaşam enerjimin çekildiğini hissettim!

[4] İnternette ilk bulduğumuz örneği burada veriyoruz. İlla ki daha pratik yolları vardır. Ayrıca eclipse gibi ortamlarda bu işleri yapmak daha kolaymış.

```
$QMAKE

# Derle.
#
make
```

Kök dosya sistemimiz artık kullanıma hazırdır. Üstelik Qt ile yazılmış bir kodumuz da mevcuttur.

BR'nin .config dosyası mutlaka ayrı bir yerde saklanmalıdır. Şu ana kadar kernel, busybox ve buildroot için .config dosyaları elde edilmiştir. Bütün .config dosyaları mutlaka ayrı bir ortamda saklanmalıdır.

8.4 Açılış Betiği

BR sistemi, busybox ile kurulan basit bir kök dosya sistemine sahip değildir. Bundan dolayı açılış betikleri farklı bir mantıkla çalışır. Öncelikle esas açılış betiği /etc/inittab içindedir. Bizim daha önceki örneklerde /etc/rcS içinde yaptığımız bazı komutlar burada /etc/inittab içinde bulunur[5]

BR tarafından sysV tekniği ile kurulan bir /etc/inittab örneği aşağıda verilmiştir.

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

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

sole::respawn:/sbin/getty -L  console 0 vt100 # GENERIC_SERIAL
```

[5] Bu kullanım tarzı çok kötüdür. Hata durumlarında yapılabilecek bir iş yoktur. Bizler sistem oturduktan sonra bu satırları /etc/rcS içine almaktayız.

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

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

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

si? satırları sıra ile işlenir. si6 satırında da görüleceği gibi açılışı esas olarak /etc/init.d/rcS[6] betiği yönetir. Bu betik aşağıda verilmiştir.

```
#!/bin/sh
# /etc/init.d/rcS

# Start all init scripts in /etc/init.d
# executing them in numerical order.
#
for i in /etc/init.d/S??* ;do

     # Ignore dangling symlinks (if any).
     [ ! -f "$i" ] && continue

     case "$i" in
        *.sh)
            # Source shell script for speed.
            (
                trap - INT QUIT TSTP
                set start
                . $i
            )
            ;;
        *)
            # No sh extension, so fork subprocess.
            $i start
            ;;
    esac
done
```

Bu betik, /etc/init.d altında bulunan ve S?? ile başlayan bütün betikleri start argümanı ile çağırır. Çağırma işi sıra numarasına göre yapılır. Çünkü S??*

[6] rcS, "run command script" kelimelerinden üretilmiş olabilir.

ifadesi, küçükten büyüğe doğru dosya adlarını elde eder. S??* dosyalarını barındıran /etc/init.d/ dizini aşağıda listelenmiştir.

```
S01logging
S20urandom
S30dbus
S30rpcbind
S40network
S45ifplugd
S50sshd
S60nfs
```

Listeleme çıkışı da ufaktan büyüğe doğrudur. Bu durumda en önce S01logging betiği ve en sona da S60nfs betiği çalışacaktır. Bütün S??* betiklerinin ana yapısı aşağıda verilmiştir.

```
#!/bin/sh

case "$1" in
        start)
                echo -n "Starting logging: "
                # PROGRAM BURADA ÇALIŞTIRILIR...
                echo "OK"
                ;;
        stop)
                echo -n "Stopping logging: "
                # KILL İLE PROGRAM BURADA DURDURULUR...
                echo "OK"
                ;;
        restart|reload)
                $0 stop
                $0 start
                ;;
        *)
                echo "Usage: $0 {start|stop|restart}"
                exit 1
esac

exit $?
```

/etc/inittab dosyası incelenirse, kapanıştan önce rcK[7] betiği işletilir. Bu betiğin mantığı rcS ile tamamen aynıdır. Kapanışı, ters sırada yapmak için "ls -r" komutu ile S??* dosyaları ters sırada elde edilir. Böylece en son işleyen program ilk kapanacaktır.

[7] rcK, "run command kill" olabilir.

Eğer bir betik S?? ile başlıyor ve sh ile bitiyorsa, doğrudan "." ile işletilir. Yani S??*.sh betiği mevcut shell içinde gömülü olarak işletilir. Aynı zamanda start/stop argümanı kullanılmaz. Genelde S??*.sh betiği /etc/init.d/ içinde bulunmaz.

Aslında bu tür bir açılış sistemi son derece anlamsızdır. Çünkü pek çok yerde echo komutu vardır, ama gömülü sistem projelerinde pek nadir monitör bulunur. Ayrıca rcK betiğine ve S??* betiklerindeki stop girişlerine de hemen hemen hiç gerek yoktur. Çünkü hemen hemen hiç bir gömülü sistem poweroff veya halt komutu ile kapatılmaz, doğrudan düğmesinden kapatılır veya fişi çekilir.

Bir de init.d/ altındaki betiklere hiç gerek yoktur. Gömülü sistem bir kere oturduktan sonra, S??* betiklerinde bulunan start girişleri /etc/init.d/rcS veya /etc/rcS içine, yukarıdan aşağıya sıra ile yazılabilir. Bu yazım şekli açılışı da hızlandıracaktır.

Çok karışık betiklerin rcS içine alınmasına gerek yoktur. start girişinde bir veya iki satır barındıran betikleri de ayrı bir dosya olarak tutmanın bir anlamı yoktur.

Tabii ki bütün bu öneriler gömülü sistem projesinin son aşamasında yapılmalıdır ki sistem nasıl açılıyor, ne olup bitiyor görelim. Bu öneriler şahsi tecrübelerimizdir. Her katılımcı zamanla kendi çözümünü üretecektir.

8.5 Nihai Test

Her adım düzgün yapılmışsa, MMC kartı yuvasına takılır ve sistem tftp/NFS yardımı ile açılır. BBB'de LCD ekran olamayabilir. Bu durumda X server kullanılarak, BBB'de çalışan Qt kodunun ekran çıktısı host makinede görülebilir. Çeşitli test teknikleri aşağıda verilmiştir.

```
# Uzaktaki X Server ile test
#
# Ubuntu 14.04'de X server, listen modunda gelmez.
# Kanyak: https://sandalov.org/blog/2024/
# $ ps -aux | grep X
# -nolisten gözükür.
# Bunu listen modunda geçirmek için
#
```

```
# $ vi /usr/share/lightdm/lightdm.conf.d/50-xserver-command.conf
# [SeatDefaults]
# Dump core
# xserver-command=X -core
# xserver-allow-tcp=true  # <-- ekle.
#
# Sonra restart
# Kontrol için:
$ telnet 127.0.0.1 6000

# Qt programından önce, test amacı ile xclock programı
# test amacı ile çalıştırılabilir.
# BR'de xclock seçilmiş olmalıdır.
#
# xlock'ın uzaktan çalıştırılması:
# BBB tarafında:
$ export DISPLAY=192.168.1.25:0.0
$ xclock

# Host tarafında:
# $ xhost +

# Örnek:
# BBB tarafında aşağıdaki komutu gir.
$ ./hello -display 192.168.1.90:0

# Ya da...
# $ export DISPLAY=192.168.1.90:0
$ ./hello

# qt'nin remote X ortamında çalışabilmesi için
# BR'de "Qt standard (X11)" seçilmelidir.
#
# Kaynak: https://github.com/pbouda/buildroot-qt-dev
```

emmc.emmc sisteminde, yeni tamamı ile eMMC'de bulunan sistemde /dev/mmcblk0p2 bölümünde, yani ikinci bölümde bir kök dosya sistemi mevcut idi. br ile elde edilen kök dosya sistemi ikinci kök dosya sistemi olarak eMMC'ye kurulabilir. Bu durumda eMMC'de 2 kök dosya sistemi olacaktır.

Busybox ile kurulan birinci kök dosya sistemi 2. bölümde oturmaktadır. br ile kurulan kök dosya sistemi ise 3. bölümde oturabilir. u-boot parametreleri uygun şekilde değiştirilerek her iki kök dosya sistemi ile de açılış yapılabilir.

BR ile kurulan kök dosya sisteminin, almaşık[8] bir kök dosya sistemi olarak

[8] almaşık: alternatif

kuruluşu aşağıda adım adım verilmektedir. Fazla sayfa kaplamasın diye bazı çıkışlar kırpılmıştır.

```
# emmc.emmc ile açılış yapılır.
# rdev ile kök dosya sisteminin 2. bölümde oturduğunda emin olunur.

root@UcanLinux:~ # rdev
/dev/mmcblk0p2 /

# Kök dosya sisteminin oturacağı, 256MB'lık 3. disk bölümünü kur.

root@UcanLinux:~ # fdisk -u /dev/mmcblk0

Command (m for help): p

        Device Boot      Start         End      Blocks  Id System
/dev/mmcblk0p1   *        2048      198655       98304   e Win95 FAT16 (LBA)
/dev/mmcblk0p2          198656      261247       31296  83 Linux

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 3

First sector (16-7667711, default 16): 261248

Last sector or +size or +sizeM or +sizeK (261248-7667711, default 7667711): +256M

Command (m for help): p

        Device Boot      Start         End      Blocks  Id System
/dev/mmcblk0p1   *        2048      198655       98304   e Win95 FAT16 (LBA)
/dev/mmcblk0p2          198656      261247       31296  83 Linux
/dev/mmcblk0p3          261248      761248      250000+ 83 Linux

Command (m for help): w

root@UcanLinux:~ # sync

root@UcanLinux:~ # reboot


# Yeni bölüm, reboot işleminden sonra kernel tarafından görülecektir.
# Eğer görülmüyorsa, fdisk programından "w" ile çıkılmamış veya
# sync yapılması unutulmuştur.
#
```

```
# partprobe veya parted gibi programlar ile reboot etmeden de çekirdek
# yeni bölümlendirme tablosundan haberdar edilebilir.
#
# Açılışta, aşağıdaki gibi bir çekirdek mesajı geçer.
#  mmcblk0: p1 p2 p3
#
# Yani yeni bölümlendirme tablosu çekirdek tarafından okunmuştur.
# Açılıştan sonra dmesg|grep mmcblk0 ile de inceleme yapılabilir.
#
# Cihaz isimlerini kontrol et.
#

root@UcanLinux:~ # ls -l /dev/mmcblk0p?
brw-------    1 root     root      179,   1 Jan  1  1970 /dev/mmcblk0p1
brw-------    1 root     root      179,   2 Jan  1  1970 /dev/mmcblk0p2
brw-------    1 root     root      179,   3 Jan  1  1970 /dev/mmcblk0p3

# 3. bölüme dosya sistemi kur.
#

root@UcanLinux:~ # mkfs.ext2 /dev/mmcblk0p3

# Sıfır kilometre, gıcır gıcır dosya sistemini /tmp/root'a bağla.
#

root@UcanLinux:~ # mount -t ext2 /dev/mmcblk0p3 /tmp/root

# Cihaz ismi ve bağlantı noktasını kontrol et.
#

root@UcanLinux:~ # df /tmp/root
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mmcblk0p3          242078        13    229565   0% /tmp/root

# 256MB kapasitesi olan bölüme, ext2 kurunca, boş alan aşağ yukarı
# 229MB kalmıştır. Aradaki farkı, ext2 dosya sisteminin alt yapısı yemiştir.
#
# Şimdi BR ile derlenen rootfs.tar dosyası bu bölüme açılmalıdır.
# rootfs.tar dosyasını taşımak yerine aşağıdaki gibi NFS
# bağlantısı ile kuruluş yapılabilir.
#
# PC'nin diskindeki /tftpboot dizinini buradaki /tmp/nfs'e bağla.
# PC tarafında /tftpboot altına yazılan her dosya, BBB tarafında
# /tmp/nfs dizini altında görülecektir.
#
# Tabii ki PC tarafında, /tftpboot dizininin paylaşıldığını, yani export
# edildiğini kabul ediyoruz.
#
```

```
root@UcanLinux:~ # mkdir /tmp/nfs
root@UcanLinux:~ # mount -t nfs -o nolock 192.168.1.33:/tftpboot /tmp/nfs

# Bağlantıyı kontrol et.
#

root@UcanLinux:~ # df

Filesystem             1K-blocks      Used Available Use% Mounted on
...
/dev/mmcblk0p3            242078        13    229565   0% /tmp/root
192.168.1.33:/tftpboot  57541632  16881792  37713792  31% /tmp/nfs


# PC tarafında, rootfs.tar dosyasını /tftpboot'a kopyala.
#

$ cp $BR_OUT/images/rootfs.tar /tftpboot

# BBB tarafında rootfs.tar artık gözükür.
# Kontrol et.
#

root@UcanLinux:~ # ls -l /tmp/nfs/*.tar
-rw-r--r--    1 sshduser sshduser 124487680 Apr 28  2016 /tmp/nfs/rootfs.tar

# BBB'deki yeni disk bölümüne rootfs.tar'ı aç.
#

root@UcanLinux:~ # cd /tmp/root

# Dosya sistemlerinde her zaman lost+found dizini bulunur.
# Ani kapanma gibi durumlarda sahipsiz kalan dosyalar, fsck
# tarafından tespit edilir ve bu dizinde biriktirilir.
# Bu dizini silmeyin.
#

root@UcanLinux:/tmp/root # ls -l
total 12
drwxr-xr-x    2 root     root         12288 Jan  1 01:54 lost+found

root@UcanLinux:/tmp/root # tar -xf /tmp/nfs/rootfs.tar

root@UcanLinux:/tmp/root # df .
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mmcblk0p3          242078    122020    107558  53% /tmp/root

# BR ile kurulan dosya sistemi kök dosya sisteminin ortalama
# yarısını işgal eder.
```

```
# RO tipli dosya sistemlerinin %100'e yakın dolu olmasının bir sakıncası yoktur.
# Tabii ki güncellemeler için gerekli yer hesaba katılmalıdır.
#

root@UcanLinux:/tmp/root # cd /tmp/root

# Yeni kök dosya sistemini incele.
#
# bin/ altında, busybox ile diğer programların aynı
# anda kullanıldığına dikkat et.
#
# Çekirdek derlemesi sırasında elde edilen modules
# ve firmware dizinlerinin de aktarıldığını kabul edelim.
#

root@UcanLinux:/tmp/root # ls -l

root@UcanLinux:/tmp/root # cd bin
root@UcanLinux:/tmp/root # ...

# Gereksiz dosyayı sil.
#
root@UcanLinux:/tmp/root # rm /tmp/root/linuxrc

# Bağlı dizinleri kopar.
#
root@UcanLinux:/tmp/root/bin # cd
root@UcanLinux:~ # umount /tmp/root
root@UcanLinux:~ # umount /tmp/nfs

# uEnv.txt'yi güncelle.
# Sadece root=... ifadesinin güncellendiğine dikkat edelim.
#

root@UcanLinux:~ # mkdir /tmp/boot

root@UcanLinux:~ # mount /dev/mmcblk0p1 /tmp/boot

root@UcanLinux:~ # df
Filesystem           1K-blocks      Used Available Use% Mounted on
...
/dev/mmcblk0p1           96788      5187     91602   5% /tmp/boot

root@UcanLinux:~ # cd /tmp/boot

root@UcanLinux:/tmp/boot # ls
bbb.dtb     mlo         u-boot.img  uenv.txt    uimage

root@UcanLinux:/tmp/boot # vi uenv.txt
```

```
root@UcanLinux:/tmp/boot # cat uenv.txt
bootargs=console=ttyS0,115200n8 root=/dev/mmcblk0p2 ro \
                 rootfstype=ext2 rootwait fixrtc

mmc_test=load mmc ${mmcdev}:${mmcpart} 80007fc0 uImage ; \
         load mmc ${mmcdev}:${mmcpart} 0x80F80000 bbb.dtb ; \
         bootm 0x80007FC0 - 0x80F80000

uenvcmd=run mmc_test

br_bootargs=setenv bootargs console=ttyS0,115200n8 \
         root=/dev/mmcblk0p3 ro rootfstype=ext2 rootwait fixrtc

br_test=run br_bootargs ; \
        load mmc 1:1 80007fc0 uImage ; \
        load mmc 1:1 0x80F80000 bbb.dtb ; \
        bootm 0x80007FC0 - 0x80F80000

root@UcanLinux:/tmp/boot # cd

root@UcanLinux:~ # umount /tmp/boot

root@UcanLinux:~ # sync

root@UcanLinux:~ # reboot

# Açılış u-boot seviyesinde durdur.
#
# uEnv.txt içindeki değişkenleri içe aktar.
# Aslında copy/paste ile de yapılabilir.
# Örnek olması için uzun uzun yazılmıştır.
#

Hit any key to stop autoboot:  0

=> ls mmc 1   <-- Linux'un 0 dediğine, u-boot 1 der.
    66268   mlo
    60627   bbb.dtb
   318824   u-boot.img
      469   uenv.txt
  4863480   uimage

5 file(s), 0 dir(s)

=> load mmc 1 82000000 uenv.txt
reading uenv.txt
469 bytes read in 4 ms (114.3 KiB/s)
```

```
=> env import -t -r 82000000 $filesize

=> print br_bootargs
br_bootargs=setenv bootargs console=ttyS0,115200n8 \
      root=/dev/mmcblk0p3 ro rootfstype=ext2 rootwait fixrtc

=> print br_test
br_test=run br_bootargs ; \
        load mmc 1:1 80007fc0 uImage ; \
        load mmc 1:1 0x80F80000 bbb.dtb ; \
        bootm 0x80007FC0 - 0x80F80000

=> run br_test
reading uImage
4863480 bytes read in 310 ms (15 MiB/s)

reading bbb.dtb
60627 bytes read in 11 ms (5.3 MiB/s)

## Booting kernel from Legacy Image at 80007fc0 ...
..

Login gelir.

# GUI programları için bir örnek.
# Bu örnekte xclock isimli saat programı çalıştırılır.
# Görüntü ağ üzerinden PC'de çıkacaktır.
#
# PC tarafında X server çalışıyor kabul edelim.
# BBB'nin IP değeri 192.168.1.100 olsun.
# PC'de aşağıdaki komut girilir.
# Bu komut BBB'den gelen görüntü bilgisinin kabul edilmesini sağlar.
#

$ xhost +192.168.1.100

# PC'nin IP numarasının 192.168.1.33 olduğunu kabul edelim.
# BBB tarafında aşağıdaki komut girilir.
# Bu komut, görüntülerin 192.168.1.33 numaralı makineye
# gönderilmesi gerektiğini söyler.
#

$ export DISPLAY=192.168.1.33:0.0
$ xclock

# PC tarafında, işleyen bir saat çıkar.
#
# Özetle, saat programı BBB tarafında işler,
# ama görüntü DISPLAY ile atanan yerde, PC'de çıkar.
```

```
#
# Örnek saat görüntüsü aşağıda verilmiştir.
#
# BBB'nin herhangi bir monitöre bağlı olmadığını ve
# görüntünün tamamen ağ üzerinden aktarıldığını hatırlatalım.
#
```

Şekil 8.1: xclock

ssh ile BBB'ye erişmek için ağ tanımları aşağıdaki gibi yapılabilir. Bu tür tanımlar, PC tarafında rootfs.tar açılarak yapılmalıdır. Güncelleme bittikten sonra, rootfs.tar yeninden oluşturulur.

```
$ vi /etc/network/interfaces

auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
  address 192.168.1.100
  netmask 255.255.255.0

$ ifup eth0

$ vi /etc/ssh/sshd_config
PermitRootLogin yes

$ /etc/init.d/S50sshd restart

PC tarafından:

$ ssh root@192.168.1.100
root/root ile gir.
```

read/only kök dosya sistemi aşağıdaki gibi tekrar bağlanıp, read/write hale getirilebilir.

```
# BBB açıldıktan sonra, kök dosya sistemi r/o bağlıdır.
# Aşağıdaki gibi kök dosya sistemi üzerinde güncelleme yapılabilir.
#

$ mount -o remount -o rw /

# Güncelleme yap.
# İş bitince tekrar ro hale getir.

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

Bölüm 9

EKLER

Bu bölümde, gömülü sistemlerle doğrudan ilgili bazı Unix kavramlarının üzerinden geçilecektir.

9.1 Çapraz Derleyici Kullanmak

El yordamı ile kurulan derleyici PATH'e eklenir.

```
PATH=$PATH:/egitim/beagle.bone/linux-devkit/bin
```

PATH'ın doğruluğunu test etmek için:

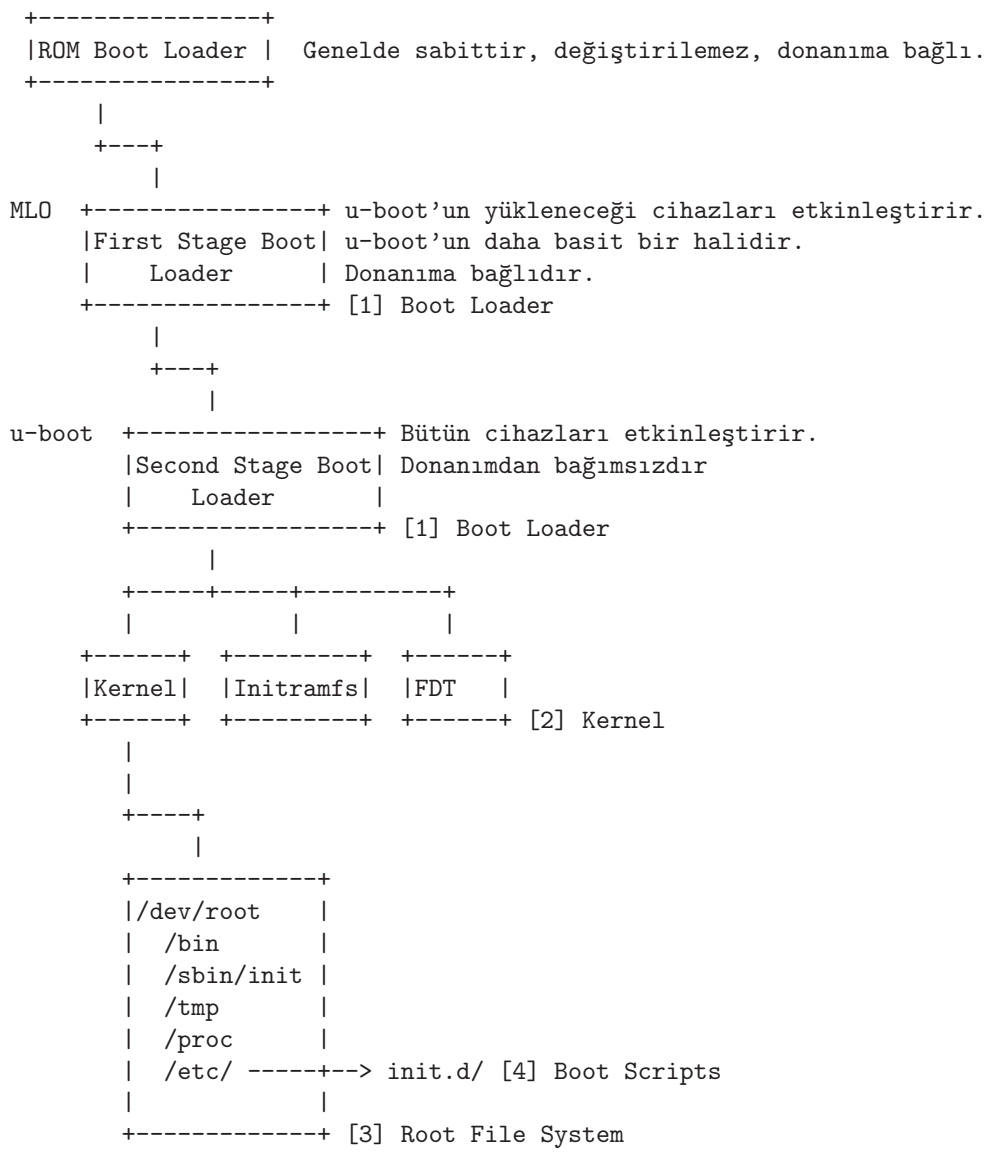
```
$ arm-linux- <TAB> <TAB>
$ which arm-linux-gcc
```

Aşağıdaki örneklerde, statik ve dinamik derleme, strip yapma, file ile dosya inceleme örnekler verilmiştir.

```
$ cat ilk.c

#include <stdio.h>
#include <stdlib.h>

int main(){
   fprintf(stdout, "Bu program sadece ARM makinede çalışır.\n");
   return EXIT_SUCCESS;
}

$ arm-linux-gcc ilk.c -o ilk

$ ls -l ilk
-rwxr-xr-x 1 nazim users 46241 Jun 18 17:13 ilk*

$ file ilk
ilk: ELF 32-bit LSB executable,
     ARM, version 1 (SYSV),
     dynamically linked (uses shared libs),
     not stripped

$ objcopy --strip-all ilk
objcopy: Unable to recognise the format of the input file ‘ilk’

$ arm-linux-objcopy --strip-all ilk

$ ls -l ilk
-rwxr-xr-x 1 nazim users 2784 Jun 18 17:13 ilk*
```

```
$ file ilk
ilk: ELF 32-bit LSB executable, ARM, version 1 (SYSV),
     dynamically linked (uses shared libs), stripped


$ arm-linux-gcc ilk.c -static -o ilk

$ file ilk
ilk: ELF 32-bit LSB executable,
     ARM, version 1 (SYSV),
     statically linked,
     not stripped

$ ls -l ilk
-rwxr-xr-x 1 nazim users 38944322 Jun 18 17:19 ilk*

$ arm-linux-objcopy --strip-all ilk
$ ls -l ilk
-rwxr-xr-x 1 nazim users 477016 Jun 18 17:19 ilk*

$ file ilk
ilk: ELF 32-bit LSB executable,
     ARM, version 1 (SYSV),
     statically linked,
     stripped

$ ./ilk
bash: ./ilk: cannot execute binary file
```

9.2 Gömülü Sistemin 4 Ayağı



9.3 Dosya Sistemleri

Gömülü sistemlerde genelde aşağıdaki dosya sistemleri kullanılır.

1. vfat
2. ext2
3. ext3
4. ext4
5. tmpfs
6. ramdisk
7. initramfs
8. cpio
9. NFS
10. cramfs
11. romfs
12. squashfs
13. zram
14. eCryptfs
15. ubifs
16. pseudo&virtual fs

Dosya sistemleri, dosya ve dizinlerin saklandığı yapılardır. Dosya sistemleri ve parametreleri amaca uygun seçilmelidir. Kötü seçim kaynak israfına ve fiziksel yapının bozulmasına sebeb olabilir.

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

Dosya sistemleri genelde 5 ortamda kurulur.

1. SD/MMC/eMMC Card: vfat, ext2, ext3, ext4, ...
2. NAND/NOR Flash: ubifs, jffs2, yaffs2, ...
3. RAM: ramdisk, tmpfs, ramfs, zramfs, ...
4. Network: NFS, samba, ...
5. Linux Kernel (Pseudo- and virtual): /dev, /proc, /sys, /dev/pts, /debugfs, ...

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

cat /proc/filesystems komutu ile çekirdeğin o an için desteklediği dosya sistemleri incelenebilir. Çekirdeğin bağladığı ilk dosya sistemine kök dosya sistemi denir.

9.4 VFAT

MS-Windows dosya sistemidir.

255 karaktere kadar dosya isimlerine izin verir.

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

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

Genelde SD/MMC kartların 1. bölümüne kurulur.

```
$ cd /tmp

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

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

$ mkdir /mnt/disk1

$ mount disk1.img /mnt/disk1

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

$ ls -l /mnt/disk1
total 0

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

$ mkdir /mnt/disk1

$ mount disk1.img /mnt/disk1

root@nkoc:/tmp# df /mnt/disk1
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/loop0         32686     0     32686   0% /mnt/disk1
```

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

Çekirdek desteği aşağıdaki gibi verilir.

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

9.5 Ext2

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

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

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

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

```
$ mkdir /mnt/disk2

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

$ mkfs.ext2  disk2.img

$ mount disk2.img /mnt/disk2

$ df /mnt/disk2
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/loop1         31729   395     29696   2% /mnt/disk2
```

Çekirdek desteği aşağıdaki gibi verilir.

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

9.6 Ext3

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

Recovery durumu çok hızlıdır.

Yerini ext4 sistemine bırakmıştır. Masaüstü sistemlerde artık ext4 kullanılmaktadır.

MMC cihazlarında, RW modu ile bağlanarak kullanılabilir.

Log tabanlı bir sistem olduğu için, diske yazılacak bütün veriler önce bir log tablosunda tutulur, daha sonra diske yazılır.

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

```
ext3 = journal + ext2
```

ext2 ve ext3 arasında her zaman geçiş yapılabilir.

Performans ve dosya bütünlüğü arasında ters ilişki vardır.

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

```
$ dd if=/dev/zero of=disk3.img bs=1M count=32

$ mkfs.ext3  disk3.img

$ mkdir /mnt/disk3

$ mount disk3.img /mnt/disk3

$ df /mnt/disk3
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/loop2         31729  4508     25583  15% /mnt/disk3
```

Çekirdek desteği aşağıdaki gibi verilir.

```
File systems -->
```

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

9.7 Ext4

ext3'ün devamıdır ama ext3 ile uyumlu değildir. ext4 dosya sistemi, ext3'ü mount edebilir ve ext3'e göre daha iyi performans gösterir. Doğrudan ext4 kullanılması tavsiye edilir.

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

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

Gömülü sistemlerde, MMC tarafında kök dosya sistemi olarak rw modunda mount edilebilir. ext2 sisteminin rw modunda bağlanması tavsiye edilmez.

9.8 Tmpfs

Sanal bellekte[1] kurulan bir dosya sistemidir.

```
virtual memory = ram + swap
```

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

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

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

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

```
$ mkdir /mnt/disk4

$ mount -t tmpfs tmpfs /mnt/disk4

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

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

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

```
$ df /mnt/disk?

Filesystem     1K-blocks  Used Available Use% Mounted on
```

[1] virtual memory

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

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

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

Çekirdek desteği aşağıdaki gibi verilir.

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

9.9 Ramdisk

RAM'ın (virtual memory değil!) disk bölümü gibi kullanılmasıdır. Dosya sistemi değildir, dosya sistemleri için altyapı hazırlar. Sanki bir disk bölümü (disc partition) gibi üzerine hemen dosya sistemi kurulabilir. tmpfs gibi umount edildiğinde yok olmaz.

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

Ramdisk adedi, kapasitesi ve blok boyu çekirdek parametresi olarak verilebilir. Varsayılan blok boyu 1024'tür.

Mümkünse ramdisk kullanılmamalı, bunun yerine tmpfs kullanılmalıdır. Ramdisk artık çok eski bir teknik olarak kalmıştır. ramdisk, kullanıldığı an RAM bellekten kapasite çalar. umount edilse bile kapasite geri verilmez.

Sanal dosya sistemi kullanmadığı için ihtiyaç durumunda swap edilemez. Prensip olarak gömülü sistemlerde swap kullanılmaz.

Bazı çekirdekler hala initial ram disk[2] kullanmaktadırlar. Fakat initrd kavramı yerini initramfs'e (tmpfs'e) bırakmıştır. tmpfs bir dosya sistemidir. Çekirdek tarafından bizim hiç ilgilenmediğimiz bir yerlerde kurulur!

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

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

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

$ dmesg | grep RAMDISK
```

4M'lık ext2 kur ve paketle?

```
$ dd if=/dev/zero of=/dev/ram0 bs=1k count=4096
$ mkfs.ext2 /dev/ram0 4096
```

[2] initrd

```
$ mkdir /mnt/disk
$ mount /dev/ram0 /mnt/disk
$ df
$ mount
$ umount /mnt/disk

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

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

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

Çekirdek desteği aşağıdaki gibi verilir.

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

9.10 Initramfs-I

initramfs[3], çekirdeğe gömülü olabilir.

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

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

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

Ayrık olan dosya boot loader tarafından çekirdekle birlikte yüklenir ve başlangıç adresi ve dosya boyu çekirdek parametresi olarak çekirdeğe gönderilir. Ayrık dosya her zaman cpio arşivi şeklindedir, dosya sistemi değildir.

Çekirdek önce kendini açar, sonra da cpio arşivini, bir tmpfs dosya sistemi içine açar. Bu dosya sistemine initramfs denir ve çekirdek bu dosya sistemini kök dosya sistemi olarak bağlar Hemen peşinden /init komutunu gözü kapalı işletir.

initrd tekniği ile açılış yapıldığında /linuxrc işletilir. initrd[4] sistemi artık eski kalmıştır ve kullanılmamaktadır.

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

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

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

[3] Initial RAM File System
[4] initial RAM disk

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

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

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

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

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

9.11 Initramfs-II

initramfs, çekirdekten ayrı kurulabilir. Çekirdek bilinen yolla derlenir.

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

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

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

uImage ve uramfs beraber kullanılır. uramfs içindeki programların GPL sınıfından olması gerekmez. Güncellemelerde sadece uramfs kullanılır, çekirdek derlenmez.

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

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

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

9.12 Cpio

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

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

output modunda dosya isimleri bir dosyadan satır satır okunur.

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

Çekirdek sadece newc arşiv formatını kullanır.

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

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

input veya extract modunda paket açılır.

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

-i: input veya extract demektir.

-o: output veya create demektir.

9.13 Nfs

NFS[5], ağ üzerinden dosyalara erişim tekniğidir. Farklı bir tanımla, ağ üzerinden dizin paylaşım yöntemidir.

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

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

$ exportfs -avr ile yapılan güncellemeler sunucuya bildirilir.

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

Ağ için çekirdek parametreleri en genel hali ile aşağıda verilmiştir.

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

autoconf parametresi çok önemlidir. Çekirdek belgelerinde aşağıdaki gibi tanımlanmıştır.

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

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

Default: any
```

[5]Network File System

Örnek çekirdek parametreleri aşağıda verilmiştir.

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

Sıradan bir dizini mount etmek için,

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

Kök dosya sistemi NFS üzerinden kullanılacaksa çekirdek aşağıdaki gibi derlenir.

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

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

9.14 Cramfs

cramfs[6], sıkıştırılmış dosya sistemidir. mount edildikten sonra da sıkıştırılmış olarak kalır, açılarak kullanılmaz. Sadece ro erişim yapılabilir. Dosya boyu 16MB'yı geçemez. FS büyüklüğü 256MB'tan büyük olamaz. Çok az yer kaplar, gömülü sistemler için uygundur.

Mevcut kısıtlamalara dikkat edilerek kullanılmalıdır. uid/gid desteği, hard link'ler ve zaman bilgileri desteklenmez.

```
$ mkfs.cramfs -v -n test.image /tftpboot/RootFS cramfs.img

$ ls -l cramfs.img
-rw-rw-r-- 1 nazim nazim 1089536 May 11 10:17 cramfs.img

$ du -ks /tftpboot/RootFS/
2120 /tftpboot/RootFS/

$ file cramfs.img
cramfs.img: Linux Compressed ROM File System data,
            little endian size 1089536 version $2
            sorted_dirs
            CRC 0x4101b0de, edition 0, 526 blocks,
            388 files

$ mount -o loop cramfs.img /mnt/disk
$ ls -l /mnt/diks   # timestamp= unix epoch time

$ df /mnt/disk
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/loop0          2104  2104         0 100% /mnt/disk
```

Çekirdek desteği aşağıdaki gibi verilir.

```
File systems  --->
    [*] Miscellaneous filesystems  --->
            <*>   Compressed ROM file system support (cramfs)
```

[6]Compressed ROM File System

9.15 Romfs

romfs[7], çok az yer kaplayan, sadece ro erişim yapılabilen bir dosya sistemidir. Daha çok doğrudan donanım içine gömülen dosya sistemleri için uygundur.

```
$ genromfs -d /tftpboot/RootFS -f romfs.img -V 'romfs image'

$ ls -l romfs.img
-rw-rw-r-- 1 nazim nazim 1977344 May 11 10:40 romfs.img

$ du -ks /tftpboot/RootFS/
2120 /tftpboot/RootFS/

$ mount -o loop -o ro -t romfs romfs.img /mnt/disk

$ df /mnt/disk
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/loop0          1931  1931         0 100% /mnt/disk

$ ls -l /mnt/disk

$ cat /proc/filesystems | grep romfs
```

Çekirdek desteği aşağıdaki gibi verilir.

```
File systems  --->
    [*] Miscellaneous filesystems  --->
            <*>   ROM file system support
```

[7]ROM File System

9.16 Squashfs

gzip/lzo/xz ile sıkıştırılmış, salt okunur dosya sistemidir. cramfs'e göre daha iyi sıkıştırma yapar. tar.gz'nin mount edilmesi gibi düşünülebilir.

Pek çok LiveCD'nin "defacto FS" standardı gibidir. uid/gid, timestamp desteği vardır. 4GB'tan büyük dosyaları destekler.

```
$ mksquashfs /tftpboot/RootFS.busybox rootfs.sq

$ ls -l rootfs.sq
-rw-r--r-- 1 root root 974848 May 11 12:30 rootfs.sq

$ du -ks /tftpboot/RootFS.busybox
2120    /tftpboot/RootFS.busybox

$ file rootfs.sq
$ mount rootfs.sq /mnt/disk
$ ls -l /mnt/disk

$ df /mnt/disk
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/loop0          1024  1024         0 100% /mnt/disk

$ losetup -a
/dev/loop0: [0801]:655457 (/mnt/rootfs.sq)

$ unsquashfs rootfs.sq
$ cd squashfs-root/
```

Çekirdek desteği aşağıdaki gibi verilir.

```
File systems  --->
 [*] Miscellaneous filesystems  --->
      <*>   SquashFS 4.0 - Squashed file system support
```

9.17 Zram

Compressed RAM based block device. Masaüstü sistemlerde swap'ın hızı ve kapasitesini artırmak için kullanılır.

/dev/ram0'ın sıkıştırılmış hali gözü ile bakılabilir.

Her zaman modül olarak derlenmesi tavsiye edilmektedir. Böylece disk sayısı amaca göre atanabilir.

/var, /tmp gibi dizinler için de kullanılabilir. Fakat tmpfs gibi olmayıp doğrudan ram'dan kapasite çalınır.

```
$ modinfo zram
$ modprobe zram num_devices=4
$ ls -l /dev/zram?
$ lsmod|grep zram
$ echo $((4096*1024)) > /sys/block/zram0/disksize

$ mkfs.ext2 /dev/zram0
$ mount /dev/zram0 /mnt/disk

$ df /mnt/disk
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/zram0          3952    24      3724   1% /mnt/disk

$ cd /mnt/disk ; ls -la

$ mount | grep zram
/dev/zram0 on /mnt/disk type ext2 (rw)

$ dmesg|grep zram
$ umount /mnt/disk
$ rmmod zram
$ lsmod | grep zram
```

Çekirdek desteği aşağıdaki gibi verilir.

```
Device Drivers  --->
    [*] Staging drivers  --->
            <M>   Compressed RAM block device support
            [ ]      Compressed RAM block device debug support
            [*]   Memory allocator for compressed pages
```

9.18 eCryptfs

Şifreli bir dosya sistemidir. VFS layer üzerinde işler, olgunluğa erişmemiştir.

Stack based desteklidir. Bundan dolayı mevcut herhangi bir dosya sistemi üzerinde şifreleme yapabilir.

Önce boş bir dizin mount edilir. Sonra bu dizine dosyalar kopyalanır. Diske yazılan dosyalar artık şifrelenmiştir.

İçi dolu dizinler mount edilmemelidir. user space tarafında, ecryptfs-utils programlarına gerek vardır.

```
$ mount -t ecryptfs /srv /srv

$ more ~/.ecryptfs/sig-cache.txt
1aba6bfb1a2bf34e

$ echo "Merhaba" > /srv/test
$ cd /srv/test
$ ls -l

$ cat test
Merhaba

$ df
Filesystem     1K-blocks      Used Available Use% Mounted on
/srv            65924860   9332148  53220888  15% /srv

$ mount | grep ecryptfs
/srv on /srv type ecryptfs (rw,ecryptfs_sig=1aba6bfb1a2bf34e,
ecryptfs_cipher=aes,ecryptfs_key_bytes=16,
ecryptfs_unlink_sigs)

$ umount /srv
$ file /srv/test
$ cat > ~/.ecryptfsrc
key=passphrase:passphrase_passwd_file=/tmp/passwd_file.txt
ecryptfs_sig=1aba6bfb1a2bf34e
ecryptfs_cipher=aes
ecryptfs_key_bytes=16
ecryptfs_passthrough=n
ecryptfs_enable_filename_crypto=n

$ cat .ecryptfs/sig-cache.txt
1aba6bfb1a2bf34e
```

```
$ passwd_file.txt, usb, tpm vs gibi bir ortamda olabilir.

$ cat /tmp/passwd_file.txt
passphrase_passwd=deneme

root@nkoc:~$ mount -t ecryptfs /srv /srv
```

Çekirdek desteği aşağıdaki gibi verilir.

```
[*] Cryptographic API  --->
   [*]   MD5 digest algorithm
   <*>   AES cipher algorithms (ARM-asm)

Security options  --->
   [*] Enable access key retention support

File systems  --->
   [*] Miscellaneous filesystems  --->
        <*> eCrypt filesystem layer support (EXPERIMENTAL)
```

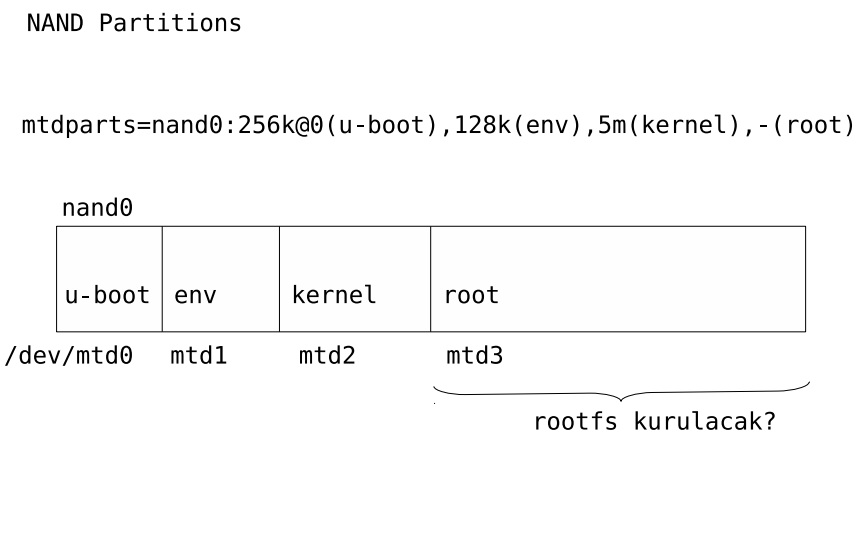
9.19 Ubifs

Flash diskler için geliştirilmiş, kapanmaya dayanıklı ve ubi üzerine kurulan, log tabanlı bir dosya sistemidir. NAND diskler için neredeyse standard gibi olmuştur.

Diğer dosya sistemleri gibi doğrudan kurulup mount edilemez, ön hazırlık gerekir. Tamamen boş bir disk mount edilirse otomatik olarak ubifs kurulur, açıkça da kurulabilir.

fsck programı yoktur. Mount sırasında hata meydana gelirse dosya sistemini düzeltmeye çalışır. Düzeltirse mount eder, düzeltmezse ya ro mount eder ya da hiç mount etmez.

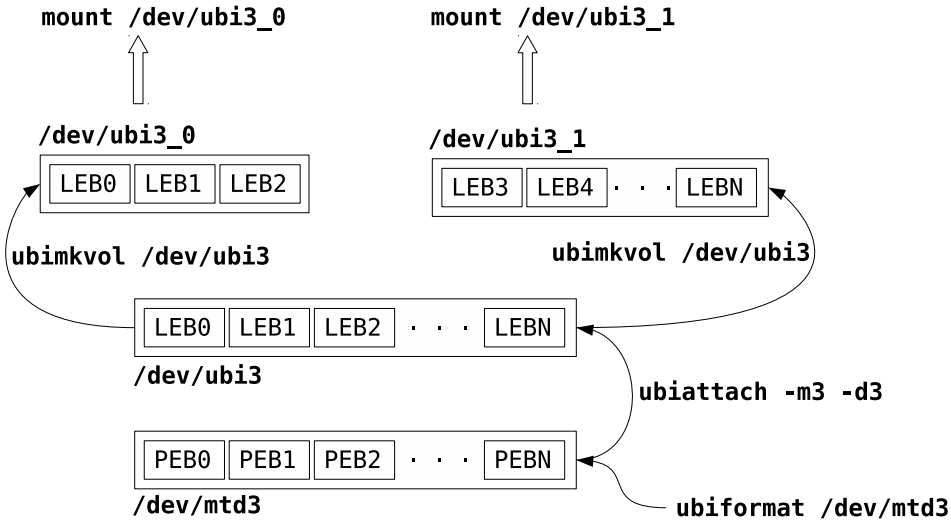
loop mount desteği yoktur, simulator ile mount edilir.



Şekil 9.1: NAND Bölümlendirmesi.

Bord tarafında çalışırken ubifs kuruluşları (1) Host tarafında, /tftpboot/RootFS altında örnek bir RootFS olduğunu kabul edelim.

```
$ ubiformat /dev/mtd3
$ ubiattach -p /dev/mtd3
$ ubimkvol /dev/ubi0 -N root -m
$ mount -t ubifs ubi0:root /mnt/root

$ mount -t nfs host:/tftpboot /mnt/nfs
$ cp -a /mnt/nfs/* /mnt/root

$ nanddump /dev/mtd3 -f /mnt/nfs/mtd3.img

$ umount /mnt/root
```



Şekil 9.2: Formatlama.

Bord tarafında çalışırken ubifs kuruluşları (2) Host tarafında, ubinize ile root.ubi isimli imajın kurulduğunu kabul edelim.

```
$ ubiformat /dev/mtd3 -f /tftpboot/root.ubi
$ ubiattach -p /dev/mtd3
$ mount -t ubifs ubi0:root /mnt/root
```

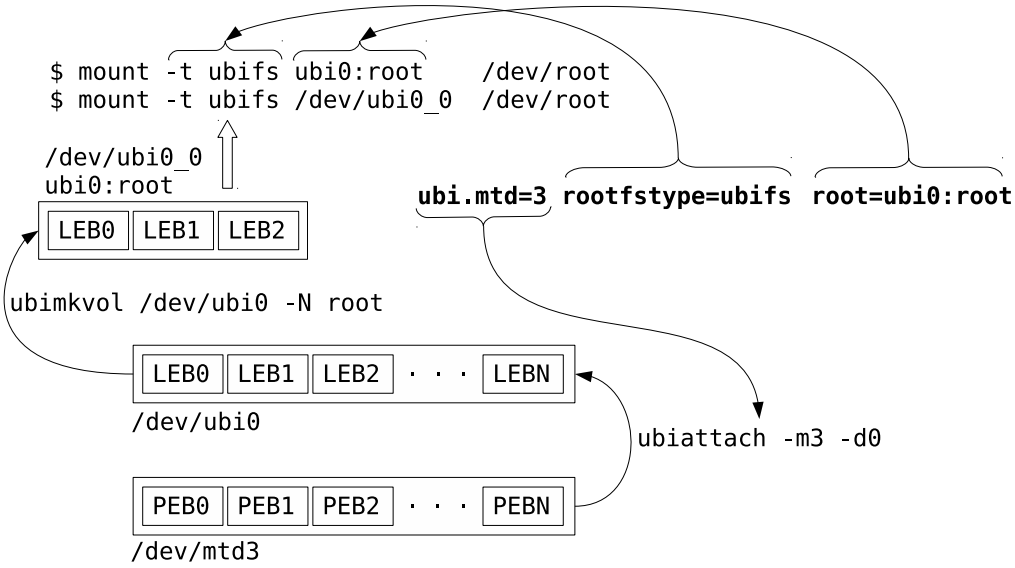
Bord tarafında çalışırken ubifs kuruluşları (3) Host tarafında, mkfs.ubifs kurulan imajın yazılması.

```
$ ubiformat /dev/mtd3
$ ubiattach -p /dev/mtd3
```

```
$ ubimkvol /dev/ubi0 -N root -m
$ updatevol /dev/ubi0_0 /tftpboot/root.ubifs
$ mount -t ubifs ubi0:root /mnt/root

$ mount -t ubifs /dev/ubi0_0 /mnt/root
```

Bord tarafında çalışırken ubifs kuruluşları (4) Host tarafında, ubinize ile root.ubi kurulmuş olsun. U-Boot seviyesinde iken:

```
=> tftp 32000000 root.ubi
=> nand erase root
=> nand write root.ubi

# Açıldıktan sonra:
$ mount -t ubifs ubi0:root /mnt/root    # veya
$ mount -t ubifs /dev/ubi0_0 /mnt/root
```



Şekil 9.3: Bağlama.

Host tarafında çalışırken ubifs kuruluşu: Bord'a ait bir imajı host tarafında oluşturma işine ubinize denir. Bord tarafında ubinize yapmanın bir anlamı yoktur. Burada esas amaç NAND için master bir imaj oluşturmaktır. Genelde müşteriye verilecek en son sistem için hazırlanır. Test veya geliştirme için ubinize kullanmak pek pratik değildir.

ubinize işlemine önce ubifs kurularak başlanır. Burada NAND imajı ile ilgili bilgi bulunmaz.

Bir imaj içinde çok fazla volume veya ubifs bulunabilir. Her bir ubifs'in parametreleri root.ini gibi bir dosyaya yazılır.

Çok fazla volume olacağı için parameterlerin argüman olarak verilmesi pratik değildir.

Fakat genelde tek NAND cihazı vardır. Bundan dolayı NAND parametreleri argüman olarak verilir.

NAND üzerinde çalışırken mutlaka 4 parametrenin el altında olması gerekir.

```
PEB          : 16KiB   -p
LEB          : 15872   -e
Min I/O Size : 512     -m
Sub page size: 256     -s
```

Bu parametrelerinin bazıları ubifs bazıları ubi için gereklidir. Örneğin, ubifs PEB ile ilgilenmez, benzer şekilde ubi sistemi de LEB ile ilgilenmez.

ubifs, ubi üzerine kuruludur. ubi sistemi bir dosya sistemi değildir, dosya sistemi ile bare NAND arasında bir katmandır.

ubi'nin esas amacı bad blokları yönetmektir. Böylece ubi üzerine kurulu dosya sistemleri bu sıkıntılı işlerle uğraşmazlar.

ubi üzerine illa ki ubifs kurmak gerekli değildir. squashfs, cramfs gibi dosya sistemileri ubi üzerine kurulabilir.

Örnek bir ubifs kuruluşu örnekte verilmiştir. ubifs ile NAND sisteminin hiç bir ilgisi yoktur. Bu ilgi ubinize komutu ile kurulur.

```
$ mkfs.ubifs -v -m 512 -e 15872 -r RootFS -c 4228 root.ubifs

ubifs parameterleri:
LEB          : 15872   -e
Min I/O Size : 512     -m


$ cat root.ini
```

```
[root]                      # Keyfi bir isim.
mode=ubi                    # Zorunlu bir satır.
image=root.ubifs            # mkfs.ubifs ile elde edildi.
vol_id=0                    # Kaçıncı ubi volume?
vol_size=32MiB              # Kaba bir boy miktarı.
vol_type=dynamic            # LEB'ler rw olabilir. ro=static
vol_name=root               # Volume name.
vol_flags=autoresize        # Ilk mount'da genişle.

$ ubinize -v -o root.ubi -p 16KiB -m 512 -s 256 root.ini

# ubi paramterleri:
# PEB          : 16KiB   -p
# Min I/O Size : 512     -m
# Sub page size: 256     -s
#
# root.ini dosyası nandwrite, nanddump, ubiformat ile mtd'ye
# yazılabilir.
```

Bir ubi sisteminde en fazla 128 volume, diğer bir deyişle 128 adet ubifs olabilir.

Gömülü sistemlerde genelde her ubi için bir adet ubifs bulunur. Birden fazla olması pek nadirdir.

Bir mtd cihazı ubiformat ile formatlanmamışsa, ubiattach her zaman başarısız olur. ubiattach, her zaman bare NAND cihazı ile ubi cihazını birbirine bağlar. Kaba bir deyişle, PEB ve LEB arasındaki ilişkiyi sağlar.

LEB boyları, her zaman PEB'den biraz daha kısadır. mtd ile ubi arasındaki bağlantı ubidetach -m No komutu ile koparılır.

umount edilmeden, yani ubi ile ubifs arasındaki ilişki kesilmeden, ubidetach işlemi yapılamaz. Her zaman önce ubidetach sonra umount yapılır.

NAND'a ne yazılırsa yazılsın, ilgili bölge MUTLAKA önce flash_erase veya nanderase gibi komutlarla silinmelidir. Silme işlemi bütün bölüme 1 yazar.

Yazma işlemi asla 0'dan 1'e yazım yapamaz. Bundan dolayı önce erase işlemi ile bütün bölüm 1 yapılır. write işlemi sadece 1'den 0'a değişimleri yapabilir.

NAND'a yazılacak her dosya mutlaka minimum I/O değerinin bir katı olmalıdır.

Bir dosya boyunu 2k'nın katı yapmak için,

```
$ dd if=foo of=foo2k bs=2k conv=sync
```

Bir volume ilk defa mount ediliyorsa ve içi tamamen boşsa, ubifs sistemi bunu anlar ve otomatik olarak ubifs dosya sistemini yaratır.

```
  $ mount -t ubifs ubi3:root /mnt/root
    UBIFS: default file-system created  <-- Sadece ilk mount
    ...                                     işleminde uygulanır.
```

mtd_debug programı ile bare NAND hakkında bilgi alınabilir.

```
$ mtd_debug info /dev/mtd3
```

Diğer faydalı komutlar ve dosyalar:

```
$ mtdinfo --all
$ ubiinfo /dev/ubi0
$ ubiinfo /dev/ubi0_0
$ cat /proc/mtd
$ cat /proc/partitions
$ cat /proc/cmdline
$ ls /proc/fs/ubifs
$ ls /proc/drivers/mtd
$ ls /proc/drivers/mtd/ubi
```

PEB'lerin %1'i bad block yönetimi için kullanılır.

Bir PEB hatalı olursa hemen yeni bir PEB atanır. Hatalı olan bad diye işaretlenir. LEB, yeni PEB'e göre güncellenir. Bad block yönetimini ubi yapar, ubifs değil.

Eğer vol_type= static ise mount işlemi her zaman ro yapılır. vol_type= dynamic ise mount işlemi ro veya rw yapılabilir.

Eğer rw mount işleminde düzeltilemez bir hata varsa, mount işlemi otomatik olarak ro yapılır.

/dev/mtd0 veya bare NAND flash cihazılarına yazım için dd kullanılamaz. Çünkü dd, bad block'ları kontrol etmez. Fakat okuma yapılabilir.

Yorum /dev/ubi0_1 için: /dev/ubi0'ı /dev/sda gibi düşün. /dev/ubi0_1'i /dev/sda1 gibi düşün.

mtd üzerinde ubi varsa asla mtd üzerinden okuma/yazma yapılmamalıdır. Çünkü ubi'nin kendi iç veri yapısı vardır. Yazma işlemi bu yapıyı bozar. Okuma işlemi ile bu yapı okunabilir ama anlamsızdır.

/dev/mtd0 her zaman bare flash'a denk gelir. Bare flash, host tarafında ubinize tarafından hazırlanır.

Tamamen boş dosya sisteminde 0xFF vardır.

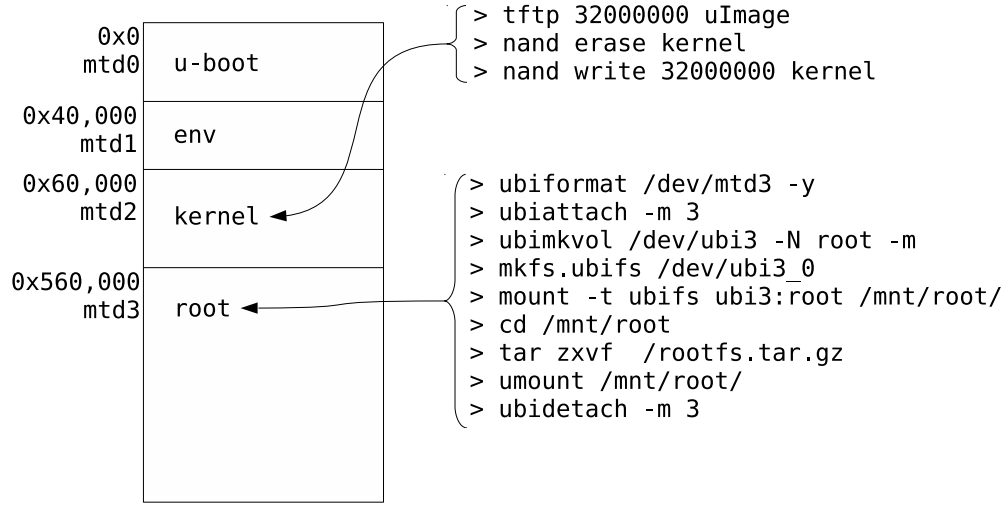
```
$ dd if=/dev/ubi0_0 of=foo
```

ubi imajlarının loop mount özelliği yoktur. Fakat aynı etki nand simülatörü kullanılarak elde edilebilir.

Host sisteminde ubi imajının incelenmesi: 128MB'lik disk kur /tmp/mtd7.img imajını mount et.

```
$ modprobe nandsim first_id_byte=0xec second_id_byte=0xa1
                   third_id_byte=0x00 fourth_id_byte=0x15

$ dd if=/tmp/mtd7.img of=/dev/mtd0 bs=2048

$ modprobe ubi mtd=0

$ mount -t ubifs /dev/ubi0_0 /mnt/ubifs

$ df /mnt/ubifs
$ ls -l /mnt/ubifs

$ umount /mnt/ubifs
$ rmmod ubifs ubi nandsim
```

Çok sıradan bir NAND kuruluşu Şekil 9.4'de verilmiştir. Bölümlerin mtdparts ile görünütüsü de devamında listelenmiştir.



Şekil 9.4: NAND diskte bölüm kuruluşları.

```
=> mtdparts

device nand0 <mini2440-nand>, # parts = 4
 #: name                        size     offset          mask_flags
  0: u-boot              0x00040000      0x00000000      0
  1: env                 0x00020000      0x00040000      0
  2: kernel              0x00500000      0x00060000      0
  3: root                0x3faa0000      0x00560000      0
```

Çekirdek desteği aşağıdaki gibi verilir.

```
Device Drivers  --->
   <*> Memory Technology Device (MTD) support  --->
        [*]   Command line partition table parsing
        <*>   NAND Device Support  --->
              <*>   NAND Flash device on OMAP2, OMAP3 and OMAP4
```

9.20 Nandsim

nandsim[8], NAND cihazlarını simüle eden bir cihaz sürücüsüdür. Elde NAND cihaz yokken, varmış gibi bütün NAND işlemlerini taklit eder.

```
$ modinfo nandsim

# NANDsim parametreleri:

modprobe nandsim first_id_byte=0x20 second_id_byte=0x33  (16MiB, 512 bytes page)
modprobe nandsim first_id_byte=0x20 second_id_byte=0x35  (32MiB, 512 bytes page)
modprobe nandsim first_id_byte=0x20 second_id_byte=0x36  (64MiB, 512 bytes page)
modprobe nandsim first_id_byte=0x20 second_id_byte=0x78  (128MiB, 512 bytes page)
modprobe nandsim first_id_byte=0x20 second_id_byte=0x71  (256MiB, 512 bytes page)

modprobe nandsim first_id_byte=0x20 second_id_byte=0xa2 third_id_byte=0x00
                                    fourth_id_byte=0x15  (64MiB, 2048 bytes page)

modprobe nandsim first_id_byte=0xec second_id_byte=0xa1 third_id_byte=0x00
                                    fourth_id_byte=0x15  (128MiB, 2048 bytes page)

modprobe nandsim first_id_byte=0x20 second_id_byte=0xaa third_id_byte=0x00
                                    fourth_id_byte=0x15  (256MiB, 2048 bytes page)

modprobe nandsim first_id_byte=0x20 second_id_byte=0xac third_id_byte=0x00
                                    fourth_id_byte=0x15  (512MiB, 2048 bytes page)

modprobe nandsim first_id_byte=0xec second_id_byte=0xd3 third_id_byte=0x51
                                       fourth_id_byte=0x95  (1GiB, 2048 bytes page)
```

[8]NAND Simulator

9.21 Pseudo File Systems

Sözde dosya sistemleri[9] kernel ve user space arasında iki yönlü bilgi alışverişi için tasarlanmıştır.

Dosya ve dizinler her zaman ya boot anında ya da tam kullanım anında kurulurlar ve sistem kapanınca yok olurlar. Fiziksel olarak bir diskte kurulmazlar bundan dolayı dosya boyları her zaman 0'dır ama içleri dolu olabilir.

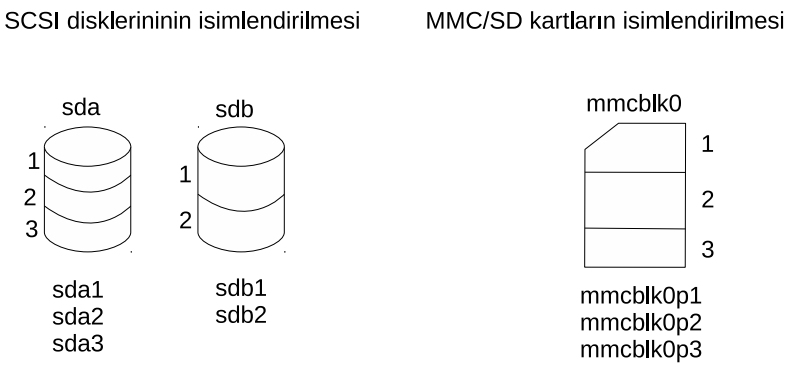
Gömülü sistemlerde genelde açılış betikleri tarafından mount edilirler.

Çok kullanılan bazı dosya sistemlerinin mount edilmesi...

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

[9]Pseudo File Systems. Bazen "sözde", bazen "sanki" kelimelerini kullanıyorum. Sözde kelimesi daha uygun gibi.

9.22 Disklerin ve Bölümlerin İsimlendirilmesi



Şekil 9.5: Disk ve Bölümlerin Adlandırılması.

9.23 MS-DOS bölümlendirmesi

Disklerin tek bir blok olarak kullanılması pratik değildir.

Diskler, daha ufak disk parçalarına ayrılarak kullanılırlar. Bu ayırma işlemine bölümlendirme denir. Linux pek çok disk bölümlendirmesini destekler.

MMC diskler için MSDOS bölümlendirmesi, NAND diskler için mtdparts (NAND Disk Partition Types) bölümlendirilmesi kullanılır. Her iki bölümlendirme desteği çekirdeğe verilmelidir.

MMC bölümlendirilmesi fdisk/sfdisk/cfdisk vs ile yapılabilir.

```
$ fdisk /dev/mmcblk0
```

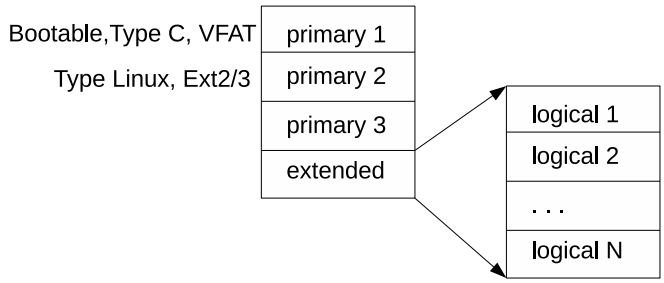
veya loop cihazı olarak...

```
$ dd if=/dev/zero of=/tmp/mmc.img bs=1M count=32
$ fdisk /tmp/mmc.img
```

Bütün işler bellekte yapılır, w ile çıkılmadığı sürece değişiklikler kaydedilmez.

x ile uzman moduna geçilir. Uzman modunda, başlangıç sektörü ayarlanabilir. U-Boot için birinci sektör c tipinde ve bootable olmalıdır.

Gömülü sistemlerde, MMC üzerinde genelde 2 bölüm kullanılır. İlk bölüm u-boot'un desteklediği bir dosya sistemi olur, vfat/ext2/ext3/ext4 gibi.



Şekil 9.6: MS-DOS Bölümlendirmesi.

Çekirdek desteği aşağıdaki gibi verilir.

```
[*] Enable the block layer  --->
    Partition Types  --->
      [*] Advanced partition selection
      [*]   PC BIOS (MSDOS partition tables) support
```

9.24 MTD Bölümlendirmesi

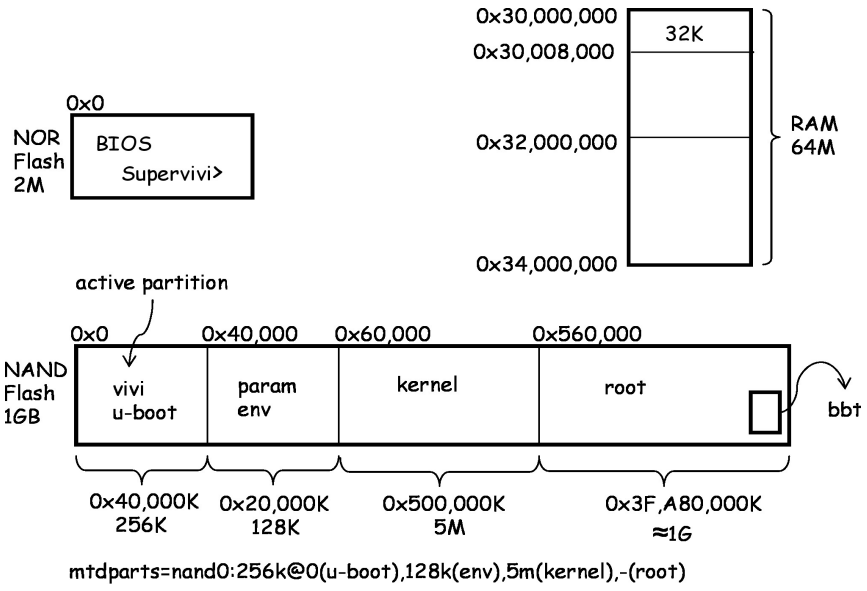
MS-DOS bölümlendirmesinde, bölümlendirme tablosu, diskin 1. sektöründe saklanır. Bu sektöre MBR denir. Çekirdek, MBR'yi okuyarak bölümlendirme hakkında bilgi sahibi olur.

Fakat NAND disklerde MBR kavramı yoktur. Çekirdek, mtdparts parametresi sayesinde NAND bölümlendirmesinden haberdar olur. mtdparts değişkeni bir kaç yoldan çekirdeğe verilebilir.

En yaygın olanı u-boot çevre değişkeni olarak, bootargs içinde tanıtmaktır. (test sistemlerinde tavsiye edilir.)

Diğer yol, çekirdek derlemesi sırasında, boot parametresi olarak vermektir. (nihai sistemler için tavsiye edilir.)

Son yol ise, doğrudan u-boot kodu içine kazımaktır. (nihai sistemler için tavsiye edilir.)



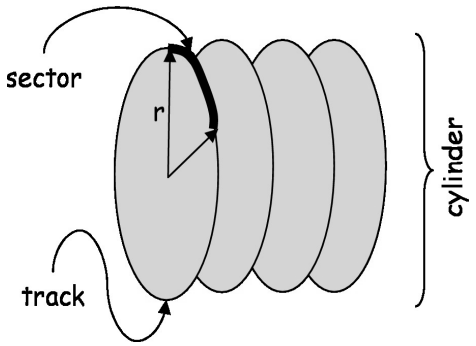
Şekil 9.7: MTD Bölümlendirmesi.

Çekirdek desteği aşağıdaki gibi verilir.

```
Device Drivers  --->
   <*> Memory Technology Device (MTD) support  --->
       [*]   Command line partition table parsing


HELP:
The format for the command line is as follows:

mtdparts=<mtddef>[;<mtddef]
<mtddef>  := <mtd-id>:<partdef>[,<partdef>]
<partdef> := <size>[@offset][<name>][ro]
<mtd-id>  := unique id used in mapping driver/device
<size>    := standard linux memsize OR "-" to  denote all
                                remaining space
<name>    := (NAME)
```

9.25 Fdisk kullanımı

```
$ fdisk [-b sectorsize] [-C cyls] [-H heads] [-S sects] device
```



Şekil 9.8: fdisk'in kullanımı.

```
$ MMC_DEVICE=/dev/mmcblk0

$ dd if=/dev/zero of=$MMC_DEVICE bs=512 count=1

$ sudo fdisk -H255 -S63 $MMC_DEVICE << EOF
o
n
p
1

+$BOOT_SIZE
a
1
t
c
x
b
1
63
r
n
p
2

+$ROOT_SIZE
p
w
EOF
```

9.26 /dev/null

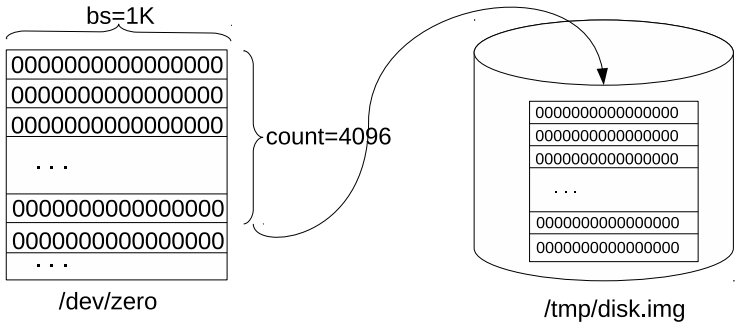
Linux sistemini,sosuz kapasiteli çöp tenekesidir.

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

9.27 /dev/zero

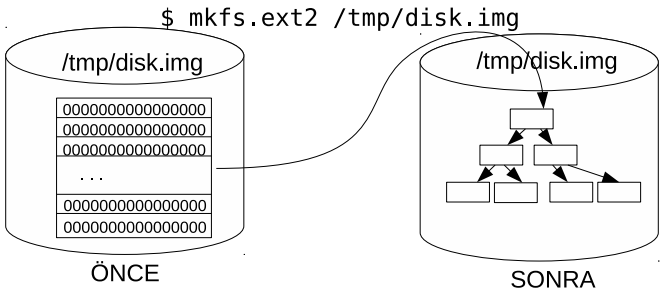
İçinde sonsuz adet "sıfır" bulunan bir cihazdır.

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



Şekil 9.9: /dev/zero cihazı.

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



Şekil 9.10: /dev/zero uygulaması.

9.28 /dev/random, /dev/urandom

/dev/random cihazı sistemdeki entropi değişimine göre keyfi sayı üretir. Sistemde entropi değişikliği yoksa sayı üretilmez. Bundan dolayı üzerinde iş yapılmayan makinelerde sayıların üretilme hızı çooook yavaştır. Fakat nitelikli keyfi sayılar üretilir.

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

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

9.29 /dev/full

Uygulama programlarında “tam dolu dosya” testi için kullanılabilir.

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

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

9.30 rsync

Genelde farklı 2 makine içindeki dizinleri eş tutabilmek için kullanılır. Özellikle gömülü sistemlere program aktarmak veya güncelleme yapmak için çok elverişlidir.

Çok düşük band genişliği kullanılır. Sadece güncellemeleri, farklılıkları taşır. Binary dosyaların dahi sadece farklılıklarını taşır.

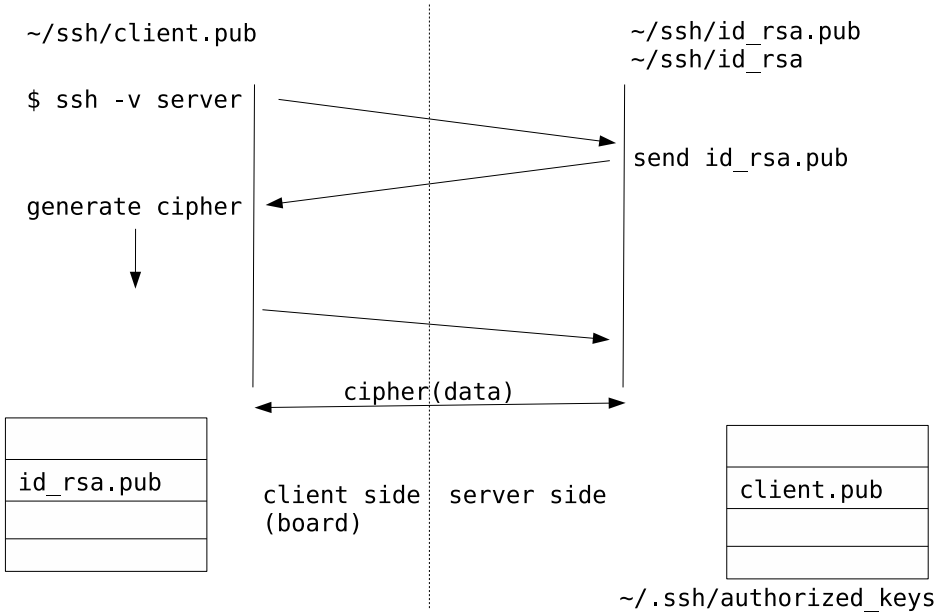
Taşıma sırasında sıkıştırma yapar. Bütün dosya özelliklerini korur. En önemlisi ssh üzerinden işler, güvenlidir.

Uzak makinelere kopyalama sırasında, uzak makindeki kullanıcının şifresi sorulacaktır. Uzaktaki makineye public anahtar kaydedilirse şifre sorulmaz.

```
# -a: archive mode.
# -v: verbose on

$ rsync arch/arm/boot/uImage root@10.1.2.3:/boot
$ rsync -a RootFS/etc  10.1.2.3:/
```

9.31 Ssh

ssh, "secure shell" demektir. Güvenli olmayan ağlarda, uzaktaki makineye güvenli biçimde bağlanmayı sağalar. Asimetrik anahtar yardımı ile simetrik anahtar değişimi yapar.



Şekil 9.11: ssh

```
$ ssh-keygen -t RSA test

$ chmod 700 ~/.ssh
$ vi ~/.ssh/config
Host server1
     HostName beagle_test
     User root
     Port 4242
     IdentityFile test
```

9.32 Makefile

Hedef dosyanın, bağımlıklıklara göre nasıl elde edileceğini tarif eden dosyaya Makefile denir. İki temel kavramı vardır: girişler ve bağımlılıklar.

```
# Makefile, girişler

all: derle goster sil

derle:
        echo "programlar derlendi."

goster:
       echo "programları göster."

sil:
       echo "hepsi silindi."


# Makefile, bağımlılıklar.
# Değişkenler: $(NAME)
# Default target: all

CFLAGS = -g -Wall -Os
PROG = test

all: $(PROG)

# $@: test, target name
# $^: file1.o file2.o, bağımlılıklar
#
# gcc -o test file1.o file2.o -g -Wall -Os
test: file1.o file2.o
     gcc -o $@ $^ $(CFLAGS)

clean:
     $(RM) -f $(PROG)

install:
    cp $(PROG) /usr/local/bin


$ cat file1.c

#include <stdio.h>
#include "project.h"
int main(){
```

```
   fprintf(stdout, "program başarılı bitti.\n");
   return 0;
}

$ cat file2.c

#include <stdio.h>
void test(){
   fprintf(stdout, "test tamamlandı.\n");
}

$ cat project.h
void test();



# compile
$ gcc -g -Os -Wall file1.c file2.c -o project

# install
$ sudo cp project /usr/local/bin

# clean
$ rm project *.o

# run
$ cd
$ project
program başarılı bitti.
```

Büyük projelerde, bütün bu adımları el ile yapmak çok zor veya çok zahmetli olacaktır. Ayrıca, bir sonraki derlemede, sadece değişiklikten etkilenen dosyalar derlenmelidir. İşte bu iş için Makefile sistemi kullanılır.

Bu işleri yapan kaba bir Makefile aşağıdaki gibi yazılıp işletilebilir.

```
# Makefile.00
project: file1.o file2.o
gcc -Os -g -Wall file1.o file2.o -o project

file1.o: project.h file1.c file2.c
gcc -c file1.c -o file1.o

file2.o: project.h file2.c
gcc -c file2.c -o file2.o
```

```
install:
sudo cp -f project /usr/local/bin

uninstall:
sudo rm -f /usr/local/bin/project

clean:
rm -f project *.o


$ which project

$ make project
gcc -c file1.c -o file1.o
gcc -c file2.c -o file2.o
gcc -Os -g -Wall file1.o file2.o -o project

$ make install
sudo cp -f project /usr/local/bin

$ ls -l project
-rwxrwxr-x 1 nazim nazim 7377 Eyl 14 22:52 project

$ make clean
rm -f project *.o

$ ls -l project
ls: cannot access project: No such file or directory

$ which project
/usr/local/bin/project

$ project
program başarılı bitti.

$ make uninstall
sudo rm -f /usr/local/bin/project

$ which project

$ project
bash: /usr/local/bin/project: No such file or directory


# Makefile.01
PROG = project
CFLAGS= -Os -g -Wall
LDFLAGS = -lm
RM = rm -f
```

```
CP = cp -f
CC = gcc

all: $(PROG) install

$(PROG): file1.o file2.o
$(CC) $(LDFLAGS) $(CFLAGS) $^ -o $(PROG)

file1.o: file1.c project.h file2.c
$(CC) $(CFLAGS) -c $< -o $@

file2.o: file2.c project.h
$(CC) $(CFLAGS) -c $< -o $@

# Makefile.01

install:
sudo $(CP) $(PROG) /usr/local/bin

uninstall:
sudo $(RM) /usr/local/bin/$(PROG)

clean:
$(RM) $(PROG) *.o
```

```
$ make
gcc  -Os -g -Wall -c file1.c -o file1.o
gcc  -Os -g -Wall -c file2.c -o file2.o
gcc  -lm -Os -g -Wall file1.o file2.o -o project
cp -f project /usr/local/bin
```

Son verilen Makefile dosyası genel olmasına rağmen sadece native kod üretir. Yani üzerinde çalıştığı işlemci için kod üretir. Çapraz derleme için CROSS_COMPILE değişkeni Makefile içine eklenebilir. Çapraz derleme için son Makefile içinde CC değişkenini aşağıdaki gibi değştirmek yeterlidir.

```
CC = $(CROSS_COMPILE)gcc
```

Bu durumda aşağıdaki gibi hem native hem de cross derleme yapılabilir.

```
$ make   # native derler.
$ file project
```

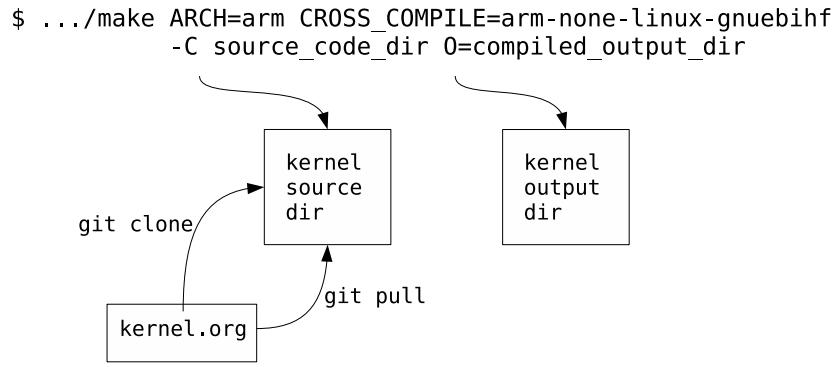
```
$ make CROSS_COMPILE=arm-linux-gnueabi-  # cross derler.
$ file project

$ make
gcc  -Os -g -Wall -c file1.c -o file1.o
gcc  -Os -g -Wall -c file2.c -o file2.o
gcc  -lm -Os -g -Wall file1.o file2.o -o project
cp -f project /usr/local/bin

$ file project
project: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
         dynamically linked (uses shared libs),
         for GNU/Linux 2.6.24,
         BuildID[sha1]=0xdfdc2158739556f903b8735284e5d2ec561dcc7a,
         not stripped

$ make clean
rm -f  project *.o

$ make CROSS_COMPILE=arm-linux-gnueabi-
arm-linux-gnueabi-gcc  -Os -g -Wall -c file1.c -o file1.o
arm-linux-gnueabi-gcc  -Os -g -Wall -c file2.c -o file2.o
arm-linux-gnueabi-gcc  -lm -Os -g -Wall file1.o file2.o -o project
sudo cp -f project /usr/local/bin

$ file project

project: ELF 32-bit LSB executable,
            ARM, version 1 (SYSV),
            dynamically linked (uses shared libs),
            for GNU/Linux 2.6.32,
            BuildID[sha1]=0x3f611dee63bb4b3489118e328c2c626d200f1467,
            not stripped


# Sadece güncellemeden etkilenen dosyalar derlenir.

$ make clean
$ vi file1.c
$ make
gcc  -Os -g -Wall -c file1.c -o file1.o
gcc  -lm -Os -g -Wall file1.o file2.o -o project
sudo cp -f project /usr/local/bin
```

Büyük projeler için el yordamı ile Makefile kurmak çok zahmetlidir. Büyük Makefile dosyaları genelde ide veya otomatik Makefile araçları tarafından kurulur. Autotools,CMake vs.

Bu tür programlarla kurulan paketler genelde aşağıdaki gibi derlenip kurulabilirler. İlk 3 satır mutlaka mevcuttur.

```
$ ./configure
$ make
$ sudo make install

$ make test
$ make clean
$ make mrproper
$ sudo make uninstall
```

u-boot, kernel, busybox ve buildroot derlenirken -C ve O= seçeneklerinin kullanılması önemle tavsiye edilir.



Şekil 9.12: Makefile'da -C ve O= kullanımı.

9.33 git

Linux topluluğu tarafından kullanılan en yaygın proje dağıtım ve sürüm kontrol sistemidir.

Temel olarak ssh sistemini kullanır. Artık neredeyse bütün yazılımlar git üzerinden dağıtılmaktadır. Şimdilik clone/pull kullanımı bize yeterli olacaktır.

```
$ git clone git://git.buildroot.net/buildroot
$ git pull
$ git branch
$ git branch -r
$ git branch -a
$ git checkout -b builroot-at91
```

clone edilen dizin geliştirme içinkullanılmamalıdır. -C ve O= ile derleme yapılmalıdır.

buildroot paketinin, /project/buildroot altına clone ile alındığını kabul edelim. buildroot sistemi aşağıdaki gibi derlenebilir ve sonuçlar /nk/workspace/out içine atılır. Böylece orijinal buildroot sisteminde hiç bir güncelleme yapılmaz. Daha sonra git pull ile sorunsuzca güncelleme yapılabilir.

Ayrıca birden fazla proje aynı buildroot sisteminin kaynak kodunu birbirlerine karıştırmadan kullanabilir.

```
# -C change directory
# O= output

$ make -C /project/buildroot O=/nk/workspace/out menuconfig
$ make -C /project/buildroot O=/nk/workspace/out
```

9.34 /etc/inittab

Çekirdek, kök dosya sistemini bağladıktan sonra, genelde /sbin/init programını işletir. Bu program sistemi ayağa kaldırır. Sistemin ayağa kalkma şekli /etc/inittab dosyası tarafından tespit edilir.

busybox'ın kullandığı inittab dosyası standard inittab dosyasından çok farklıdır. Run Level kavramı yoktur. Zaten gömülü sistemler için bu kavrama gerek de yoktur.

```
# Startup the system
null::sysinit:/bin/mount -t proc proc /proc
#null::sysinit:/bin/mount -o remount,rw / # REMOUNT_ROOTFS_RW
null::sysinit:/bin/mkdir -p /dev/pts
null::sysinit:/bin/mkdir -p /dev/shm
null::sysinit:/bin/mount -a
null::sysinit:/bin/hostname -F /etc/hostname

# now run any rc scripts
::sysinit:/etc/init.d/rcS

# Put a getty on the serial port
ttyO2::respawn:/sbin/getty -L ttyO2 115200 vt100 # GENERIC_SERIAL

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

# Stuff to do before rebooting
null::shutdown:/etc/init.d/rcK
null::shutdown:/bin/umount -a -r
null::shutdown:/sbin/swapoff -a   # !!!
```

9.35 /etc/passwd

```
$ ls -l passwd
-rw-r--r-- 1 root root 596 May 31 23:29 passwd

$ ls -l shadow
-rw------- 1 root root 380 Ağu  4 09:38 shadow

Örnek satırlar:
root:x:0:0:root:/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh

Alanlar:
default:x:1000:1000:Default non-root user:/home/default:/bin/sh

default              : login name
x                    : encrypted password
1000                 : user ID
1000                 : group ID
Default non-root user: comment field
/home/default        : home directory
/bin/sh:             : user command
```

Şifre kolonu boşsa, kullanıcı şifresiz girebilir. Şifre kolonunda x varsa, şifre shadow içinde saklıdır.

Son kolon çok önemlidir. Login/Passwd geçildikten sonra çalıştırılacak komuttur. Genelde /bin/sh veya /bin/false seçilir. Buradaki değer, SHELL değişkenine otomatik olarak atanır. Boşsa /bin/sh kabul edilir.

HOME değişkeni, "home directory" kolonu olarak atanır. Giriş yapıldıktan sonra geçilecek ilk dizindir.

```
$ man 5 passwd
$ man passwd
$ man adduser
```

9.36 /etc/shadow

Şifre bigisi ve sürelerini saklar. Sadece root erişebilir. passwd dosyasındaki her bir şifreli kullanıcı için burada bir kayıt mevcuttur.

```
$ man 5 shadow
$ man passwd

$ cat shadow

root:$1$V4H6aNhe$jZWUzp9iO8sJVIXpYBjEH.:10933:0:99999:7:::
bin:*:10933:0:99999:7:::
daemon:*:10933:0:99999:7:::
adm:*:10933:0:99999:7:::
lp:*:10933:0:99999:7:::
sync:*:10933:0:99999:7:::
shutdown:*:10933:0:99999:7:::
halt:*:10933:0:99999:7:::
uucp:*:10933:0:99999:7:::
operator:*:10933:0:99999:7:::
ftp:*:10933:0:99999:7:::
nobody:*:10933:0:99999:7:::
default::10933:0:99999:7:::
```

root login name

1V4H6aNhe$jZWUzp9iO8sJVIXpYBjEH. ! veya * varsa passwd ile giriş yapılamaz. Ama başka türlü login olabilir. Şifresiz giriş için boş bırakılabilir. ! ile başlarsa şifre kilitlidir.

:10933:0:99999:7::: Zaman bilgileri

9.37 /etc/group

```
$ man 5 group
$ man groupadd

$ cat /etc/group

adm:x:4:nazim
tty:x:5:
disk:x:6:
cdrom:x:24:nazim,can,ahmet
```

cdrom group name

x password

24 GID

nazim,can,ahmet User list.

9.38 /etc/init.d

buildroot sistemi aşağıdakine benzer bir rcS betiği ile sistemi ayağa kaldırır.

```
$ cat /etc/init.d/rcS

#!/bin/sh

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

rcK betiği ile sistemi kademeli olarak, açılışın ters sırasına göre kapatır.

```
$ cat /etc/init.d/rcK

#!/bin/sh

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

S??* dosyası örneği:

```
#!/bin/sh

case "$1" in
  start)
        echo "Starting network..."
        /sbin/ifup -a
        ;;

  stop)
        echo -n "Stopping network..."
        /sbin/ifdown -a
        ;;

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

```
        ;;

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

esac
exit $?
```

9.39 Atomik güncelleme

Prensip olarak bütün dosya sistemleri RO bağlanmalıdır. RW olması gereken dosyalar tmpfs'e yönlendirilebilir. Fakat güç kesilince bu dosyalar yok olur.

Uygulamaların config dosyaları gibi bazı dosyaların kalıcı olması istenir. Kalıcı dosyalar illa ki RW modunda bir bağlı bir bölüm üzerinde tutulmalıdır.

RW modunda bağlı bir bölüm üzerinde güvenli güncelleme yapabilmek için "atomic update" denilen çok basit bir yöntem uygulanabilir. Bu yöntem ani kapanma gibi durumlarda dosyanın bütünlüğünü korur.

Unutulmamalıdır ki ubifs gibi kapanmaya dayanıklı sistemler dosya değil file system bütünlüğünü sağlarlar. Diğer bir deyişle, ani kapanmalardan sonraki mount işleminin kesin olarak yapılabileceğini garantilerler ama dosya içeriği konusunda bir garanti sunmazlar.

Dosya içeriğini korunması konusundaki garanti "atomic update" ile sağlanır. Ubifs ile dosya sistemi ve "atomic update" ile de dosya bütünlüğü korunmuş olur.

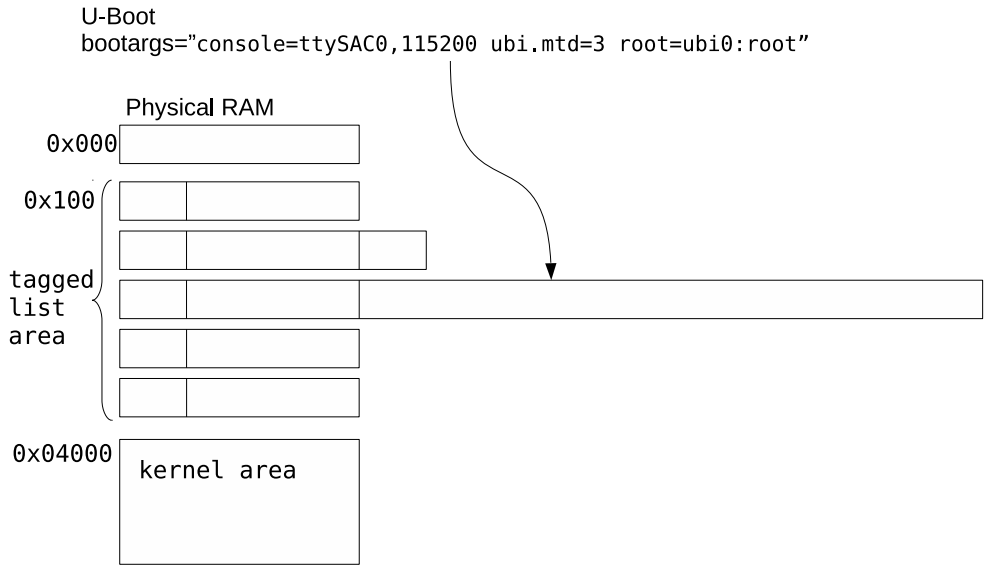
F1 dosyası güncellenecek olsun.

```
F2= copy(F1)
update(F2)
sync
mv F2 F1
```

Bu yöntemde, ani kapanmalarda F1 bozulmaz. Çünkü bütün güncelleme F2 üzerinde yapılır. Sadece, 4. adımda, mv sırasında, dosya ismi değiştirme sırasında tehlike vardır. Fakat posix standardı, rename veya move işleminin atomik olması gerektiğini söyler ve ani kapanmalarda mv işlemi F1'i bozamaz. ubifs ve diğer pek çok dosya sistemi posix standardını sağlar.

Canlı dosyalar veya büyük dosyalar bu tekniğe uygun değildir.

9.40 Çekirdek Parametre Alanı

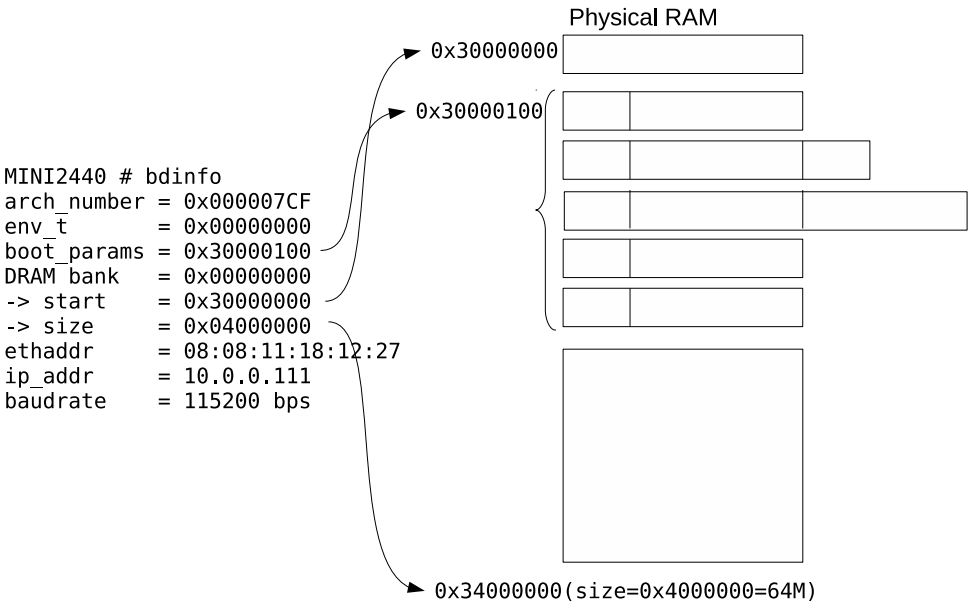


Şekil 9.13: Çekirdek parametre listesi.

Parametre listesi fiziksel RAM belleğin başlangıç adresinden sonra, tam 0x100 adresinden başlar. Parametre listesinin ilk elemanı CORE ve son elemanı NONE olmak zorundadır. MEM zorunludur. Diğer etiketlerin hiç bir olmayabilir.

Parametre listesi 0x4000 adresini geçmemelidir. Linux gözü kapalı biçimde 0x4000 adresinden sonrasını kullanmaya başlar, parametreler ezilebilir.

Parametreler u-boot tarafında bootargs ile, çekirdek derlemesinde "boot parameters" ile, modül yüklerken modül parametreleri şeklinde verilebilir.



Şekil 9.14: bdinfo'daki adresler.

9.41 Modül parametreleri

insmod sırasında modüllere parametreler var=val şeklinde atanabilir. Modül olmayan, derleme sırasında çekirdeğe doğrudan eklenen cihazlara ise parametreler modül_ismi.var=val şeklinde verilir.

Dinamik modüllere, henüz yüklü olmadığı için açılış sırasında parametre aktarılamaz. Yürütme zamanında değişiklik için bakınız, /sys/module/usbcore/parameters

```
$ insmod usbcore blinkenlights=1

> setenv bootargs \... usbcore.blinkenlights=1"

$ lsmod

$ modinfo snd

> setenv bootargs \ ... snd.debug=0"

$ modprobe snd debug=0

$ cd /sys/module/snd/parameters
$ ls -l

$ cat debug

$ echo \1" > debug   # Diğer dosyalar ro durumundadır.
```

9.42 U-Boot değişkenleri

autoload yes/no no ise imaj yüklenmez, yes ise tftp ile imaj yüklenir. Her durumda ağ yapılandırması yapılır. Diğer bir deyişle tftp otomatik olarak işletilir. Sadece rarpb, bootp ve hdcp için geçerlidir.

autostart yes/no bootm otomatik olarak işletilir. Sadece rarpb, bootp, dhcp, tftp disk, docb için geçerlidir.

baudrate Konsolun hızı, varsayılan 115200

bootargs cmdline bilgisi.

bootcmd bootdelay sonrası çalışan autoboot betiği.

bootdelay bootcmd'yi yürütmeden önceki bekleme süresi. 0, bekleme yok; -1, autoboot iptal edilir.

bootfile tftp'nin yükleyeceği dosyanın ismi.

ipaddr bord'un ipadresi, tftp kullanılır.

loadaddr tftp'nin yükleme yapacağı bellek adresi.

ethaddr Eternet adresi, salt okunur.

mtdparts mtd bölümlendirme tablosu, bootargs ile çekirdeğe gönderilir. u-boot ve kernel aynı tabloyu kullanır.

serverip tftp sunucusunun bulunduğu adres.

Bazı env değişkenleri otomatik olarak bootp, dhcp ve tftp tarafından atanırlar. Bu değişkenler genelde ağ yapılandırması ile ilgili olanlardır.

filesize değişkeni, en son kullanılan bootb, dhcp veya tftp ile komutları tarafından güncellenir. Bunun dışında borda veya kullanıcıya özgü pek çok değişken olabilir.

Değişkenler 3 farklı biçimde saklanabilirler.

1. u-boot değişkenleri NAND tarafında keyfi bir NAND bölümünde tutulabilriler.. Genelde bu bölüme env veya uboot-env isimleri verilir. Bu bölüm her zaman raw formatındadır, dosya sistemi barındırmaz.

```
> set dynpart addr
> saveenv
```

Yukarıdaki ilk komut ile bu bölümün adresi u-boot'a tanıtılır. Sonraki komut ile de mevcut değişkenler env bölümüne kalıcı olarak yazılır. Eğer bu bölüm bozulursa, kaynak kodu içine gömülü olan değişkenler kullanılır.

2. u-boot değişkenleri kaynak kodu içine gömülebilirler. Eğer NAND/MMC'de bulunan değişkenler okunamaz ise, kod kod içinde gömülü olan değişkenler kullanılır ve bu durum açılışta ikaz mesajı olarak gösterilir.

   Her borda ait header dosyası farklı bir yer ve isimde olabilir. Kaynak kodunda, boards.cfg içinde desteklenen bordlar ve config dosyalarının isimleri bulunabilir.

   env değişkenleri bbb için include/configs/am335x_evm.h dosyası içine yazılabilir. Bu dosya her bord için farklıdır. Nihai sistemde env yerine doğrudan header dosyası içinde tanım yapılabilir.

3. u-boot değişkenleri MMC'den yüklenebilir. Genelde, MMC 1. bölümünde, uEnv.txt dosyası içine yazılır. Açılış sırasında uygun bir betik uEnv.txt dosyası içindeki değişkenleri okur.

   uEnv.txt dosyası içinde ubootcmd komutu varsa, bu komutu tomatik işletir. "mmc 0:1= /dev/mmcblk0p1"

   Yükleminin genel yapısı aşağıdaki gibidir.

```
> load mmc 0:1 addr uEnv.txt
> env import -t addr size
```

9.43 U-Boot Programları

U-Boot derlendikten sonra pek çok program üretilir.

MLO Farklı isimleri de olabilir. u-boot.bin programının çok basit bir halidir. Genelde u-boot.bin programını yükleyecek cihazları ayağa kalıdırır ve u-boot.bin programını yükler. 1. seviye boot yükleyicisi de denir.

Genelde tek bir yükleyici her cihazdan boot edemez. Seri, USB, Eternet, NAND, MMC'den boot edebilmek farklı farklı derlenir. Her cihazda MLO olmayabilir. MLO genel değildir ve borda bağlıdır.

u-boot.bin raw binary programdır. ARM işlemcisi tarafından doğrudan yürütülür. 2. seviye yükleyicidir. Cihazdan hemen hemen bağımsızdır, çok geneldir.

Borda ait bütün cihazları ayağa kaldırır. Gömülü Linux projelerinde, u-boot'un ayağa kalkması çok önemli bir aşamadır. Çünkü u-boot öncesi meydana gelen hataların tespit edilmesi çok zordur. u-boot'tan sonrası kullanıcı ile çok iyi bir etkileşime sahiptir ve hatalar çok kolay tespit edilebilir.

u-boot.img u-boot.bin programının u-boot imajı haline getirilmiş halidir.

```
u-boot.img= image_header + u-boot.bin
```

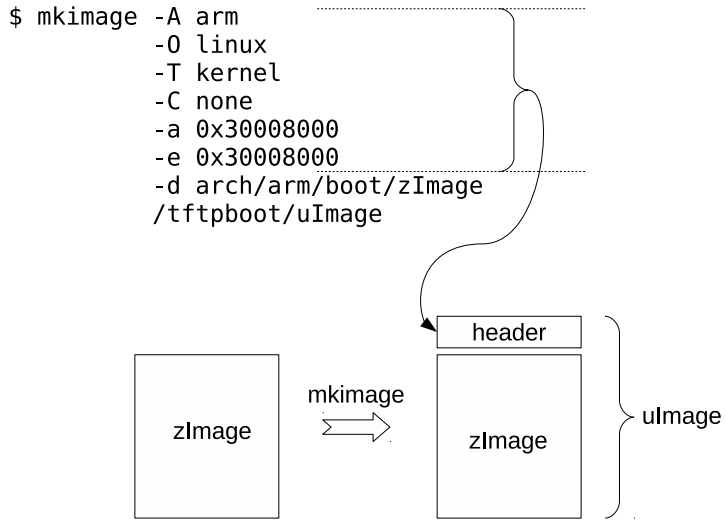
Bazı bordlar illa ki u-boot.img programını isterler.

tools/mkimage Host tarafında çalışan, u-boot imajı üretme programıdır. Genelde /usr/local/bin gibi bir yere kopyalanır. Çok basit bir işleve sahiptir. Herhangi bir dosyanın tepesine 64 baytlık[10] bir alan ekler. Dosyanın içeriğine hiç karışmaz. Dosya ASCII dahi olabilir.

```
mkimage [-x] -A arch -O os -T type -C comp -a addr -e ep
             -n name -d data_file[:data_file...] image

          -A ==> set architecture to 'arch'
          -O ==> set operating system to 'os'
          -T ==> set image type to 'type'
          -C ==> set compression type 'comp'
          -a ==> set load address to 'addr' (hex)
          -e ==> set entry point to 'ep' (hex)
```

[10]Çoklu imajlarda bu boy daha büyüktür.

```
            -n ==> set image name to 'name'
            -d ==> use image data from 'datafile'
            -x ==> set XIP (execute in place)
```

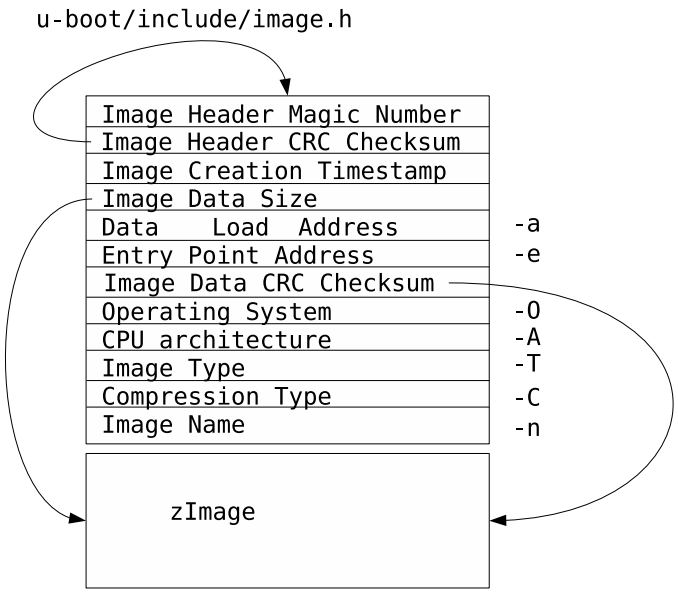


Şekil 9.15: mkimage programının argümanları

Her bord üreticisi u-boot üzerinde borda uygun güncellemeleri yapar. Her bord için bir config dosyası bulunur. Bu dosyanın ismi el kitaplarından veya borda ait örnek betiklerden veya boards.cfg dosyasından elde edilmelidir.

u-boot, kernel ve buildroot için önerilen toolchain kullanılmalıdır. Bazı bordlar için u-boot, kernel ve br için farklı toolcahin'ler önerilebilir, bu öneriye sadık kalınmalıdır.

Özellikle toolchain için önerilen sürüm numaralarına dikkat edilmelidir. Aynı derleyicinin farklı sürümleri dahi sorun yaratmaktadır.



Şekil 9.16: U-Boot imajının başlığı

9.44 U-Boot örnekleri

9.44.1 scripts

```
# test.scr
echo
echo bu bir deneme betidir.
echo MMC icinde ne var?
mmcinit
fatls mmc 0:1
echo

# Host tarafında:
$ mkimage -T script -C none -n 'test betiği' -d test.scr /tftpboot/script.img

# Bord tarafında:
> tftp 32000000 script.img
> iminfo 32000000
> autoscr 32000000
```

9.44.2 base, memory display

Bütün adreslerin başına bu değer eklenir. Varsayılan değer 0'dır.

```
> base 32000000
Base Address: 0x32000000

> tftp 0 script.img
> md.b 0 64
> md.w 0 64
> md.l 0 64

# b: byte, 8 bit.
# w: word, 16 bit
# l: long word, 32 bit
```

9.44.3 bdinfo

```
> bdinfo
   arch_number = 0x000007CF
   env_t       = 0x00000000
```

```
boot_params = 0x30000100 <-- kernel parameters için.
DRAM bank   = 0x00000000
-> start    = 0x30000000 <-- Çok önemli, +0x100 çekirdek
                            parametersi listesinin başlangıç
                            adresi.
-> size     = 0x04000000 <-- 64M
ethaddr     = 08:08:11:18:12:27
ip_addr     = 10.0.0.111
baudrate    = 115200 bps
```

9.44.4 boot, bootd, bootm

boot: run bootcmd'nin kısa hali

bootd: run bootcmd'nin başka bir kısa hali

```
# bootm:
> bootm kernel
> bootm kernel initramfs
> bootm kernel initramfs fdt
> bootm kernel - fdt
```

9.44.5 bootp, dhcp

```
> bootp    # veya
> dhcp
```

Her iki komut ile de ağ yapılandırması otomatik olarak yapılır. Bu yapılandırma sonrasında, ağ ile ilgili env değişkenleri otomatik olarak atanır. Eğer outoload değeri yes ise, tftp üzerinden boot dosyası yüklenir. Eğer autostart yes ise yükleme sonrası otomatik olarak bootm işletilir.

9.44.6 compare, crc, coninfo

```
> cmp a1 a2 0x32
```

a1 ve a2 adreslerini karşılaştırır. İlk farklı adres raporlanır. cmp.w kabul edilir.

```
> crc32 address count [addr]
```

CRC32 hesapla ve addr'ye sonucu yaz.

```
> coninfo
   serial   80000003 SIO stdin stdout stderr

# S: system device, I: input,  O:output
```

9.44.7 copy

```
> cp a b count

> bootp
Load address: 0x32000000

> iminfo 34000000
## Checking Image at 34000000 ...
Unknown image format!

> cp.b 32000000 34000000 2081a4
                         ^
                         çekirdeğin boyu

> iminfo 34000000
...
Verifying Checksum ... OK
```

9.44.8 dynenv, dynpart

dynenv u-boot değişkenlerinin saklanacağı NAND cihazının başlangıç adresini tanımlar veya sorgular.

dynpart Her NAND bölümünün ofset değerini hesaplar. U-Boot içinde gömülü olan bölümlendirme bilgisini kullanarak mtdparts değişkenini otomatik olarak atar.

9.44.9 go

Bare metal[11] cihazlarda arm kodu işletmek içindir. u-boot kaynak kodu içinde examples/ dizini altında hello_world.c mevcuttur, incelenebilir.

u-boot.bin programı da go ile başlatılabilir.

```
> tftp 32000000 hello.bin

> go 32000000
  ## Starting application at 0x32000000 ...
  Hello World
  Hit any key to exit ...

  ## Application terminated, rc = 0x0
```

9.44.10 itest

```
> mw 200000 0

> while itest *200000 == 0
> do
>   tftp 200000 tqm5200/uImage
>   echo === done ====
> done

> check_ub_ver=if itest.s \${tmp} == \${ver};
     then
       echo equal;
     else
       echo diff ;
     fi

> run check_ub_ver
```

9.44.11 loop, memory display, memory test

loop:

```
> loop.l 32000000 0xF
```

[11] işletim sistemi olmadan

16 adet long veriyi sonsuz kere okur. Bir tür read testidir. Ancak reset ile durur.

md: memory display

```
> tftp 32000000 script.img
> md.b 32000000 ff
> md.w 32000000 ff
> md.l 32000000 ff
```

mtest: simple memory test NAND veya EPROM'a uygulanamaz. RAM'ı değiştirerek test eder. Çok uzun sürer.

9.44.12 memory write

```
 > md.b 32000000 10
   32000000: da fd ff ff d9 fd ff ff d8 fd ff ff d7 fd ff ff    ................

> ? mw
mw [.b, .w, .l] address value [count]
    - write memory

> mw.b 32000000 FF 10

> md.b 32000000 10
32000000: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
```

9.44.13 memory modify

Adresin değerini gösterir. Yeni veri bekler. Boş enter yapılırsa veri değişmez. Geçerli bir veri girilirse adrese atanır. Geçersiz bir veri girilirse işlem biter.

```
> mm.b 32000000
  32000000: 00 ? a
  32000001: 01 ? b
  32000002: 02 ? c
  32000003: 03 ? .
> md.b 32000000 F
  32000000: 0a 0b 0c 03 04 05 ff ff ff ff ff ff ff ff ....
>
```

nm'de adres artımı olmaz. Daha çok device register testi içindir.

9.44.14 NFS

```
> setenv bootargs console=ttySAC0,115200
> nfs 32000000 10.0.0.4:/tftpboot/uImage.initramfs   veya
> nfs 32000000 /tftpboot/uImage.initramfs
> bootm
```

bootm komutundan sonra login gelir. Eğer setenv autostart yes girilirse, bootm yapmaya gerek yoktur.

```
/etc/exports:
/tftpboot 10.0.0.114(rw,insecure,no_subtree_check,async,no_root_squash)
```

exports dosyasında yapılan her güncellemeden sonra exportfs komutu girilmelidir.

```
$ exportfs -avr
```

9.44.15 rarpboot

arp, MAC-IP eşlemesi yapar.

```
$ mkdir /var/lib/arpd
$ arpd eth0
$ arp -n -i eth0
```

incomplete: ip var ama karşı gelen bir mac yok. arp soru göndermiş ama kimse yanıt vermemiş.

tcpdump ile inceleme:

```
$ tcpdump -i eth0
```

```
19:32:21.132675 ARP, Request who-has 10.0.0.111 tell nkoc.local, length 28
19:32:21.133068 ARP, Reply 10.0.0.111 is-at 08:08:11:18:12:27 (oui Unknown), length  46
19:32:26.129722 ARP, Request who-has nkoc.local tell 10.0.0.111, length 46
19:32:26.129778 ARP, Reply nkoc.local is-at 00:90:f5:dc:51:d2 (oui Unknown), length 28
```

ARP tablosuna 10.0.0.114'e karşı gelen IP girilmelidir. Ki bord boot ederken kendi mac adresini yayınlayacak ve buradan IP elde edecektir.

```
$ arp -d 10.0.0.111  # Eskileri sil.
$ arp -i eth0 -s 10.0.0.114 08:08:11:18:12:27
$ arp -i eth0 -n
```

Test: rarp ile adres alır, tftp ile yükleme yapar.

```
> rarpboot 32000000 /tftpboot/uImage.initramfs
```

9.45 BR

Tek başına busybox sistemi, gelişmiş bir gömülü sistem kurmak için yeterli olmayabilir.

Bu durumda kurulacak her bir paket veya kütüphane, kaynak kodundan indirilmeli, ilgili mimari için yamaları geçilmeli, çapraz derlenmeli, RootFS içine install edilmeli, gerekli kütüphaneler install edilmeli, varsa açılış betikleri kurulmalıdır.

Bir çok paket için bu işi yapabilmek son derece zahmetlidir.

Bütün bu işleri otomatik olarak yapan programlar vardır. Şu anda bu programların içinde en yaygın olanı br'dir. br tamamen betiklerle yazılmıştır.

BuildRoot veya kısaca br, Gömülü Linux Sistemi kurma aracıdır.

Toolchain, RootFS, library, kernel ve bootloader imajı kurar.

İstenilen paketler busybox veya kernel derlemesi gibi tek tek seçilebilir.

Seçilen paketler, doğrudan kaynak kodundan otomatik olarak indirilir ve derlenir.

Qt gibi paketlere sahip bir RootFS derlemek için peygamber sabrına gerek vardır.

Her ne kadar kernel ve u-boot için de desteği olmasına rağmen bu desteklerin kullanılması tavsiye edilmez. Bordun el kitaplarından bakılarak u-boot ve linux için gerekli çalışma yapılmalıdır.

Pratikte sadece RootFS kurmak için kullanılır.

Pek çok formatta RootFS kurulabilir. Örneğin doğrudan NAND'a yazılabilecek ubi imajı kurulabilir. Fakat bu tür kullanım çok katı olduğu ve güncellenmesi çok zor olduğu için tavsiye edilmez.

RootFS sistemi rootfs.tar olarak kurulabilir. Bu durumda tar dosyası açılarak güncelleme yapılabilir.

700'den fazla pakete destek verir.

Her 3 ayda bir kararlı sürüm yayınlanır.

menuconfig ile paketler ve diğer özellikler seçilir. Nihayetide .config dosyası üretilir.

make ile derleme yapılır. Gece yatmadan önce derleme verilmesi tavsiye edilir.

Kaynak kodları dl/ dizini altına indirilir.

Nihai imajlar output/images altında kurulur. Şu anda sadece rootfs.tar ile ilgilenmekteyiz.

rootfs.tar dosyası uygun bir dizine açılır ve gerekli güncellemeler yapılabilir.

RootFS esas itibari ile mevcut bir iskelet RootFS üzerine kurulur. İsterse kullanıcı kendi iskelet sistemini kurabilir.

O= seçeneği kullanılmazsa, bütün çıkışlar kaynak kodu içinde output/ dizini altına yapılır.

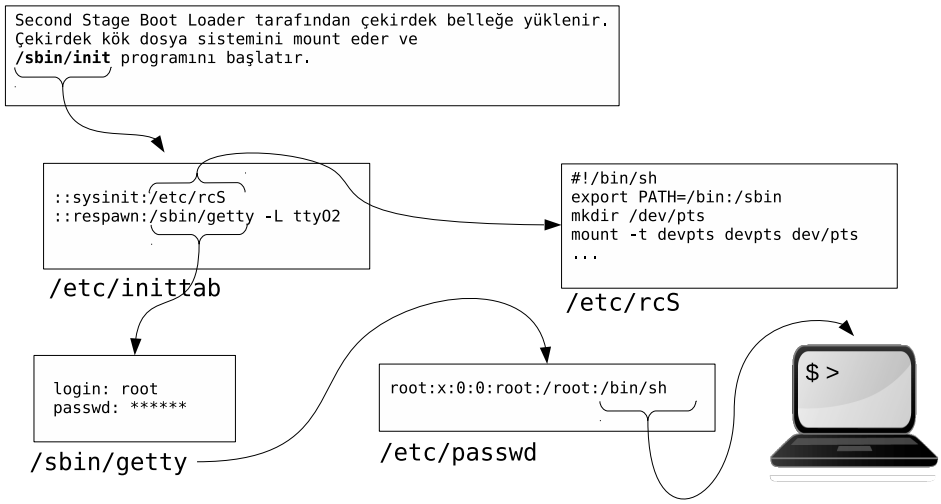
stages/ dizini çok önemlidir. Host tarafında derleme yapılırken burada üretilen kütüphaneler kullanılacaktır.

Mevcut bir toolchain kullanılabilir veya br'nin borda uygun bir toolchain üretmesi istenebilir.

BR tarafından RootFS aşağıdaki gibi otomatik olarak üretilir. output/target içine fs/skeleton içinde bulunan iskelet RootFS kopyalanır. Gerekli kütüphaneler kopyalanır. Paketler install edilir. Varsa post-build betikleri işletilir. ubifs, jffs2 gibi RootFS imajı üretilir.

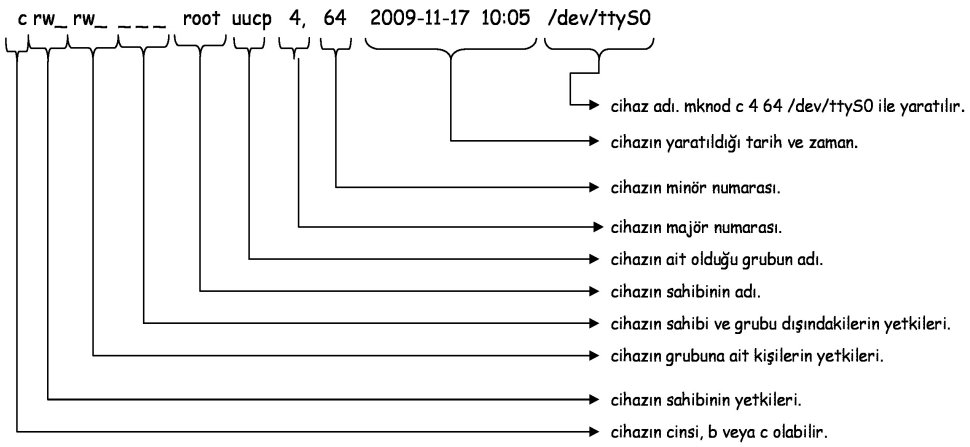
Çalışabilir bir RootFS elde edildikten sonra mutlaka .config dosyası ayrı bir yerde saklanmalıdır.

9.46 Açılış Sırası



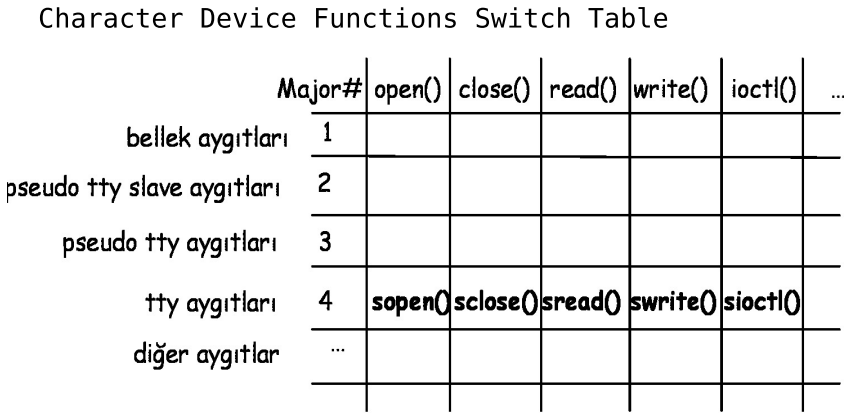
Şekil 9.17: Açılış sırası.

9.47 /dev dizini



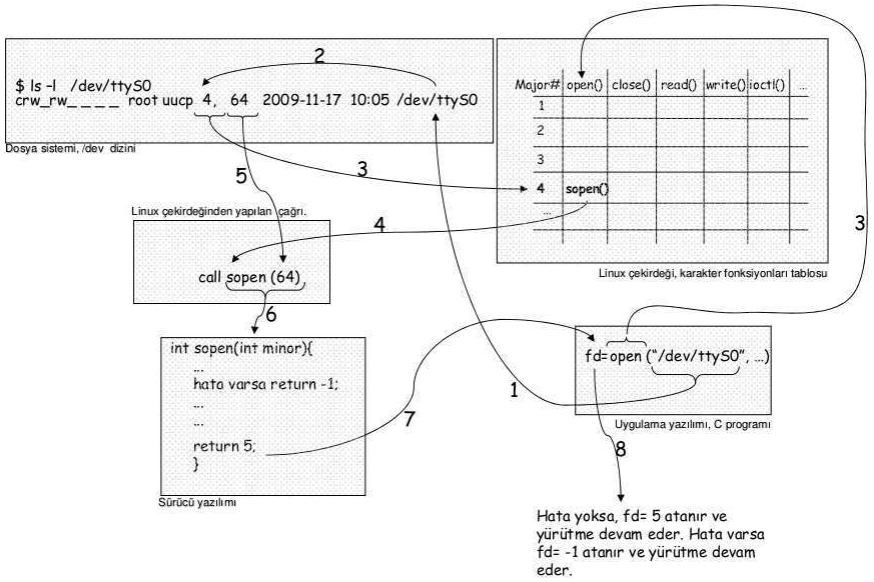
Şekil 9.18: /dev dizini.

9.48 VFS, open/read

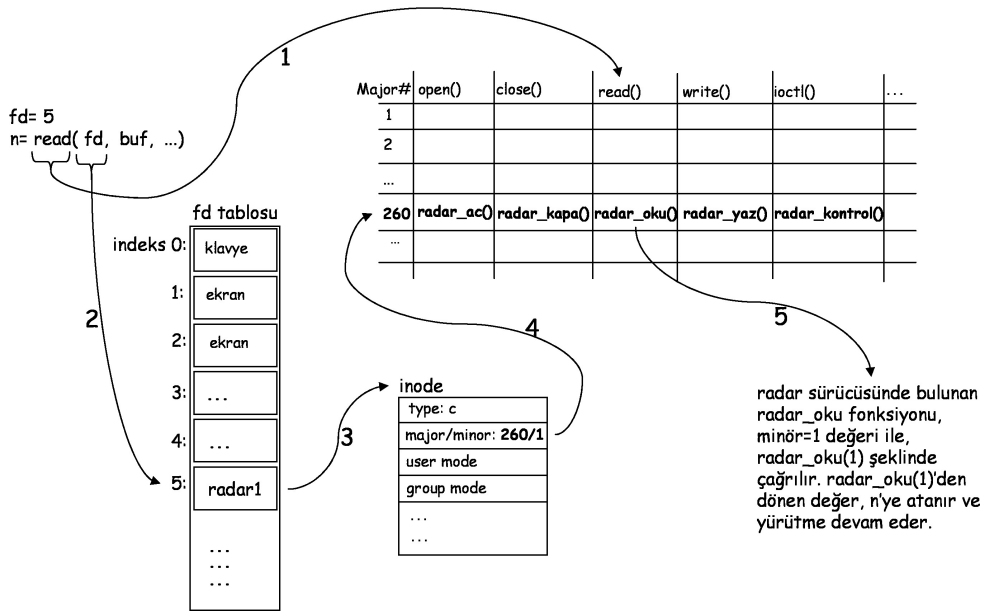


Şekil 9.19: VFS, switch table.

Geniş bilgi için bakınız, https://www.kernel.org/doc/Documentation/devices.txt Major device için 12 bit, minor device için 20 bit'lik alan ayrılmıştır. Her iki sayı, toplamda 32 bit alan kaplar.



Şekil 9.20: VFS, open.



Şekil 9.21: VFS, read.

9.49 bootp/dhcp

```
$ dhcpd eth0 -f -d

   Internet Systems Consortium DHCP Server 4.2.4
   Copyright 2004-2012 Internet Systems Consortium.
   All rights reserved.
   For info, please visit https://www.isc.org/software/dhcp/
   Wrote 0 deleted host decls to leases file.
   Wrote 0 new dynamic host decls to leases file.
   Wrote 0 leases to leases file.
   Listening on LPF/eth0/00:90:f5:dc:51:d2/10.0.0.0/24
   Sending on   LPF/eth0/00:90:f5:dc:51:d2/10.0.0.0/24
   Sending on   Socket/fallback/fallback-net

   DHCPDISCOVER from 08:08:11:18:12:27 via eth0
   DHCPOFFER on 10.0.0.113 to 08:08:11:18:12:27 via eth0
   DHCPREQUEST for 10.0.0.113 (10.0.0.4) from 08:08:11:18:12:27 via eth0
   DHCPACK on 10.0.0.113 to 08:08:11:18:12:27 via eth0

# Ya da aşağıdaki gibi başlat ve /var/log/syslog'a bak.
$ service isc-dhcp-server start

# Ayrılan IP'ler için bak: /var/lib/dhcp/dhcpd.lease
```

DHCP sunucusu için çok basit bir yapılandırma dosyası aşağıda verilmiştir.

```
# /etc/dhcp/dhcpd.conf:

ddns-update-style none;
allow bootp;
allow booting;

subnet 192.168.1.0 netmask 255.255.255.0{
}

subnet 10.0.0.0 netmask 255.255.255.0{
}

host test1{
      hardware ethernet 08:08:11:18:12:27;
      fixed-address 10.0.0.114;
      next-server 10.0.0.4;         # tftp server adresi.
      filename "uImage";            # \autoload yes" ise yüklenir.

      option host-name "test1";
      option domain-name "ucanlinux.com";
      option routers 10.0.0.4;
      option root-path "/tftpboot";
}
```

9.50 Terminaller

Grafik ekradan kara ekrana geçmek için Ctrl+Alt+F1 ... Ctrl+Alt+F6

Kara ekranda grafik ekrana geçmek için Alt+F7

Kara ekranlar arası dolaşmak için Alt+F1..Alt+F6

Kara ekran sayısını değiştirmek için /etc/inittab güncellenir.

9.51 Kabuğun çevre değişkenleri

Kabuk içinde bulunan bazı değişkenler bir tabloda saklanabilir. Bu tabloya çevre değişkenleri tablosu denir.

Tablonun amacı, kabuk içinde çalışan proseslerin bu değişkenelere erişebilmesini sağlamaktır. Çevre değişkenleri env komutu ile listenebilir. Kırpılmış örnek bir çıkış aşağıda verilmiştir.

```
$ env

SESSION=ubuntu
TERM=xterm
SHELL=/bin/bash
LC_NUMERIC=tr_TR.UTF-8
GIT_EDITOR=vim
USER=nazim
PATH=/nk/workspace/projects/linux.egitimi/bbb4/toolchain/arm/bin:...
PWD=/nk/workspace/projects/linux.egitimi/bbb4/latex
LANG=en_US.UTF-8
HOME=/nk
LANGUAGE=en_US
LOGNAME=nazim
LC_TIME=tr_TR.UTF-8
LC_NAME=tr_TR.UTF-8
OLDPWD=/nk/workspace/projects/linux.egitimi/bbb4/latex/ekler
```

Kabuk değişkenleri[12] genelde ufak harflerle, export'a girecek çevre değişkenleri[13] genelde büyük harflerle verilir.

```
# Listeleme
$ env

# Tabloya değişken ekleme
$ export EGITIM="linux eğitimi"
$ export A=3

# Inceleme
$ echo $EGITIM
$ echo A
```

[12] shell variables
[13] environment variables

```
$ echo $A
$ env | grep EGITIM

# Silme
$ export -n EGITIM
$ echo $EGITIM
$ env | grep EGITIM

# Tabloya sonradan değişken ekleme.
$ S1=abc
$ S1="$S1:abc"
$ export $1

$ unset S1
# Hem export tablosundan hem de kabuk içinden siler.
```

Bazı standard çevre değişkenleri aşağıda listelenmiştir.

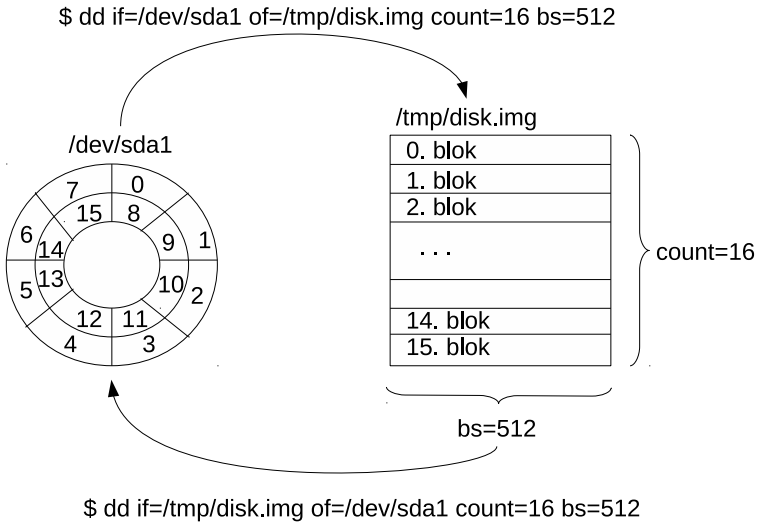
```
LD_LIBRARY_PATH: Shared library search path.
DISPLAY        : Screen id for X.
EDITOR         : Default editor.
HOME           : Current user home directory.
HOSTNAME       : Name of the local machine.
MANPATH        : Manual page search path.
SHELL          : Current shell name.
TERM           : Current terminal type.
USER           : Current user name.
PATH           : Search path for commands.
```

C tarafında çevre değişkenini elde etmek.

```
#include <stdlib.h>
char *getenv(const char *name);
```

9.52 dd

dd[14] komutu gömülü sistemlerle uğraşanlar için elzem bir komut olmasına rağmen çok ama çok dikkatli kullanılmalıdır.



Şekil 9.22: device dump.

[14]device dump. Dikkat edelim ki device destroy olmasın

9.53 stdin, stdout, stderr

Her proses[15], otomatik olarak stdin, stdout ve stderr dosyalarını açar. Çıkışta da otomatik olarak kapatır. Bu işlemler aşağıdaki gibi sembolize edilebilr.

```
#include <stdio.h>

int main(int argc, char *argv[]){

   FILE *stdin, *stdout, *stderr;

   stdin = fopen("/dev/pts/0", "r");
   stdout= fopen("/dev/pts/0", "w");
   stderr= fopen("/dev/pts/0", "w");  // dup(stdout)

   ...
   ...

   on_exit:
   flose(stdin);
   fclose(stdout);
   fclose(stderr);
   ...
}
```

Her proses, çılan her dosya için biricik sıra numarası verir. std* dosyalarının sıra numaralarai aşağıdaki gibi elde edilebilir. Düşük seviye I/O işlemlerinde doğrudan bu numaralara kullanılabilir.

[15]Bir programın çalışan örneğine proses denir.

```
$ vi fileno.c

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]){

   fprintf(stdout, "stdin  fileno: %d\n", fileno(stdin));
   fprintf(stdout, "stdout fileno: %d\n", fileno(stdout));
   fprintf(stdout, "stderr fileno: %d\n", fileno(stderr));

   return EXIT_SUCCESS;
}


$ ./fileno
stdin  fileno: 0
stdout fileno: 1
stderr fileno: 2

# Kabuğun kullandığı dosya numaraları.
# $$ ifadesi, kabuğun PID numarasını temsil eder.

$ ls -l /proc/$$/fd
total 0
lrwx------ 1 nazim nazim 64 Eyl  8 23:54 0 -> /dev/pts/0
lrwx------ 1 nazim nazim 64 Eyl  8 23:54 1 -> /dev/pts/0
lrwx------ 1 nazim nazim 64 Eyl  8 23:54 2 -> /dev/pts/0
lrwx------ 1 nazim nazim 64 Eyl  9 00:08 255 -> /dev/pts/0
```

9.54 Statik derleme ve soyma

Statik derlemede, kütüphaneler, derleme sırasında otomatik olarak kodun içine gömülürler. Kod gömülü sisteme taşınırken, tek başına taşınır. Herhangi bir kütüphane taşımaya ihtiyaç yoktur.

```
$ gcc -O2 -Wall -o dene dene.c

$ ls -l dene
-rwxrwxr-x 1 nazim nazim 7450 Eyl  9 21:15 dene

$ gcc -O2 -Wall -o dene --static dene.c

$ ls -l dene
-rwxrwxr-x 1 nazim nazim 779629 Eyl  9 21:16 dene


$ file dene

dene: ELF 32-bit LSB executable,
      Intel 80386,
      version 1 (GNU/Linux),
      statically linked,
      for GNU/Linux 2.6.24,
      BuildID[sha1]=0xcea9356c60acfd86e5a7c1eb0689186d74b1a2b1,
      not stripped

$ objcopy --strip-all dene

$ ls -l dene
-rwxrwxr-x 1 nazim nazim 709232 Eyl  9 21:24 dene

$ file dene
# stripped özelliğini gör!
```

Gömülü sistemlere yüklenen her program mutlaka strip[16] işleminden geçirilmelidir ki disk ve bellekte az yer kaplasın ve çabuk yüklensin.

[16] soyma, striptiz kelimesi de aynı köktendir.

9.55 Kabuk kavramı

Dosyadan veya standard girişten okuduğu komutları yürüten yorumlayıcıya veya yazılım aracına kabuk denir. Örneğin bash, ksh,...

Kabuk programlarına betik denir. Betik başlarında yorumlayıcı isimleri #! ile verilir.

```
#!/bin/bash
#!/bin/perl
#!/bin/benim_yorumlayici
```

İşletim sistemi, bir dosyanın başında #! ifadelerini bulursa, dosyanın geri kalan bütün satırlarını bu #! ifadesinin devamındaki programla yürütür.

Kabuğun temel yapısı aşağıdaki gibi özetlenebilir.

```
while(1){
     echo PS1
     read Command
     if ( Command == exit ) break;
     if ( & exist ) execute_background(Command);
               else execute_foreground(Command);
     }
```

9.56 Jobs

Her kabuk arka plandaki işleri 1'den başlayarak numaralar. Bu numaralar PID numarası gibi kullanılabilir. PID'den ayırd edebilmek için %No kullanılır.

```
$ jobs   # + current job, - prev. job

$ fg
$ fg %n

$ Ctrl + Z
$ bg

$ kill -l
$ kill %n
$ kill PID

$ ps -ef
$ pstree -p
$ top

$ ps -ux  # current user's processes.

$ ps -ef
$ ps -aux # List all.

$ ps -aux | grep nazim | grep bash

# PID: Process ID
# VSD: code+data+stack size
# RSS: KB in RAM
# TTY: Terminal
# R  : Runnable
# S  : Sleep
# W  : Paging
# Z  : Zombie
```

9.57 Düzenli İfadeler

```
.       herhangi bir karakter

[...]   içerdeki herhangi bir karakter

[^...]  içerde olmayan herhangi bir karakter, ^:not

^       satır başı

$       satır sonu

A*      A ifadesi 0 veya daha fazla meydana çıkarsa

A^      1 veya daha fazla

A?      0 veya 1 kez
...

# Örnekler:

$ ls /tmp/abc*[1234]

$ ls /tmp/abc*[1-4]

$ tail [^r-t]*.[1-4]

$ ps aux | grep -E 20[1-9]{2}   # A{N} A, N kere tekrar eder.
```

9.58 Bazı komutlar

```
$ ls -t     # time, en yeni dosya en önce
$ ls -S     # En büyük en önce
$ ls -lrt   # En yeni en dipte, r:reverse order

$ cat <<EOF > /tmp/giris
bu
bir
giriş
denemesidir.
Ctrl+D

$ less file1 file2  # more'dan daha yetenekli.

$ head -n file   # Varsayılan n=10

$ tail -n file
$ tail -f file   # Çok faydalı.

$ wc file  # line, word, char. counts. Genelde -l ile kullanılır.

$ grep pattern file    # İçinde pattern geçen satırları listeler.
$ grep -v pattern file # pattern geçmeyen satırları listeler.
$ grep -c pattern file # pattern geçen satırları sayar.

$ sort < file1 > file2
$ sort -r > file2    # reverse sort.
$ sort -ru > file2   # reverse, unique

$ tee file  # Çıkışlar, aynı anda  hem stdout hem de
            # dosyaya aktarılır.

$ time command
real(actual elapsed time)
user(CPU time, program)
sys(CPU time, kernel)
real= user + sys + (i/o wait + running other task)

$ which command
$ which falan

$ du -ks *   # disk usage

$ df         # disk free for the file system
$ df .
$ df -h      # h: human readable
```

```
# *.gz
$ gzip file     # Dizin sıkıştıramaz, genelde tar ile kullanılır.
$ gzip -9 file
$ gzip -9 prog -c > prog.gz
$ file prog.gz
$ gunzip prog.gz

# *.bz2
$ bzip2 file    # gzip'den %20-25 daha iyi.
$ bunzip2 file


# *.lzma
$ lzma file      # bz2'den %10-20 daha iyi.
$ unlzma file

# tape archive
# c: create
# v: verbose
# f: file
# t: test
# x: extract
$ tar cvf test.tar file1 file2 dir1 dir2 dir3 ...
$ tar tvf test.tar
$ tar xvf test.tar

# tar ve compress ortak kullanılabilir.
# z: *.gz
# j: *.bz2
# --lzma: *.lzma
# - : output file is stdout
$ tar zcvf etc.tar.gz /etc
$ tar cvf - /etc | gzip -9  -c  > etc.tar.gz
# ASCII to PS
# Özellikle bilgisayar programlarını yazıcıdan kullanışlı biçime
# almak için kullanılır.
$ a2ps test1.c test2.c test.h -o test.ps
$ ps2pdf test.ps
$ gv test.ps
$ evince test.pdf
$ xpdf test.pdf
$ pdf2ps test.pdf

# \*.c" ile *.c farklıdır!
$ find . -name \*.c"
$ find . -name \*.c" -exec evince {} \;

# locate, veritabanı kullandığı için çok hızlıdır.
$ locate \*.mp3"
```

```
$ locate \*Qt*.so"

$ id
$ id user
$ groups

# logout yapmadan başka kullanıcıya geçmek.
# - Yeni kullanıcının bütün tanımlarını kullan.
#    Diğer bir deyişle, .bash_profile işlet.
$ su user
$ su - user
$ su - root
$ su -          # su - root ile aynıdır.

# -c: continue
# -m: mirrors a site
# -r: recursively
# -np: no parent, followlinks in the current directory.
$ wget -c http://slacks.net/slack.iso

# check integrity. 128 bit checksum.
$ md5sum *.iso > SUM
$ md5sum -c SUM

# md5sum yerine shaXsum'da kullanılabilir.
$ sha256sum /etc/passwd
```

9.59 History

Ctrl+r ile geçmiş komutlar üzerinde tersten arama yapılabilir. Bash kabuğunda geçmiş komutlar .bash_history dosyasına saklanır. C programlama içinden readline kütühanesi ile history desteği verilebilir.

```
$ history   # bütün geçmiş komutlar.

$ !!  # son komutu işlet.

$ !5   # 5 nolu komutu işlet.

# !cp  # cp ile başlayan son komutu işlet.
```

9.60 Aliases

Sık kullanılan komutlar için kısayol tanımlamasıdır. Kalıcı olması için bir dosyada veya /.bashrc'de tanımlanabilir. Bir dosyada tanımlanırsa, dosya aşağıdaki gibi çalışıtırılmalıdır.

```
$ . ./test_alias

$ alias

$ alias ls="ls -la"
$ alias cp="cp -i"

$ alias c="gcc -O2 -Wall -o test.c test"
$ alias c

$ unalias c
$ alias c
```

9.61 Date

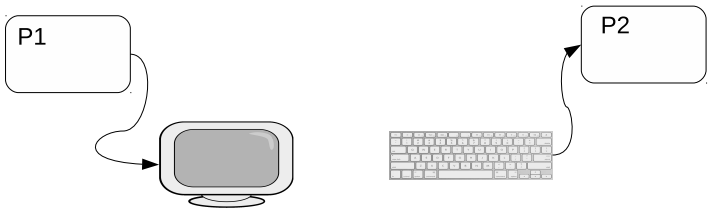
O anki zamanı döndürür. Locale değişkenine göre tarih/zaman bilgisini yazar.

```
$ date

# Şu anki zamanı Unix Epoch Time'dan itibaren saniye
# cinsinden göster.
# Unix Epoch Time : 1 Ocak 1970, gece yarısı
$ date +%s

# Dosya ismi üretmek için kullanılabilir.

F="backup.‘date +%F-%H-%M-%S‘.tgz"
tar zcvf /mnt/usb/$F workspace/project
```

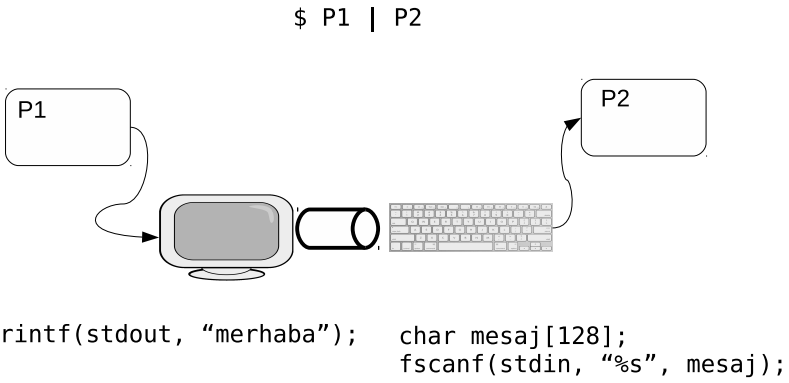
9.62 Unnamed Pipes

Bir prosesin çıkışını başka bir programın girişine bağlama işi pipe[17] ile yapılır. Dosya sistemindeki bir dosya kullanılmadığı için, bu tekniğe isimsiz pipe denir.

Linux komutları prensip olarak tek bir işi çok iyi yapmak üzere programlanmıştır. Bu komutlar i/o redirection ile etkileşimli bir biçimde kullanılarak, yani giriş ve çıkışlar $<, >, >>, |$ gibi ifadelerle yönlendirilerek son derece karmaşık işler yapılabilmektedir.



Şekil 9.23: stdout, terminale, stdin ise klavyeye bağlıdır.



Şekil 9.24: stdout, stdin'e bağlıdır.

[17]pipe, bildiğimiz boru demek. Ama Türkçesi kulağı çok tırmalıyor.

```
$ cat /etc/passwd | wc -l

$ ifconfig eth0 | grep HW
eth0      Link encap:Ethernet  HWaddr 00:90:f5:dc:51:d2

$ cat /etc/passwd | while read X
> do
> echo okunan satır $X
> done

okunan satır root:x:0:0:root:/root:/bin/bash
okunan satır daemon:x:1:1:daemon:/usr/sbin:/bin/sh
okunan satır bin:x:2:2:bin:/bin:/bin/sh
okunan satır sys:x:3:3:sys:/dev:/bin/sh
....

$ yes | command
$ yes no | command
$ yes ""| command  # otomatik yes girişi. boş enter gibi.
```

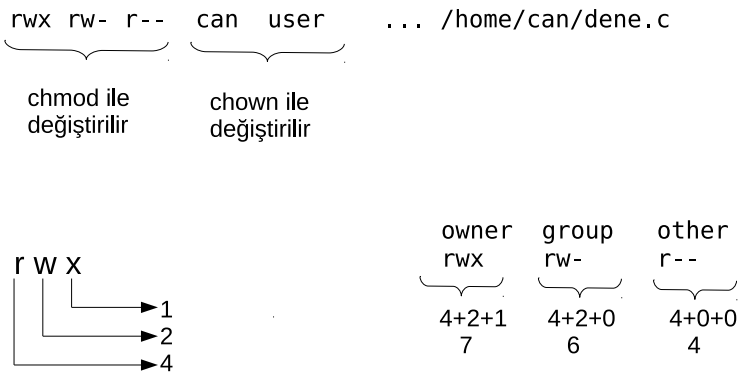
9.63 Sembolik linkler

Başka bir dosya veya dizini gösteren özel bir dosyadır. Gömülü sistemlerde kullanılan busybox sistemi sembolik linklerin genişçe kullanımına çok güzel bir örnektir.

Bir de karşıt olarak hard link kavramı vardır. Hard link kullanımı yaygın değildir. Hard link'de bir dosyanın farklı bir çok adı olabilir. Hangisinin esas ad olduğu belli değildir. Bundan dolayı yönetimi çok zordur.

```
$ ln -s mevcut_dosya sembolik_isim
$ ln -s mevcut_dosya

$ rm sembolik_isim  # Sadece sembolik ismi siler.

# -s kullanılmazsa hard link yaratılır.
# Hard link yaygın değildir, tavsiye edilmez.
```

9.64 Mod ve sahiplik



Şekil 9.25: Mod ve sahiplik.

9.65 Root kullanıcısı

root kullanıcısı her işi yapmaya yetkilidir. UID değeri her zaman 0'dır. Ya da 0 UID değerine sahip bütün kullanıcılar root yetkisine sahiptir.

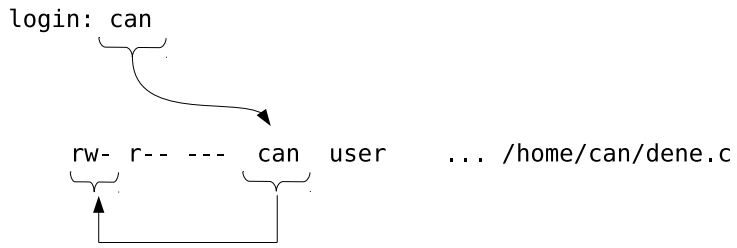
Gömülü sistemlerde genelde root ile çalışılırken, standard dağıtımlarda root kullanılmaz. Bunun yerine sudo kavramı getirilmiştir.

root şifresini vermeden bir kullanıcıya root yetkisi verilebilir. Bunun için ilgili kullanıcı /etc/sudoers dosyasına kaydedilir. Böylece bu kullanıcı root şifresini bilmeden sudo komutu ile root yetkisinde iş yapabilir. Gömülü sistemlerde genelde sudo kullanılmaz.

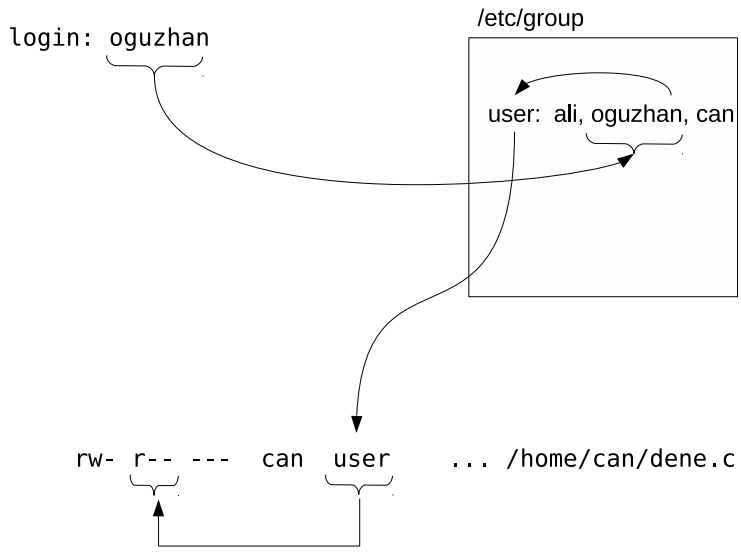
```
$ sudo mount /dev/sda3 /mnt/disk3
```

Standard bir kullanıcı, "su -" komutu ile root kullanıcısına logout olmadan geçebilir. Sudo yetkili kullanıcı ise sudo -s ile root'a geçebilir.

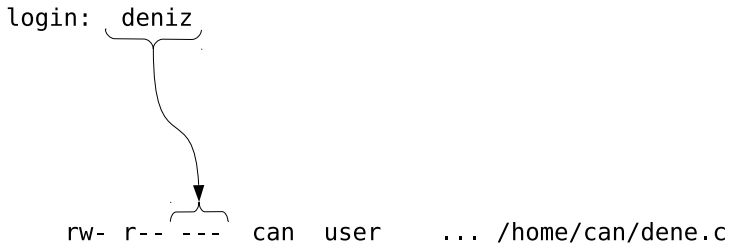
9.66 Login kavramı



Şekil 9.26: Login, Owner



Şekil 9.27: Login, Group



Şekil 9.28: Login, Other

9.67 Kılavuz Sayfaları

Kılavuz sayfları[18] kendi içinde gruplara ayrılmıştır.

doxgen kullanılarak, uygumala programları için de kolaylıkla man sayfaları yazılabilir.

1. Genel komutlar. pwd, ls, man
2. Sistem çağrıları(çekirdek fonksiyonları) open, read
3. Kütüphane fonksiyonları. fprintf, fopen
4. Özel dosyalar. zero, null, full
5. Dosya formatları. passwd, exports
6. Oyunlar ve ekran koruyucuları. noof, molecule
7. Diğerleri. fifo, iso_8859-9, environ
8. Sistem yönetim komutları ve servisleri. nfsd, mount, mke2fs

Dosyalar MANPATH ile gösterilen yerde aranırlar. Çok fazla kılavuz sayfası vardır. Az yer kaplasın diye gzip ile sıkıştırılmışlardır.

```
$ man  N command
$ man -k keyword
$ man -f  word
$ echo $MANPATH
```

[18] man pages. Kılavuz sayfalarını kullanalım, kullanmayanları uyaralım. Kılavuzu stackowerflow olanın burnu boktan çıkmazmış.

9.68 Geçici dosyalar

mktemp komutu ile tamamen keyfi isimli geçici dosya ve dizin üretilebilir. "-d" seçeneği ile geçici dizin de yaratılabilir.

İşimiz bitince dosya silinmelidir. C içinde yaratılan geçici dosyalar, proses bitince otomatik olarak silinirler.

Geçici dosyalar mevcut kullanıcı için rw- modunda yaratılırlar.

```
$ mktemp
$ mktemp test.XXXXXX
$ ls -l test.*
```

9.69 Proses kavramı

Bir programın çalışan örneğine proses denir. Linux'de herşey ya dosya ya da prosestir.

Her prosesin biricik pid ve ppid numarası vardır. init prosesinin pid numarası her zaman 1'dir ve çekirdek tarafından başlatılır.

```
$ pstree -p
```

Genelde anne ölünce çocuk da ölür. nohup komutu ile çocuk ölümleri engellenir.

```
$ nohup prog 1>out 2>err &
```

Kabuktan proses başlatma:

```
$ ls
$ sleep 20 &
```

Arkada bulunan (çalışan veya geçici olarak durdurulmuş olan) bütün prosesler için kabuk tarafından özel bir numara verilir. Bu numara sadece ve sadece mevcut kabuk için geçerlidir. Bu numaraya "iş numarası[19]" denir. Bu numara pid yerine kullanılabilir.

Her proses exit(N) ile son bulur. N değeri 0..255 arasında değişir. N değeri anne prosese döner. Hata yoksa genelde 0 kullanılır.

Proses son bulduğunda, bütün açık dosyalar otomatik olarak kapatılır. Eğer prosesin çocukları varsa, genelde 1 nolu prosese evlatlık verilir. Parent prosese CHLD sinyali gönderilir. Eğer önce anne ölür, sonra çocuk ölürse, çocuk exit değerini ve sinyalini gönderecek bir anne proses bulamaz. Bu tür proseslere zombi denir ve belirli bir süre sonra proses tablosundan otomatik olarak silinirler.

[19] job number

9.70 /proc dosya sistemi

Gerçek, fiziksel bir cihazla ilgili olmayan bir dosya sistemidir. Bundan dolayı /proc veya /sys, /debugfs gibi benzer dosya sistemlerine, pseudo veya "sözde" veya "sanki" dosya sistemi denir.

Bu dosya sistemi tamamen sanal olup, çekirdek tarafından, ya açılş ya da kullanım anında, bellekte kurulur. Bundan dolayı pek çok dosyanın yaratılma zamanı, hemen komutun girildiği zamandır.

Bu tür dosya sistemlerinin esas amacı, kernel space bilgilerine, user space tarafından kolayca erişim sağlamaktır. Ya da her iki alan arasında bilgi alışverişini sağlamaktır.

Sözde dosya sistemlerinin içeriğinde bir standardlaşma yoktur. Bir çekirdek sürümünde olan bir dizin veya dosya başka bir sürümde hiç olmayabilir ya da farklı bir biçimde bulunabilir. "$ man 5 proc" kılavuz sayfası içeriği genişçe açıklar.

/proc altındaki dosyalar C tarafından, sıradan bir dosya gibi fopen() ile açılarak içeriği kullanılabilir.

```
FILE *f;
f= fopen("/proc/cpuinfo", "r");
read & parse lines...
fclose(f);
```

Kabuk tarafında ise genelde cat komutu ile dosyalar okunur ve echo komutu ile güncellenir.

/proc dosya sisteminde her proses için bir adet dizin bulunur. Bu dizinin ismi pid numarası ile aynıdır.

Bu dosya sistemi genelde prosesler için yaratıldığından, dosya sistemi /proc dizinine mount edilir. "$$" ifadesi çalışan kabuğun pid numarasını verir. Çok kullanılan bazı /proc dosyaları aşağıda verilmiştir.

filesystems Çekirdekte o anda desteklenen dosya sistemlerini listeler. Modül olarak derlenmiş ama henüz yüklenmemiş destekler gösterilmez. Bunun için /proc/config.gz kullanılabilir.

uptime Ayakta kalınan ve boş geçen süreler raporlanır.

mounts Bağlı dosya sistemlerine ait bilgiler.

self O anda çalışan prosesin pid değeri. $$ ile karıştırma.

tty/driver Seri sürücülere ait bilgiler.

pid/cwd O anda çalışılan dizin.

pid/environ Çevre değişkenleri.

pid/exe argv[0] bilgisi.

fd Aktif fd'leri gösterir.

sys/net/ipv4/ip_forward 1 ise "paket yönlendirme" yapılır.

diğerleri sys/kernel, hostname, version, ...

```
$ cd /proc/$$/fd
$ ls -l
$ ls -l /proc/1/fd
$ cd /proc/‘pidof udevd‘/fd
$ ls -l
```

/proc dizinini kullanan prosesler lsof komutu ile tespit edilebilir.

```
$ lsof /proc
```

Gömülü Sistemlerde /proc dosya sistemi genelde /etc/rcS gibi bir betik tarafından açıkça mount edilir. Ya da /etc/fstab dosyasına giriş yapılarak mount edilir. /proc dosya sisteminin açıkça mount edilmesi örneği aşağıda verilmiştir.

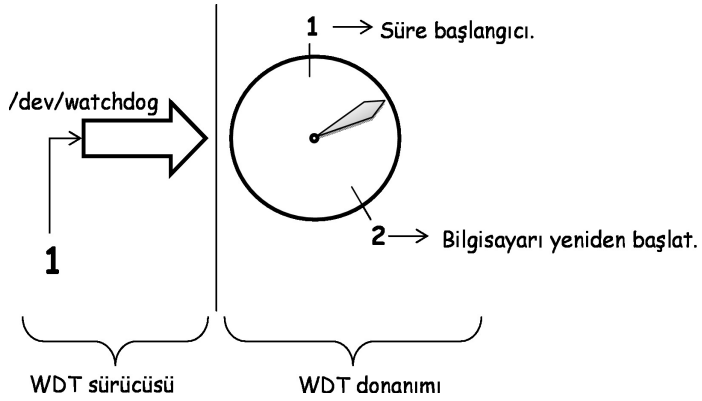
```
$ mount -t proc  proc      /proc  # veya
$ mount -t proc  none      /proc
$ mount -t ext2  /dev/sda1 /disk  # karşılaştırma için...

# /etc/fstab satırı.
proc /proc proc defaults 0 0

# Birinci 0 değeri dump içindir, kullanılmaz. Ikinci 0 değeri fsck içindir.
```

```
#       0: fsck yapma.
#       1: rootfs'tir. fsck yap.
#       2: En sona bu dosyayı fsck yap.
```

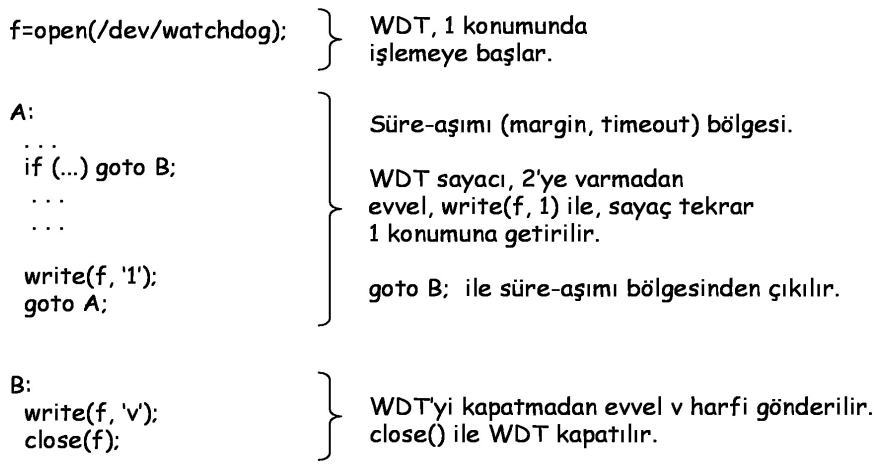
9.71 WDT

WDT[20], bazı bilgisayarda olan çok özel bir donanımdır. Bu donanım, belirli süre içinde dürtülmezse bütün sisteme "hard reset" atar. Özellikle, askeri, sağlık gibi uygulamalarda, askeri sistemlerde, yapay uydular gibi cihazın yanına gidilemeyecek uygulamalarda mutlaka kullanılmalıdır.

Projeler daha dizayn aşamasında WDT için gerekli özellikleri kendi içlerine katmalıdırlar. Proje bittikten sonra sisteme WDT eklemek çok risklidir. Diğer bir deyişle, yolda düzülen kervana WDT eklenmez.



Şekil 9.29: Watchdog timer.



Şekil 9.30: Uygulama içinde WDT kullanımı.

[20]Watchdog Timer, namı diğer bekçi iti :)

9.72 Login niye yohdir?

1. Çekirdek parametresi olan console tanımsız veya uygun değil
2. /etc/securetty'de tty tanımı yok.
3. busybox modu x değil veya root yetkisinde değil veya kütüphaneler eksik veya kütüphaneler uyumsuz.
4. init= parametresinin gösterdiği program mevcut değil, x modunda değil, ...
5. /init yok, /sbin/init yok, /linuxrc yok, ...
6. inittab'da seri konsol için getty tanımı yok.
7. stackowerflow'a bak!

9.73 Unbrick

BBB tuğla olursa aşağıdaki gibi geri getirilebilir[21]. Aşağıdaki dd komutu, SD kartın takılı olduğu cihazın ismi olmalıdır.

```
$ wget http://downloads.angstrom-distribution.org/demo/beaglebone/\
            testing/BBB-eMMC-flasher-v2013.12-2014.08.25.img.xz
$ unxz BBB-eMMC-flasher-v2013.12-2014.08.25.img.xz
$ dd if=BBB-eMMC-flasher-v2013.12-2014.08.25.img of=/dev/mmcblk0 bs=1M
```

BBB'yi kapat, 5V harici kaynağı tak, USB'den besleme yapma, SD kartı tak, S2 butonu basılı iken, 5V ver, ilk ışık yanıp/sönmeye başlayınca elini S2'den kaldır.

SD karttan Linux açılır ve otomatik olarak emmc.sh çalışır. ps komutu ile kontrol edilebilir. 1 dakikadan daha az bir sürede iş biter. Yanıp sönen 4'lü led grubu sabitlenir. ps ile bakınca emmc.sh gözükmez.

Daha sonra eMMC'den açmak için mutlaka fiziksel olarak power çekilmeli ve tekrar takılmalıdır. Çünkü BBB cihazı, üzerinde power olduğu sürece en son nereden açıldığını hatırlar. S2'ye basarak açılış yapıldığı için hep SD karttan açacaktır, taa ki power kesilene kadar.

Artık USB power ile de besleme yapılabilir. SD kart çıkarılır, power kablosu çekilir ve tekrar açılış yapılırsa, eMMC'deki uboot devreye girecektir.

9.74 free-electrons.com'dan vi tablosu

[21] Kaynak: http://www.crashcourse.ca/wiki/index.php/BBB_software_update_process

vi basic commands

Summary of most useful commands

Entering command mode

`[Esc]` Exit editing mode. Keyboard keys now interpreted as commands.

Moving the cursor

`h` (or left arrow key) move the cursor left.
`l` (or right arrow key) move the cursor right.
`j` (or down arrow key) move the cursor down.
`k` (or up arrow key) move the cursor up.
`[Ctrl] f` move the cursor one page **f**orward .
`[Ctrl] b` move the cursor one page **b**ackward.
`^` move cursor to the first non-white character in the current line.
`$` move the cursor to the end of the current line.
`G` go to the last line in the file.
`nG` go to line number *n*.
`[Ctrl] G` display the name of the current file and the cursor position in it.

Entering editing mode

`i` **i**nsert new text before the cursor.
`a` **a**ppend new text after the cursor.
`o` start to edit a new line after the current one.
`O` start to edit a new line before the current one.

Replacing characters, lines and words

`r` **r**eplace the current character (does not enter edit mode).
`s` enter edit mode and **s**ubstitute the current character by several ones.
`cw` enter edit mode and **c**hange the **w**ord after the cursor.
`C` enter edit mode and **c**hange the rest of the line after the cursor.

Copying and pasting

`yy` copy (yank) the current line to the copy/paste buffer.
`p` **p**aste the copy/paste buffer after the current line.
`P` **P**aste the copy/paste buffer before the current line.

Deleting characters, words and lines

All deleted characters, words and lines are copied to the copy/paste buffer.

`x` delete the character at the cursor location.
`dw` delete the current **w**ord.
`D` delete the remainder of the line after the cursor.
`dd` delete the current line.

Repeating commands

`.` repeat the last insertion, replacement or delete command.

Looking for strings

`/string` find the first occurrence of *string after the cursor.*
`?string` find the first occurrence of *string* before the cursor.
`n` find the **n**ext occurrence in the last search.

Replacing strings

Can also be done manually, searching and replacing once, and then using `n` (next occurrence) and `.` (repeat last edit).

`n,ps/str1/str2/g` between line numbers *n* and *p*, **s**ubstitute all (**g**: global) occurrences of *str1* by *str2*.
`1,$s/str1/str2/g` in the whole file ($: last line), **s**ubstitute all occurrences of *str1* by *str2*.

Applying a command several times - Examples

`5j` move the cursor 5 lines down.
`30dd` **d**elete 30 lines.
`4cw` **c**hange 4 **w**ords from the cursor.
`1G` **g**o to the first line in the file.

Misc

`[Ctrl] l` redraw the screen.
`J` join the current line with the next one

Exiting and saving

`ZZ` save current file and exit vi.
`:w` **w**rite (save) to the current file.
`:w file` **w**rite (save) to the *file* file.
`:q!` **q**uit vi without saving changes.

Going further

vi has much more flexibility and many more commands for power users!
It can make you extremely productive in editing and creating text.
Learn more by taking the quick tutorial: just type `vimtutor`.
Find many more resources on the net!

Free Electrons

Şekil 9.31: vi komutları

Dizin