Gömülü Linux Sistemleri

Login'e Kadar Linux

Basit Bir Gömülü Sistemin Kuruluşu

Giriş

Bu yazıda basit bir gömülü Linux sisteminin beagleboard'a kuruluşundan bahsedilecektir. Her ne kadar örnek sistem olarak beagleboard seçilmiş olsa da, burada anlatılan hemen hemen bütün konular diğer bordlara da çok az değişiklikle uygulanabilir. Prensipler değişmemektedir.

Burada bahsi geçen konular daha çok yeni başlayanlar içindir. Bu yazıda çok basit bir amacımız vardır. Bir cihaza güç veya elektrik verildikten sonra login gelene kadar arada bulunan aşamaları tek tek uygulamaktır. Konu hakkında bilgisi olanlar doğrudan özet kısmına göz atabilirler.

Gömülü sistemler prensip olarak bir işi çok iyi yapmak üzere kurulurlar. Gömülü sistemler işletim sistemine bağlı kalmadan da kurulabilirler. Çok geniş bir yelpazeye ve uygulama alanına sahip bu tür sistemlerle ilgilenmiyoruz. Bizler sadece Linux yüklenebilen cihazlarla ilgilenmekteyiz. Bundan dolayı "gömülü sistem" ifadesi doğrudan "gömülü Linux sistemi" anlamına gelmektedir. Ayrıca Linux dışındaki işletim sistemleri de ilgi alanımızın dışındadır.

1. Yazının Kaynağı

Gömülü sistemlerin prensip olarak bir işi çok iyi yapmak üzere tasarlandıklarından bahsetmiştik. Bizler trafikteki araçlar üzerine takılabilecek bir kara kutu sistemi üzerinde çalışmaktayız. Bu kara kutu sistemi ivme ölçer, cayro, manyetik alan ölçer gibi pek çok sensörden bilgi toplamakta ve aracın kinematik bilgilerini kaydetmektedir. Bu kısmın yazının konusu ile ilgisi yoktur. Fakat bu sensörler bir beaglebord makinesine bağlıdır ve burada koşan bir C programı gerekli işlemleri yapmaktadır.

Bu örnekten de görüleceği gibi gömülü sistemimiz sensörlerden bilgi toplamak gibi asli bir işe sahiptir. Bunun dışında tabi ki başka işler de yapmaktadır. Ama sistemin yapacağı esas iş, sensörlerle ilgilenmektir. İşte bu yazıda anlatılacak gömülü sistem, bu proje için geliştirilen ve halen kullanılmakta olan bir Linux sistemidir ve son derece genel bir yapıya sahiptir.

2. Beagleboard

Bu yazıda Beagle Board Rev C3 sistemi üzerinde uygulama yapılacaktır. Bu makine üzerinde malesef eternet kartı mevcut değildir. Bundan dolayı dosya alış verişi MMC kart yardımı ile yapılacaktır. Bu yazıda MMC (MultiMediaCard) ile SD(SecureDigital) aynı anlamda kullanılacaktır.

Bordun, seri port üzerinden Linux yüklü bir masaüstü makineye bağlı olduğunu kabul edilmektedir. Eğer masaüstü makinede seri port yoksa seri/usb çevirici ile de aynı bağlantı yapılabilir.

Masaüstü tarafında seri kanal iletişim programı olarak minicom kullanılmaktadır. Okuyucu kendi gözde terminal programını da kullanabilir. Seri kanal, 115200, 8N1 ve NoFlow olarak ayarlanmalıdır. Masaüstü olarak bahsedilen makine gerçek bir masaüstü PC veya bir notebook veya üzerinde Linux koşan herhangi bir makine olabilir.

3. Linux Dağıtımları Hakkında

Bizler masaüstünde Slackware Linux kullanmaktayız. Fakat bu yazıda bahsi geçen hiç bir konu dağıtıma bağlı olmayacaktır. Bütün paketler el ile derlenecek ve herhangi bir paket yöneticisi kullanılmayacaktır. Okuyucunun da paket yöneticisi kullanmaması, bütün paketleri el ile derlemesi tavsiye edilir. Ayrıca, eğer açıksa, “otomatik mount” özelliğinin kapatılması tavsiye edilir.

4. Çapraz Derleme

Örnek bordumuz ARM işlemci kullanmaktadır. ARM işlemcili makineler, cep telefonları ve tablet bilgisayarlar gibi cihazlar sayesinde inanılmaz derecede yaygınlaşmıştır. Buna karşın masaüstü makinelerimiz halen x86 işlemcileri kullanmaktadırlar. Doğal olarak masaüstünde bulunan derleyiciler x86 kodu üretmektedir. x86 işlemci için x86 kodu üreten derleyiciye native derleyici denir. Genellersek kendi üzerinde çalıştığı mimari için kod üreten derleyiciye native derleyici denir.

Doğal olarak x86 makinede bulunan native derleyici, ARM için kod üretemez. Kendi çalıştığı mimari için değil de başka bir mimari için kod üreten derleyiciye çapraz derleyici (cross compiler) denir. Buna göre, eğer bizler x86 makine üzerine uygun çapraz derleyiciyi yüklersek, ARM makinede çalışacak kodu, x86 makinede üretebiliriz. Çapraz derleyicilerin ürettiği koda hedef mimari kodu denir.

Yazılım geliştirme süreci sadece derleyicilere bağımlı değildir. Bir kodun derlenerek obje haline getirilmesi, objelerin linker ile bağlanması ve çalışabilir kodun elde edilmesi, kodun çalıştırılarak debug edilmesi gibi aşamalar yazılım geliştirilirken kullanılır. Dikkat edilirse burada elde edilen çıkış bir başka programın girişi olmaktadır.

Örneğin, derleyici object kod elde eder. Elde edilen object kod linker ile birleştirilir. Linker ile birleştirilen kod, debug ile incelenebilir. Birinin çıktısı diğerinin girdisi olan bu tür yazılım sistemlerine araç zinciri (tool chain) denir. Çünkü sistemin genel mantığı zincir gibidir.

x86 makinesinde, ARM kodu elde edebilmek için en az 1 adet tool chain'e gerek vardır. ARM işlemcili makineler için birden fazla tool chain vadır. Bizler beagleboard'u satanların tavsiye etmiş olduğu ve CodeSourcery tarafından desteklenen toolchain'i kullanacağız. Bu tool chain, artık mentor graphics tarafından desteklenmektedir.

http://www.mentor.com/embedded-software/codesourcery

adresinden "Download Lite Edition" butonu ile ilgili sayfaya geçilir. "ARM processors" başlığı altında bulunan "Download GNU/Linux Release" ile gösterilen adrese girilir. Burada e-posta adresi ile kayıt yapılır. Kayıt yapılan e-postaya, indirme için gerekli olan adres gönderilir. Bu adrese tıklanır ve gelen sayfadan "Advanced Packages" altında bulunan "IA32 GNU/Linux TAR" bağlantısındaki paket indirilir. Eskiden tek tıkla yapılırdı bu iş.

Bu yazı yazılırken en son sürüm **2012.03-57** idi. Paketin tam adı ise, **arm-2012.03-57-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2** ile verilmiştir.

Bu toll chain sadece 32 bit makineler için destek vermektedir. 64 bit makineler için 32 bitlik kütüphanelerin kurulması gerekir.

Tool chain aşağıdaki gibi kurulur. Paket yöneticisi kullanılması tavsiye edilmez ki ne yapıldığını görelim. Bizler bu tür paketleri /cross dizini altına kurmaktayız. Okuyucu kendisine uygun herhangi bir dizin seçebilir.

```
$ cd /cross
$ tar jxvf arm-2012.03-57-arm-none-linux-gnueabi-i686-pc-linux
$ ln -s arm-2012.03 arm-none-linux-gnueabi
$ ls arm-none-linux-gnueabi/bin

  arm-none-linux-gnueabi-addr2line
  arm-none-linux-gnueabi-ar
  arm-none-linux-gnueabi-as
  arm-none-linux-gnueabi-c++
```

```
arm-none-linux-gnueabi-c++filt
arm-none-linux-gnueabi-cpp
arm-none-linux-gnueabi-elfedit
arm-none-linux-gnueabi-g++
arm-none-linux-gnueabi-gcc
arm-none-linux-gnueabi-gcc-4.6.3
arm-none-linux-gnueabi-gcov
arm-none-linux-gnueabi-gdb
arm-none-linux-gnueabi-gdbtui
arm-none-linux-gnueabi-gprof
arm-none-linux-gnueabi-ld
arm-none-linux-gnueabi-nm
arm-none-linux-gnueabi-objcopy
arm-none-linux-gnueabi-objdump
arm-none-linux-gnueabi-ranlib
arm-none-linux-gnueabi-readelf
arm-none-linux-gnueabi-size
arm-none-linux-gnueabi-sprite
arm-none-linux-gnueabi-strings
arm-none-linux-gnueabi-strip
```

Tool chain tar ile açıldığında bütün dosyalar *arm-2012.03/* isimli bir dizin içinde kurulur. Tahmin edileceği gibi bu dizin, aslında sürüm numarasıdır. Bu tür bir dizini kullanmak pek pratik olmayacağı için 3. adımda dizin ismi *arm-none-linux-gnueabi* ile sembolik olarak bağlanmıştır. Bu adımdan sonra bütün betiklerde /cross/arm-none-linux-gnueabi ismi kullanılacaktır. Yeni bir sürüm yüklendiğinde sadece sembolik linki değiştirmek yeterli olacaktır. Betiklerdeki isimleri güncellemeye gerek kalmayacaktır.

4. adımda tool chain'e ait bütün programlar listelenmiştir. Bütün programlar bin/ dizini altındadır. Tool chain'i kullanıma hazır hale getirmek için PATH değişkenine

/cross/arm-none-linux-gnueabi/bin girilmesi yeterlidir. Örneğin:

```
export PATH=$PATH:/cross/arm-none-linux-gnueabi/bin
```

Artık tool cahin kullanıma hazırdır.

Özetlemek gerekirse paketi açıp, paket içindeki bin/ dizinini PATH değişkenine eklemek çapraz derleyici veya tool chain'i kullanmak için yeterlidir.

Tool chain'in sürümü aşağıdaki gibi elde edilebilir.

```
$ arm-none-linux-gnueabi-gcc --version
```

```
arm-none-linux-gnueabi-gcc (Sourcery CodeBench Lite 2012.03-57
Copyright (C) 2011 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTIC
```

Çapraz derleyiciyi test etmek için aşağıdaki gibi basit bir C programı yazılabilir. Bu program daha sonra bord üzerinde çalıştırılacaktır.

```
/* ilk.c */

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]){

   fprintf(stdout, "ARM işlemci için ilk program.\n");
   fprintf(stdout, "Merhaba ARM.\n");

   return EXIT_SUCCESS;
}
```

Örnek programın ismi ilk.c olsun. Bu programı x86 için aşağıdaki gibi derleyebiliriz.

```
$ gcc -O2 -Wall -Werror -o ilk.dynamic ilk.c
```

-O2 ile optimizasyon seviyesi verilir. -Wall ile bütün ikazların verilmesi sağlanır. -Werror ile ikazların error gibi algılanması sağlanır. Bir C programında hiç ikaz bulunmamalıdır. Prensip olarak ikaz alınan bir C programı kullanılmamalıdır. Mümkünse bütün ikazlar temizlenmelidir. -o ilk.dynamic ile elde edilecek yürütülebilir dosyasının ismi verilir. Burada derleme dinamik olarak yapılmaktadır. Daha sonra statik derleme yapılacaktır. En sonda bulunan ilk.c ile de derlenecek dosyanın adı verilmektedir. Bu derleme sonunda elde edilecek çalışabilir kod, native bir koddur. Sadece x86 makinede işler.

Çapraz derleme yapmak ve hedef mimari için kod üretmek için yukarıdaki gcc ifadesinin hemen baş tarafına **arm-none-linux-gnueabi-** ön ekini getirmek yeterli olacaktır. Buna göre ilk.c programı aşağıdaki gibi çapraz olarak derlenebilir.

```
$ arm-none-linux-gnueabi-gcc -O2 -Wall -Werror -o ilk.dynamic
```

arm-none-linux-gnueabi- ifadesine çapraz derleyici ön-eki (cross compiler prefix)

denir. Dikkat edilirse bin/ dizini altında bulunan bütün toolchain programları bu ön ek ile başlar. arm-none-linux-gnueabi-gcc komutu girildiğinde, PATH değişkeninden dolayı, ilgili komut /cross/arm-none-linux-gnueabi/bin altında bulunacak ve çapraz derleme yapılacaktır.

Ön ek kullanımı çapraz derleme işini çok basitleştirmektedir. Örneğin ppc işlemcileri için ppc- ve mips işlemcileri için mips- ön ekli çapraz derleyiciler bulunmaktadır.

Derleme sırasında -static seçeneği kullanılırsa, hedef kod statik olarak üretilir. Diğer bir deyişle gerekli bütün kütüphaneler kodun altına eklenir. Bundan dolayı da ilgili kod aşırı derecede büyür. Statik derleme örneği ve elde edilen kod boylarının karşılaştırılması aşağıda verilmiştir.

```
$ arm-none-linux-gnueabi-gcc -O2 -Wall -Werror -static -o ilk.

$ ls -l ilk.dynamic ilk.static

 -rwxr-xr-x 1 nazim users   5821 Aug 23 11:36 ilk.dynamic
 -rwxr-xr-x 1 nazim users 677923 Aug 23 11:36 ilk.static
```

Çıkıştan da görüleceği gibi, dinamik kodun boyu çok kısadır. Fakat dinamik kodu çalıştırmak için gerekli kütüphaneler sistemde bulunmalıdır. Statik üretilmiş kod için hiç bir kütüphaneye gerek yoktur. Her durumda çalışır.

file komutu çok önemli bir Linux komutudur ve bize dosya hakkında bazı özet bilgiler verir. Aşağıda statik ve dinamik kodların bilgileri file ile elde edilmiştir. Kolay okunması için çıkışa satırlar eklenmiştir.

```
$ file ilk.static
 ilk.static: ELF 32-bit LSB executable,
             ARM, version 1 (SYSV),
             statically linked,
             not stripped

$ file ilk.dynamic
 ilk.dynamic: ELF 32-bit LSB executable,
              ARM, version 1 (SYSV),
              dynamically linked (uses shared libs),
              not stripped
```

Çıkıştan da görüleceği gibi “ARM, version 1″ ifadesi bize ARM mimarisi için kod üretildiğini gösterir. Ayrıca kodun statik ve dinamik olarak derlendiğini raporlar. Her iki bilginin son ifadesinde “not stripped” bilgisi bulunur.

strip kelimesi “soymak” demektir. Yani kodlar *soyulmamıştır*. Kod içinde çeşitli debug bilgileri, isimler, tablolar vs bulunur. Bu bilgiler kodu çok büyütür. İstersek objcopy veya strip komutu ile bu bilgiler atılabilir, soyulabilir. Aşağıdaki örnekte hem statik hem dinamik çıkış soyulmuş ve dosya boyları listelenmiştir. Boylardaki bir miktar küçülme açıkça görülmektedir.

```
$ arm-none-linux-gnueabi-objcopy --strip-all ilk.dynamic

$ arm-none-linux-gnueabi-objcopy --strip-all ilk.static

$ ls -l ilk.dynamic ilk.static
 -rwxr-xr-x 1 nazim users   3252 Aug 23 11:36 ilk.dynamic
 -rwxr-xr-x 1 nazim users 548692 Aug 23 11:36 ilk.static
```

Dikkat edilirse, aynen derleyici kullanır gibi, objcopy komutunun önüne yine derleyici ön eki getirilmiştir. Eğer sadece $ objcopy –strip-all ilk.dynamic gibi bir komut girecek olsaydık, x86 makineye ait objcopy komutu çalışacaktı. Fakat biz tool chain içinde bulunan ve hedef makine için çalışacak olan objcopy komutunu kullanmak istiyoruz. Bundan dolayı objcopy komutunun önüne arm-none-linux-gnueabi- ön eki getirilmiştir.

ldd komutu bir komutun bağlı olduğu kütüphanelerin listesini verir. Örneğin x86 makinede ls komutunun bağımlı olduğu kütüphaneler aşağıdaki gibi listelenebilir.

```
$ ldd /usr/bin/ls
        linux-gate.so.1 =>  (0xffffe000)
        librt.so.1 => /lib/librt.so.1 (0xb770a000)
        libcap.so.2 => /lib/libcap.so.2 (0xb7706000)
        libacl.so.1 => /lib/libacl.so.1 (0xb76fe000)
        libc.so.6 => /lib/libc.so.6 (0xb759b000)
        libpthread.so.0 => /lib/libpthread.so.0 (0xb7581000)
        /lib/ld-linux.so.2 (0xb773c000)
        libattr.so.1 => /lib/libattr.so.1 (0xb757c000)
```

Çıkıştan da görüleceği gibi ls komutunun bağlı olduğu bütün kütüphaneler listelenmiştir. Bu komutun işleyebilmesi için bütün kütüphanelerin yüklü olması gerekir. Aynı mantık gömülü sistemde çalışan ve dinamik olarak derlenen komutlarda da geçerlidir. Fakat tool chain içinde arm-none-linux-gnueabi-ldd gibi bir komut mevcut değildir.

objdump komutu ldd’nin işini hemen hemen gerçekler. Aşağıda ARM için derlediğimiz kodun bağımlı olduğu kütüphanelerin listesi elde edilmiştir.

```
$ arm-none-linux-gnueabi-objdump -x ilk.dynamic | grep NEEDED
```

```
NEEDED               libgcc_s.so.1
NEEDED               libc.so.6
```

Görüldüğü gibi, ilk.dynamic programı libgcc_s ve libc kütüphanelerine ihtiyaç duyar. Gömülü sistem kurulurken bu kütüphaneler bir yerlerden bulunmalı ve /lib veya /usr/lib altına kopyalanmalıdır.

objdump programının en kötü tarafı özyineli bağımlılıkları göstermemesidir. Bir kütüphanenin başka bir kütüphaneye bağımlılığını bulmak için objdump komutu her bir kütüphane için tek tek işletilmelidir. ldd komutuysa özeyineli olarak bütün bağımlılıkları inceler.

ilk.c programını -static seçeneği ile de derlemişti. Statik derlenmiş kodların kütüphaneye ihtiyaçları yoktur. Çünkü kütüphaneleri kendi kodlarının altına açıkça eklemişlerdir. objdump ile statik kodun kütüphaneye ihtiyacı olmadığı aşağıdaki gibi incelenebilir.

```
$ arm-none-linux-gnueabi-objdump -x ilk.static | grep NEEDED
```

Hiç bir “NEEDED” satırı yoktur. Çünkü statik derlenmiştir.

5. Boot Yükleyicilerinin Derlenmesi: MLO ve U-Boot

Hangi bord kullanılırsa kullanılsın, boot yani ilk yükleme mekanizması genelde değişmez. Bordların içinde, genelde ROM’larına gömülmüş çok basit bir yükleyici bulunur. Bu yükleyici doğrudan donanıma bağlı olup çok az yeteneğe sahiptir. Bu tür yükleyiciler genelde daha yetenekli bir yükleyiciyi yüklemekle görevlidirler.

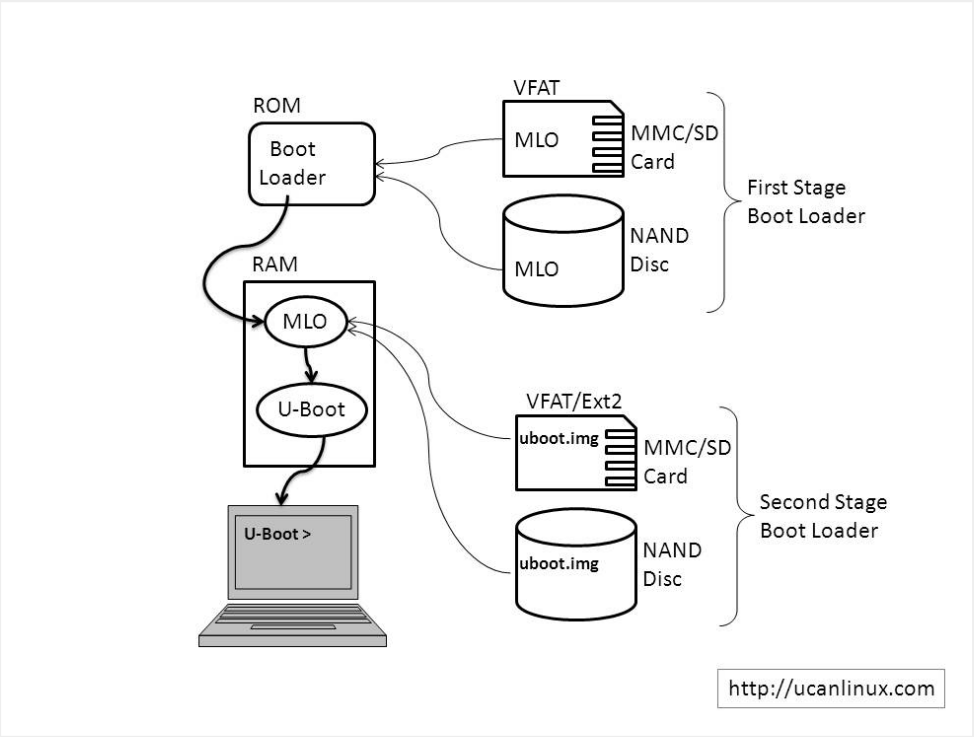
Bordların ROM’unda yüklü olan bu tür yükleyicilere Rom Boot Loader denir. Rom Boot Loader programının yüklemiş olduğu daha yetenekli yükleyiciye birici aşama yükleyici (first stage boot loader) denir. Pek çok birinci aşama yükleyicisi vardır. Beagle board için kullanılan birinci aşama yükleyicisine MLO denir.

Buna göre, Rom boot yükleyicisi MLO programını yükler. MLO programı da aslında fazla yetenekli bir program değildir. MLO Rom boot yükleyicisinden daha yeteneklidir. VFAT üzerinden yükleme yapabilir, aynı zamanda seri veya usb üzerinden de yükleme yapabilir. MLO programı donanıma daha yakındır, çok genel değildir.

Nihayetinde MLO programı daha yetenekli bir boot yükleyicisine gerek duyar. MLO’nun yüklediği ve fazla yeteneği olan boot yükleyicisine ikinci aşama boot yükleyicisi (second stage boot loader) denir. Pek çok ikinci aşama boot yükleyicisi mevcuttur. ARM sistemleri için ikinci aşama boot yükleyicisi olarak u-boot yaygın

biçimde kullanılır.

Özetlersek, Rom boot yükleyicisi, MLO isimli birinci aşama boot yükleyicisini, MLO ise u-boot isimli ikinci aşama boot yükleyicisini yükler. u-boot son derece gelişmiş bir yükleyicidir. U-boot aynı zamanda bir monitör programıdır, kullanıcı ile etkileşimli çalışır ve ağ desteği mevcuttur.



— Şekil 1.1 : Birinci ve ikinci boot yükleyicileri.

Şu ana kadar anlatılanlar Şekil 1.1'de özetlenmiştir. ROM boot yükleyicisi VFAT ile formatlanmış bir dosya sisteminden veya NAND diskten doğrudan MLO kodunu yükleyebilir. RAM'a yüklenen MLO ise, VFAT veya EXT2 dosya sistemlerinden u-boot sistemini yükleyebillir. Aynı zamanda NAND diskten de u-boot yükleyebilir. Şekilde gösterilmemesine rağmen, MLO sistemi, seri kanal ve usb üzerinden de u-boot kodunu yükleyebilir.

Bazı yazılarda, Rom boot yükleyicisi için birinci aşama, MLO için 2. aşama ve u-boot için 3. aşama yükleyici denir. Bu tamamen sisteme nereden bakıldığı ile ilgilidir. Donanımcılar genelde Rom boot yükleyicisini 1. aşama olarak adlandırılırlar. Ama biz yazılımcıyız.

Şimdi MLO ve u-boot programları, elimizdeki çapraz derleyici ile kurulacak sonra borda aktarılacaktır.MLO programı artık u-boot içinde gelmektedir. Bundan dolayı sadece u-boot'u derlemek yeterli olacaktır.

U-boot programın en son sürümü, http://denx.de adresinden aşağıdaki gibi indirilir. Bu yazı yazılırken son sürüm 2012.07 idi.

```
$ wget ftp://ftp.denx.de/pub/u-boot/u-boot-2012.07.tar.bz2
$ tar jxvf u-boot-2012.07.tar.bz2
$ cd u-boot-2012.07
```

Paketin içinde bulunan **include/configs/omap3_beagle.h** dosyası çok önemlidir. Mutlaka incelenmelidir. Bizler bu dosyada sadece CONFIG_SYS_PROMPT değerini değiştirdik. Böylece prompt olarak UcanLinux lafı gelecektir 

u-boot sistemi aşağıdaki gibi çapraz olarak derlenir.

```
1 $ make CROSS_COMPILE=arm-none-linux-gnueabi- omap3_beagle_co
  Configuring for omap3_beagle board...

2 $ make CROSS_COMPILE=arm-none-linux-gnueabi-

3 $ ls -l MLO u-boot.img tools/mkimage
  -rw-r--r-- 1 nazim users  45780 Aug 23 19:19 MLO
  -rwxr-xr-x 1 nazim users  56344 Aug 23 19:18 tools/mkimage
  -rw-r--r-- 1 nazim users 335588 Aug 23 19:19 u-boot.img

4 # cp tools/mkimage /usr/local/bin
5 $ file u-boot.img

  u-boot.img: u-boot legacy uImage,
  U-Boot 2012.07 for beagle board,
  Firmware/ARM,
  Firmware Image (Not compressed),
  335512 bytes,
  Fri Aug 24 09:16:02 2012,
  Load Address: 0x80100000,
  Entry Point: 0x00000000,
  Header CRC: 0x6ABF9E12,
  Data CRC: 0x8ADBC852
```

1. ve 2. adımda CROSS_COMPILE değişkeni ile çapraz derleyicinin ön eki verilir. Paketleri derleyen Makefile sistemi içinde genelde

```
${CROSS_COMPILE}gcc  -O2 -Wall ...
```

gibi ifadeler bulunur. make komutunda verilen CROSS_COMPILE ön eki, make

sistemi tarafından yerine konur ve yukarıdaki ifade

```
arm-none-linux-gnueabi-gcc -O2 -Wall ...
```

şekline gelir. Böylece u-boot sistemi her türlü çapraz derleyici ile derlenebilir. Eğer çapraz derleme değil de native derleme yapılacaksa CROSS_COMPILE değişkeni açıkça belirtilmez. Bu durumda ${CROSS_COMPILE} değeri yerine hiç bir ifade konulmaz ve sadece gcc değeri kalır. Bu durumda native derleme yapılır.

Yukarıdaki 1. adımda bord için uygun config dosyası seçilir. Başka bir işlem yapılmaz.

2. adımda u-boot sistemi derlenir. Benim makinem çok eski olduğu için derleme uzun sürdü. Hızlı makinelerde derleme süresi kısa olacaktır.

Derleme başarı ile bittiğinde pek çok program elde edilir. Şimdilik bizlere MLO, u-boot.img ve tools/mkimage programları gereklidir.

3. adımda bu programların listesi elde edilmiştir. Birinci aşama boot yükleyicisi MLO ve ikinci aşama boot yükleyicisi u-boot.img programlarının boylarına dikkat edilmelidir. MLO daha ufaktır. u-boot.img ise çok daha büyüktür. Bu boylara bakılarak, bu programların yüklenebileceği en ufak disk miktarı tahmin edilebilir.

Son olarak tools dizini altında bulunan mkimage isimli bir program üretilir. Bu programın kullanımı ile ilgili örnekler, yazının ilerleyen bölümlerinde verilecektir. mkimage programı çok kullanılan bir programdır ve 4. adımda verildiği gibi, /usr/local/bin gibi daha genel bir yere kopyalanması tavsiye edilir.

Son olarak 5. adımda u-boot.img programı, file komutu ile incelenmiştir.U-boot derlendiğinde, esas olarak u-boot.bin programı elde edilir. Derleme sırasında u-boot.bin programı mkimage ile u-boot imajı haline getirilir. Diğer bir deyişle, u-boot.bin programının en üstüne 64 baytlık çok özel bir başlık eklenir. Bu başlık ekleme işini mkimage programı yapar. mkimage'nin eklemiş olduğu başlık, yazının ilerleyen kısmında ayrıca incelenecektir.

Şimdi Şekil 1.1'de temsil edilen açılış gerçeklenecektir. Diğer bir deyişle elimizde bulunan MLO ve u-boot.img programları VFAT formatlı ve MMC karta yüklenecek ve bord u-boot seviyesine kadar açılacaktır. Gömülü sistemlerle çalışırken u-boot promptuna kadar gelebilmek önemli bir aşamadır. Çünkü u-boot sistemi son derece yeteneklidir ve kullanıcıya pek çok seçenek sunar. U-boot gelmeden önce oluşacak hatalar genelde çok vakit kaybettirirler. Çünkü bu adımlarda hata mesajları ya hiç yoktur ya çok yetersizdir.

6. MMC Yardımı ile Boot Yükleyicilerinin Borda Taşınması ve Açılış

Elimizde bulunan MLO ve u-boot.img programları aşağıdaki gibi borda taşınır ve bord u-boot seviyesinde açılır. Öncelikle masaüstü makinede, eğer varsa, automount özelliği kapatılmalıdır. Her işi el yordamı ile yapmak bizce çok önemlidir.

Masaüstü makinede MMC/SD kart okuyucunun mevcut olduğu kabul edilmektedir. MMC kart okuyucuya takıldığında, işletim sistemi tarafından karta bir cihaz ismi atanır. Bu cihaz ismi birkaç yolla tespit edilebilir.

Takılan cihazın ismi aşağıdaki gibi fdsik veya dmesg komutu ile elde edilebilir. fdisk programı diskleri bölümlendirmek için kullanılır. dmesg programı ise çekirdek mesajlarını listeler.

```
# fdisk -l
 ...
 Device Boot      Start         End      Blocks   Id  System
 /dev/sdb1   *           51       60799       30374+   4  FAT16

# dmesg|tail

USB Mass Storage support registered.
 ...
 sd 6:0:0:0: [sdb] Assuming drive cache: write through
 sd 6:0:0:0: [sdb] No Caching mode page present
 sd 6:0:0:0: [sdb] Assuming drive cache: write through
 sdb: sdb1
```

Yukarıdaki komut çıkışlarından da görüleceği gibi, takılan MMC kartın simi sdb'dir. Ayrıca kartta tek bir bölüm mevcuttur. Bizler bu bölümü dosyaları taşımak için kullanacağız.

Eğer kart daha önce formatlanmamış ise fdisk /dev/sdb ile bölümlendirme yapılır ve mkfs.vfat /dev/sdb1 ile de dosya sistemi kurulabilir.

MLO ve u-boot.img programları aşağıdaki gibi karta kopyalanır. Önce MLO'nun kopyalanmasına dikkat edilmelidir. Asla unmount işlemi yapılmadan kart yuvasından çekilmemelidir.

```
# mkdir /mnt/mmc
# mount /dev/sdb1  /mnt/mmc
# cp    MLO        /mnt/mmc
# cp    u-boot.img /mnt/mmc
```

```
# ls -l /mnt/mmc
 -rwxr-xr-x 1 root root  45780 Aug 23 19:35 MLO
 -rwxr-xr-x 1 root root 335588 Aug 23 19:49 u-boot.img

# umount /mnt/mmc
```

Bordda şu an için elektrik yoktur. MMC kartı boarda takılır. Bordun seri kablo ile masaüstüne bağlanmış olduğu kabul edilmektedir. Masaüstü tarafında,

```
# minicom -s
```

komutu ile seri terminal programı setup modunda başlatılır. “-s” seçeneği setup anlamındadır. Bir kereliğine mahsus seri kanal ayarları yapılmalıdır. “Serial port setup” menüsünden girilerek seri kanal ayarları 8N1, 115200, NoFlowControl şeklinde yapılır. Seri kanal ismi genelde /dev/ttyUSB0 veya /dev/ttyS0 şeklindedir.

“Save setup as” veya “Save setup as dfl” ile yapılan ayarların kalıcı olması sağlanır. Daha sonra “exit” ile çıkılarak ana ekrana düşülür.

Bir parmak USER butonunda basılı iken Bord’a güç verilir. USER butonuna basılması, açılışın MMC’den olmasını garantiler.

Ya da bord açıkken, bir parmak USER butonuna basılı iken diğer parmakla RESET butonuna basılır. Bu durumda da açılış yine MMC’den yapılır. Açılış aşağıda verilmiştir.

```
40V
 U-Boot SPL 2012.07 (Aug 23 2012 - 19:18:22)
 Texas Instruments Revision detection unimplemented
 OMAP SD/MMC: 0
 reading u-boot.img
 reading u-boot.img

U-Boot 2012.07 (Aug 24 2012 - 09:15:25)

 OMAP3530-GP ES3.0, CPU-OPP2, L3-165MHz, Max CPU Clock 600 mHz
 OMAP3 Beagle board + LPDDR/NAND
 I2C:   ready
 DRAM:  256 MiB
 NAND:  256 MiB
 MMC:   OMAP SD/MMC: 0
 In:    serial
 Out:   serial
 Err:   serial
```

```
Beagle Rev C1/C2/C3
timed out in wait_for_pin: I2C_STAT=0
No EEPROM on expansion board
Die ID #763400030000000000403230911101a005
Net:   Net Initialization Skipped
No ethernet found.

Hit any key to stop autoboot:  0

UcanLinux >
```

Bord nasıl açılmıştır? USER butonuna basılı tuttulduğu için ROM boot yükleyicisi MMC/SD kartta MLO programını bulmuş ve belleğe yüklemiş ve yütütmeyi MLO'ya vermiştir. MLO programı da MMC'de bulunan u-boot.img programını belleğe yüklemiş ve yürütmeyi u-boot.img programına aktarmıştır. Şekil 1.1'de verilen durum aynen gerçekleşmiş ve u-boot promptu UcanLinux> şeklinde ekrana gelmiştir.

Çıkıştan da görüleceği gibi, u-boot sürümü 2012-07 olarak gözükmektedir. Aynı zamanda prompt ifadesi de değişmiştir. Sonuçta bizim derlediğimiz MLO ve u-boot sistemi çalışmaktadır.

Henüz elde edilen bilgiler kalıcı biçimde NAND diske kazınmamıştır. İlerleyen bölümlerde MLO ve u-boot diske kazınacak ve MMC kart ile bir işimiz kalmayacaktır.

Tekrar etmekte fayda görüyoruz. Gömülü sistemlerle uğraşırken u-boot promptunu elde etmek önemli bir aşamadır. Bu adımdan sonrası nispeten daha kolaydır. Çünkü yapılacak işlerin borda bağımlılığı gittikçe azalmaktadır.

U-boot promptu geldikten sonra help ile mevcut komutların listesi elde edilebilir. Ayrıca printenv komutu ile u-boot'un ön tanımlı çevre değişkenleri elde edilebilir. Bizler bu yazıda u-boot komutları ile ilgilenmeyeceğiz. Başka bir yazının konusu olacaktır. Fakat bu değişkenlerin 3 tanesi çok önemlidir ve okuyucu bunları ayrıca incelemelidir. Bu değişkenler sıra ile, bootargs, mtdparts ve bootmcd'dir.

bootargs değişkeni içinde çekirdeğe aktarılacak parametreler mevcttur. U-boot bu değişken ile hiç ilgilenmez. U-boot çekirdeği yükleyeceği zaman buradaki bütün ifadeyi gözü kapalı bir biçimde çekirdeğe, "çekirdek parametresi" olarak aktarır.

mtdparts değişkeninde ise, flash diskin bölümlendirme bilgisi bulunur. MS-DOS bölümlendirmelerinde çekirdek, MBR'yi okuyarak disk bölümleri hakkında bilgi sahibi olur. Fakat flash disklerde bölümlendirme için ayrı bir alan yoktur. Çekirdek bölümlendirme hakkındaki bilgileri mtdparts değişkeninden elde eder. U-boot programı, çekirdeği yüklerken mtdparts değişkenini bootargs içine gömerek çekirdeğe gönderir.

Çekirdek açılırken “Kernel Command Line” bilgisi içinde, kendisine gelen bütün parametreleri raporlar. bootargs komutu gibi, u-boot programı mtdparts ile hiç ilgilenmez. Gözü kapalı bir biçimde bu değer çekirdeğe aktarılır.

Son olarak en önemli değişken **bootcmd** ile verilendir. U-boot açıldıktan sonra bu değişkenin içinde bulunan komutları işletir. Bordu otomatik olarak açabilmek için bu değişken kullanılır.

Bütün bu değişkenlerin ilk değerleri **include/configs/omap3_beagle.h** dosyası içinde tanımlanmıştır. Nihai sistemlerde bütün tanımlar burada yapılabilir.

Bordumuzu u-boot seviyesine kadar açtık. Fakat henüz Linux işletim sistemi ile ilgili hiç bir iş yapmadık. Şimdi sıra ile çekirdeği derleyecek, sonra kök dosya sistemini kuracağız. Sonra da en nihayetinde MLO, u-boot.img, çekirdek ve kök dosya sistemini diske yazarak bordun otomatik olarak Linux ile açılmasını sağlayacağız.

Tekrar etmetke fayda görüyoruz. Burada anlatılan kavramlar borda bağlı değildir, geneldir.

7. Linux Çekirdeği

Linux çekirdeği esas ilgi alanımızdır. http://kernel.org adresinden son kararlı çekirdek sürümü indirilir. Gömülü sistemlerde asla kararsız çekirdek kullanılmamalıdır. Bu yazı yazılırken son kararlı sürüm sürüm 3.5.2 idi. Çekirdek paketi aşağıdaki gibi hedef makine için derlenir.

```
1 # tar jxvf /tmp/ftp/linux-3.5.2.tar.bz2
2 # cd linux-3.5.2
3 # export CROSS=arm-none-linux-gnueabi-

4 # make ARCH=arm CROSS_COMPILE=$CROSS omap2plus_defconfig
    configuration written to .config
```

3. adımda, arm-none-linux-gnueabi- ön eki CROSS isimli bir değişkene atanmıştır. Hem çeşni olsun diye hem de uzun uzun arm-none-linux-gnueabi- lafını yazmamak için yapılmıştır. Doğrudan 4. adımda CROSS_COMPILE=arm-none-linux-gnueabi- olarak da yazılabilirdi.

Çekirdek paketi içinde Beagleboard için yazılmış hazır bir config dosyası bulunur. 4. satırda, bu hazır dosyanın, .config ismi ile çekirdeğin kaynak kodunun açıldığı yerine hemen altına kopyalanması sağlanır. Bu .config dosyası, özellikle donanım ile ilgili pek çok tanıma sahiptir. Yukarıdaki adım 4. adım sadece 1 kez yapılmalıdır. Yoksa daha önce üzerinde çalıştığımız .config dosyası ezilecektir. Şu ana kadar çekirdeği

derlemek için ön hazırlık yapılmıştır. Derleme işi aşağıdaki gibi yapılır.

```
5 # make ARCH=arm CROSS_COMPILE=$CROSS menuconfig
```

```
6 # make ARCH=arm CROSS_COMPILE=$CROSS uImage -j2
  ...
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage

  Image Name:   Linux-3.5.2-ucan-linux
  Created:      Fri Aug 24 13:05:33 2012
  Image Type:   ARM Linux Kernel Image (uncompressed)
  Data Size:    2616224 Bytes = 2554.91 kB = 2.50 MB
  Load Address: 80008000
  Entry Point:  80008000
  Image arch/arm/boot/uImage is ready
```

5. adımda çekirdek için gerekli özelliklerin seçimi yapılır. Bu adımdan çıkıldığında 4. adımda kurulan .config dosyası güncellenmiştir. Eğer 4. adım tekrar edilirse, .config dosyası eski haline dönecektir.

Komutta bulunan -j2 ifadesi, aynı anda sistemde çalışabilecek iş sayısını tanımlar. -j2 durumunda, derleme aynı anda 2 farklı paralel proses ile yapılacak demektir. Eğer sistemde 4 işlemci varsa -j4 denildiğinde, derleme işi 4 paralel proses ile yapılacaktır. Bu durumda derleme zamanı çok azalacaktır. -jN seçeneğinde N sayısı olarak CPU sayısı verilebilir. Ya da hiperthreading varsa 2*N değeri verilebilir. N değeri daha büyük verilirse bir faydası olmaz. CPU sayısından az verilirse boşu boşuna makine başında vakit keybedilmiş olur.

.config dosyası her zaman çok önemlidir ve mutlaka çekirdek kodunun dışında, ayrı bir yerde aşağıdaki gibi saklanmalıdır.

```
# cp .config /uygun/bir/yer/kernel.config.3.5.2
```

Derleme işi 6. adımda başlar. Benimki gibi çok eski bir makinede derleme çooook uzun sürer. 6. adımda “$ make … uImage” ile derleme yapılmıştır. Bunun anlamı “derleme sonunda bana uImage imajını üret” demektir. Aynı satırı “$ make … zImage” şeklinde de yazabilirdik. Bu durumda çekirdek zImage biçiminde elde edilecekti. Çekirdek imajları pek çok biçimde elde edilebilir. Fakat arm işlemciler için genelde uImage tipinde imaj üretilir.

Esas imaj her zaman zImage imajıdır. zImage imajının en üstünde mkimage programı tarafından 64 baytlık u-boot başlığı eklenir. Başlık eklenmiş bu imaja uImage imajı denir. Adet üzerine u-boot imajı eklenmiş dosyalar “u” harfi ile başlatılırlar.

Derlenen imajlar arch/arm/boot altında bulunabilir. Aşağıda hem zImage hem de uImage çekirdek imajları listelenmiştir.

```
# cd arch/arm/boot/zImage

# ls -l uImage zImage
 -rw-r--r-- 1 root root 2616288 Aug 24 13:10 uImage
 -rwxr-xr-x 1 root root 2616224 Aug 24 13:10 zImage
```

Yukarıdaki çıkıştan da görüleceği gibi iki imaj arasında 64 baytlık fark vardır.
Bu fark mkimage başlığından gelir. Bu arada zImage isminin başında bulunan z harfi, zipped, yani sıkıştırılmış kelimesinden gelir. Bu durumda uImage imajı, esas itibarı ile sıkıştırılmış ve tepesine 64 baytlık u-boot bilgisinin eklendiği bir dosyadır. Hem zImage hem de uImage dosyaları aşağıdaki gibi file komutları ile incelenebilir.

```
1 # file zImage
  zImage: data

2 # file uImage

  uImage: u-boot legacy uImage,
  Linux-3.5.2-ucan-linux,
  Linux/ARM,
  OS Kernel Image (Not compressed),
  2616224 bytes,
  Fri Aug 24 13:05:33 2012,
  Load Address: 0x80008000,
  Entry Point: 0x80008000,
  Header CRC: 0x8BD4AD91,
  Data CRC: 0xA5E56116
```

Birinci satırda, file komutu zImage dosyasını veri dosyası gibi görmüştür. Çünkü dosya sıkıştırılmıştır ve file komutu için sıradan bir veri dosyası gibi algılanmaktadır.

2. satırda ise zImage dosyasının tepesinde bulunan 64 baytlık özel bilgi, file komutu tarafından algılanmakta ve gerekli bilgiler hemen altında listelenmektedir. Kolay okunması için araya satırlar eklenmiştir. Normalde bu çıkış tek satırdır.

Bu çıkştan da görüleceği gibi 64 baytlık u-boot başlığı kolayca analiz edilebilir. Burada

en kafa karıştıran bilgi “OS Kernel Image (Not compressed)” satırıdır. Aslında bizim imajımız zImage dosyasından dolayı sıkıştırılmıştır. Burada “Not compressed” denilen sıkıştırma hikayesi mkimage tarafından yapılan sıkıştırmadır. mkimage komutu gelen dosyanın sıkıştırılıp sıkıştırılmadığını bilmez. zImage zaten sıkıştırılmış olduğu için ayrıca sıkıştırma yapılmaz.

Örneğin bir bash betiğini u-boot imajı haline getirelim. Bu durumda mkimage komutuna “compress yap” dersek, bu betik mkimage tarafından ayrıca sıkıştırılacaktır.

u-boot imajlarının başlık bilgisi, file komutundan farklı olarak, aşağıdaki gibi mkimage komutunun kendisi ile de okunabilir.

```
$ mkimage -l uImage

Image Name:   Linux-3.5.2-ucan-linux
Created:      Sat Aug 25 14:00:07 2012
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    2516032 Bytes = 2457.06 kB = 2.40 MB
Load Address: 80008000
Entry Point:  80008000
```

U-boot programı, bir imajı “Load Address” olarak verilen adrese gözü kapalı bir biçimde yükler. “Entry Point” ile verilen adresten itibaren de yürütmeyi başlatır. Biz bu adresleri farklı vermek isteyebiliriz. Ya da “Image Name” ile verilen sadece bilgi amaçlı ismi keyfimize göre değiştirmek isteyebiliriz. Bu gibi başlık bilgileri nasıl değiştirilir? Çünkü bu bilgiler çekirdek derlemesi sırasında atanmıştır.

Başlık bilgileri doğrudan çekirdekte bulunan değişkenlerin atanması ile güncellenebillir, ama tavsiye edilmez. Daha basit olarak, çekirdek “$make … zImage” ile derlenir ve mkimage komutu ile de uImage imajı keyfimize göre kurulabilir. mkimage’nin kullanımı ilerleyen bölümlerde etraflıca incelenecektir.

Yazıyı çok şişireceği için çekirdek derlemesinden burada hiç bahsedilmemiştir. Çekirdek derlerken bazı basit kurallara dikkat etmekte yarar vardır. Bu kurallar,

“EXPERIMENTAL” işaretlenmiş özellikler, çok zorunlu değilse, çekirdeğe katılmamalıdır. Çünkü bu özellikler henüz olgunluğa erişmemiştir. Olgunluğa erişme garantisi de yoktur. Belirli bir süre sonra çekirdekten tamamen atılabilir ya da olgunluğa erişip EXPERIMENTAL etiketi kaldırılabilir.

“DEPRECATED” işaretlenmiş özellikler çekirdeğe katılmazlar. Çünkü bu tür özellikler bir kaç sürüm sonra çekirdekten tamamen kaldırılacaktır.

Gömülü sistemlerin yapacağı işler ve içinde çalıştığı donanım genelde çok kesindir. Bundan dolayı modül kullanmaya gerek yoktur. Zorunlu olmadıkça hiç bir özellik <M> şeklinde, yani modül şeklinde işaretlenmemelidir. Fakat bazı sürücüler illa ki modül olarak derlenmek ister, başka türlü çalışmazlar, ya da projedeki donanımlar değişken olabilir, ya da bazı modüller yükleme sırasında modül parametresi talep edebilirler. Bu gibi durumlarda modül kullanmak zorunludur.

Eğer modül kullanıyorsak, yani "$ make … menuconfig" sırasında, <M> olarak işaretlenmiş çekirdek özellikleri varsa, bu modüller aşağıdaki gibi ayrıca derlenmelidir. Bu komut "$ make … uImage" den sonra verilmelidir.

```
$ make ARCH=arm CROSS_COMPILE=$CROSS modules
```

Bu durumda <M> ile işaretlenen modüller ayrıca derlenir. Derlenen modüller ".ko" uzantılarına sahiptir. ko, "kernel object" demektir. Sıradan object dosyalarından bir farkı yoktur. Sadece içinde ayrı bir text alanı daha vardır ve burada modül ile ilgili bilgiler bulunur.

Modül derlemesi bittikten sonra $ find . -name "*.ko" komutu ile bütün modüller bulunur ve kök dosya sisteminde /lib/modules/3.5.2 veya daha genel olarak /lib/modules/`uname -r` altına taşınır. Kök dosya sisteminin kuruluşu ilerleyen bölümlerde etraflıca incelenecektir. Bu projede hiç modül kullanmadağımız için doğrudan örnek veremedik. Fakat mantık çok basittir, "$ make … modules" ile modüller derlenir ve "*.ko" dosyaları kök dosya sistemine taşınır, hepsi bu.

Kullanılmayan hiç bir özellik çekirdeğe katılmamalıdır. Böylece çekirdek boyu ufalır ve yükleme zamanı azalır.

Özellikle dosya sistemleri çekirdeği çok şişirir. Gömülü sistemlerde kullanılabilecek dosya cinsi çok azdır. Kullanılmayan hiç bir dosya sistemi çekirdeğe katılmamalıdır.

Ramdisk kullanılmamalı, bunun yerine ramfs kullanılmalıdır. Ramfs'in ramdisk'e göre çok büyük faydaları vardır. Ramfs'ler dosya sistemi istemezler, çekirdeğe ilgili dosya sisteminin desteğini vermeye gerek yoktur, ram belleği ramdisk gibi hemen çalıp, üzerine oturmazlar. Ramfs'ler ihtiyaç oldukça büyürler veya küçülürler, ayrıca ramfs'ler swap edilebilirler. Hem swap edilebilen hem de gerektiğinde ufalabilen dosya sistemi ramfs dışında mevcut değildir.

Bu kurallar derya denizdir ve çok fazla artırılabilir. İyi derlenmiş bir çekirdek gömülü sistemin başarısını doğrudan etkileyecektir.

Çekirdeği de derledik ve cebimize koyduk. Şimdi geriye kök dosya sisteminin

kuruluşu ve açılış betiğinin yazılması kalmıştır. Boot yükleyicileri, çekirdek, kök dosya sistemi ve açılış betikleri bir gömülü sistemin 4 temel unsurunu oluşturur.

Şimdi 3. unsur olan kök dosya sistemi kuruluşu üzerinde çalışacağız. Kök dosya sisteminin temelini Busybox sistemi oluşturmaktadır.

8. Busybox

Pek çok Linux komutu çok yeteneklidir, pek çok seçeneğe sahiptir, pek çok işi yapabilirler. Örneğin ls komutunun pek çok seçeneği (options) mevcuttur. Gömülü bir sistemde bu kadar seçeneğe gerek var mıdır? Tabii ki yoktur.

Busybox projesinde, çok gerekli olan pek çok komut yeni baştan C ile yazılmıştır. Fakat komutlara çok az özellik eklenmiştir. Böylece onlarca komut bir arada tek bir dosyada toplanmış ve gömülü sistemler için çok uygun bir hale getirilmiştir.

Busybox sistemi aynen çekirdek ortamı gibi derlenir. İhtiyacımız olan komutlar seçilir ve derleme yapılır. Derleme sonunda busybox isimli tek bir dosya elde edilir. Busybox'ın derlenmesi aşağıda verilmiştir.

http://busybox.net adresinden en son sürüm indirilir. Bu yazı yazılırken son sürüm 1.20.2 idi.

```
$ wget http://busybox.net/downloads/busybox-1.20.2.tar.bz2
$ tar jxvf busybox-1.20.2.tar.bz2
$ cd busybox-1.20.2
$ make menuconfig
```

make menuconfig sırasında gelen menülerden aşağıdaki seçimler yapılır. Diğer seçimler bizim için şu anda hayati öneme sahip değildir.

```
Build Options -->
 [ ] Build BusyBox as a static binary (no shared libs)
 [ ]   Build BusyBox as a position independent executable
 [ ] Force NOMMU build
 [ ] Build shared libbusybox
 [ ] Build with Large File Support (for accessing files > 2 GB
 (arm-none-linux-gnueabi-) Cross Compiler prefix
 ()  Path to sysroot
 ()  Additional CFLAGS
 ()  Additional LDFLAGS
 ()  Additional LDLIBS
```

Yukarıda görüleceği gibi “Build Options” mensünde çapraz derleyicinin ön eki verilmiştir. Busybox daha farklı yöntemlerle de çapraz derlenebilir. Çok basit olduğu için, bizler çapraz derleme ön ekini “Build Options” kısmında açıkça verdik. Kullanmadık ama, aynı özellik çekirdek derlemesi sırasında da kullanılabilirdi.

Eğer aynı anda hem menuconfig içinde hem de CROSS_COMPILE değişkeni ile çapraz derleme ön eki verilirse, CROSS_COMPILE ile verilen ön eke öncelik verilir.

Busybox derlemesindeki en önemli kısım, aşağıdaki gibi “Installation Options” menüsünde bulunur.

```
Installation Options ("make install" behavior) -->
   What kind of applet links to install (as soft-links)  --->
   (/nk/blog/1/RootFS) BusyBox installation prefix
```

Her zaman “soft link” kullanılmalıdır. Hard link tam bir muammadır, asla kullanılmamalıdır. “Busybox installation prefix” ile busybox sisteminin kurulacağı bir dizin ismi verilir. Busybox buraya kurulacaktır. Busybox kurulurken /bin, /sbin ve /usr dizinlerini yaratır. /bin dizini içine busybox programının kendisi kopyalanır. Diğer bütün dizinlerde ise komutlara ait sembolik bağlar bulunur.

Bizler busybox’ı /nk/blog/1/RootFS dizini altına kuracağız. Daha sonra /etc, /proc, /lib gibi diğer dizinleri el yordamı ile yaratarak kök dosya sistemini tamamlayacağız. Diğer bir deyişle, kök dosya sisteminin /bin, /sbin ve /usr dizinlerini busybox kuracaktır. Diğer dizinleri el ile kuracağız.

Aynen çekirdek derlemesinde olduğu gibi, busybox’ın .config dosyası aşağıdaki gibi farklı bir yerde saklanmalıdır.

```
# cp .config  /uygun/bir/yer/busybox.config.1.20.2
```

Busybox, aşağıdaki gibi tek bir make komutu ile çapraz olarak derlenir.

```
# make -j2

# ls -l busybox
 -rwxr-xr-x 1 root root 916872 Aug 24 13:26 busybox
```

Elde edilen kodun büyüklüğü yaklaşık 1 MB’dir. Aslında gömülü sistem için bu çooook büyük bir değerdir. Hiç bir gömülü sistemde bu kadar çok Linux komutu bulunmaz. Nihai sistemde kullanılmayan bütün komutlar busybox içinde atılmalıdır. Bunun da çok

basit bir yolu vadır. Gömülü sistemde kullanılan komutların tamamı açılış betiklerinde ortaya çıkar. Açılış betikleri incelenmeli ve burada geçen komutlar busybox içine dahil edilmelidir. Bu durumda görülecektir busybox'ın boyu çok ufalacaktır. Fakat şimdiki bordların bellek ve disk kapasiteleri nadasa bırakılacak kadar büyüktür. Bundan dolayı bu tür optimizasyonlar gözardı edilmektedir. Fakat yine de yapılmasında fayda vardır. Aşağıda file komutu ile busybox dosyası incelenmektedir.

```
# file busybox

busybox: ELF 32-bit LSB executable,
         ARM, version 1 (SYSV),
         dynamically linked (uses shared libs),
         stripped
```

Çıkıştan da görüleceği gibi, busybox'ın kodu strip edilmiştir. Çünkü birinci amaç kod boyunun kısa olmasıdır. Ayrıca "dynamically linked" ifadesi bize busybox'ın dinamik olarak derlendiğini ve kütüphanelere ihtiyaç duyduğunu söyler. Bu kütüphaneler ilerleyen bölümlerde /lib altına kurulacaktır.

"Build Options" menüsünde, "[*] Build busybox as a static binary" seçimi yapmış olsaydık, busybox statik olarak derlenecek ve hiç bir kütüphaneye ihtiyaç duymayacaktır. Bu arada busybox kodunun da aşırı derecede şişeceğini hatırlatalım.

Derleme işi bittikten sonra, busybox aşağıdaki gibi kök dosya sisteminin içine kurulur. Eksik olan diğer dizinler yine aşağdaki gibi yaratılır.

```
1 # make install
2 # cd /nk/blog/1/RootFS

3 # ls -l

 total 12
 drwxr-xr-x 2 root root 4096 Aug 24 13:28 bin/
 lrwxrwxrwx 1 root root   11 Aug 24 13:28 linuxrc -> bin/busyb
 drwxr-xr-x 2 root root 4096 Aug 24 13:28 sbin/
 drwxr-xr-x 4 root root 4096 Aug 24 13:28 usr/

4 # rm linuxrc

5 # du -ks .
    924 .

6 # mkdir dev etc home lib mnt proc root tmp var sys
```

```
7 # ln -s /sbin/init

8 # ls -l
  drwxr-xr-x 2 root root 4096 Aug 25 14:12 bin/
  drwxr-xr-x 2 root root 4096 Aug 24 13:36 dev/
  drwxr-xr-x 2 root root 4096 Aug 25 13:23 etc/
  drwxr-xr-x 2 root root 4096 Aug 24 13:36 home/
  lrwxrwxrwx 1 root root   10 Aug 24 13:37 init -> /sbin/init
  drwxr-xr-x 3 root root 4096 Aug 24 13:45 lib/
  drwxr-xr-x 2 root root 4096 Aug 24 13:36 mnt/
  drwxr-xr-x 2 root root 4096 Aug 24 13:36 proc/
  drwxr-xr-x 2 root root 4096 Aug 24 13:36 root/
  drwxr-xr-x 2 root root 4096 Aug 25 14:12 sbin/
  drwxr-xr-x 2 root root 4096 Aug 25 20:12 sys/
  drwxr-xr-x 2 root root 4096 Aug 24 13:36 tmp/
  drwxr-xr-x 4 root root 4096 Aug 25 14:12 usr/
  drwxr-xr-x 2 root root 4096 Aug 24 13:36 var/

9 # cp -a /ornek/etc/dizini/* etc/

10 # ls -l etc
   total 120
  -rw-r--r-- 1 root root     5 Oct  6  2005 TZ
  -rw-r--r-- 1 root root    28 Jul 11  2007 adjtime
  -rw-r--r-- 1 root root    29 Oct  6  2005 exports
  -rw-r--r-- 1 root root   148 Jul  2  2009 fstab
  -rw-r--r-- 1 root root    97 Jul  7 22:02 group
  -rw-r--r-- 1 root root    24 Apr 22  2010 gshadow
  -rw-r--r-- 1 root root    40 Feb  1  2011 hosts
  -rw-r--r-- 1 root root    39 Jan 21  2011 inetd.conf
  -rw-r--r-- 1 root root   186 Jul 24 19:44 inittab
  -rw-r--r-- 1 root root    21 Jul 24 20:04 issue
  -rw-r--r-- 1 root root   724 Jun 27 10:25 ld.so.cache
  -rw-r--r-- 1 root root     1 Jun 27 10:21 ld.so.conf
  lrwxrwxrwx 1 root root    12 May 14  2011 mtab -> /proc/moun
  -rw-r--r-- 1 root root   154 Oct  6  2005 nsswitch.conf
  -rw-r--r-- 1 root root   315 Jan 27  2011 passwd
  -rwxr--r-- 1 root root   176 Jul 24 19:59 profile*
  -rw-r--r-- 1 root root  1623 Oct  6  2005 protocols
  -rwxr-xr-x 1 root root   396 Jul 24 20:03 rcS*
  -rw-r--r-- 1 root root    20 Jul 24 14:04 resolv.conf
  -rw-r--r-- 1 root root  1615 Oct  6  2005 rpc
  -rw-r--r-- 1 root root    81 Jul 24 19:57 securetty
  -rw-r--r-- 1 root root 29366 Oct  6  2005 services
  -rw-r--r-- 1 root root   251 Jan 27  2011 shadow
  -rw-r--r-- 1 root root    61 Jul  7 16:36 sudoers

11 # cp -a /ornek/lib/dizini/* lib/
```

```
12 # ls -l lib

 -rwxr-xr-x 1 root root  122212 Jun 27 09:33 ld-2.12.2.so
 lrwxrwxrwx 1 root root      12 Feb 14  2012 ld-linux.so.3 ->
 -rwxr-xr-x 1 root root 1184972 Jun 27 09:32 libc-2.12.2.so
 lrwxrwxrwx 1 root root      14 Feb 14  2012 libc.so.6 -> libc
 -rwxr-xr-x 1 root root     135 Jan 27  2012 libgcc_s.so
 -rwxr-xr-x 1 root root  182397 Jan 27  2012 libgcc_s.so.1
 -rwxr-xr-x 1 root root  427528 Jan 27  2012 libm-2.12.2.so
 lrwxrwxrwx 1 root root      14 Feb 14  2012 libm.so.6 -> libm
 drwxr-xr-x 2 root root    4096 Jul 24 14:06 modules/

13 # du -ks RootFS/
   3104    RootFS/
```

Yukarındaki her bir adımı kısaca açıklayalım.

/nk/blog/1/RootFS dizinin içinin tamamen boş olduğunu kabul ediyoruz.

1. adımda /nk/blog/1/RootFS altında otomatik olarak /bin, /sbin ve /usr dizinleri yaratılır. busybox programı bin/ altına atılır ve seçilen bütün komutların sembolik linkleri atanır.

3. adımda busybox'ın kurduğu dizinler görülebilir. linuxrc dosyası ramdisk destekli açılışlar içindir. Artık çok eski kalmıştır. 4. adımda bu sembolik link silinir.

6. adımda kök dosya sistemi için gerekli olan diğer dizinler yaratılır.

7. adımda /sbin/init'e /init bağlanır. Bizler kök dosya sistemi olarak initramfs kullanacağız. Initramfs'li sistemlerde çekirdek kök dosya sistemini bağladıktan hemen sonra /init programını gözü kapalı bir biçimde işletir. Bu bir betik dahi olabilir. Ya da /bin/sh gibi bir programa sembolik bağ olabilir. Bu durumda kök dosya sistemi bağlandıktan hemen sonra kabuk promptu gelecektir. Bizler /init programına /sbin/init programını bağladık. Böylece kök dosya sistemi bağlandıktan sonra çekirdek /sbin/init'i çalıştıracaktır. Bu programın işleyişine ilerleyen bölümlerde değinilecektir. Son bir hatırlatma, /init programı veya sembolik olarak gösterdiği program çalışabilir modda, yani x modunda ve yürütülebilir bir program olmalıdır. Yani ELF veya betik olmalıdır.

9. adımda etc/ dizini, örnek bir etc/ dizini kullanılarak doldurulur. Okuyucu amacına göre etc/ dizini kurabilir.

11. adımda /lib dizini gerekli kütüphaneler ile doldurulur. Bizler sadece busybox'ın

kullanacağı kütüphaneleri ekledik. Okuyucu ihtiyacı olan bütün kütüphaneleri buraya kopyalayabilir. Kütüphaneler Angstrom dağıtımından yürütülmüştür. Beagleboard'da çalışan herhangi bir dağıtımdan kütüphaneler kopyalanabilir.

Kök dosya sistemimiz artık hazırdır. 13. adımda kök dosya sisteminin kapladığı alan raporlanmıştır. Kök dosya sistemi bu hali ile yaklaşık 3MB alan kaplar. Ayrıca gerekli olur diye strace programı ve ilk.dynamic programı da kök dosya sistemine eklenmiştir.

Kök dosya sistemini masaüstü makinesinin diskinde hazırladık. Diğer bir deyişle, diskte /nk/blog/1/RootFS dizini altında kök dosya sistemimiz yatmaktadır. Kök dosya sistemini bu hali ile gömülü sisteme aktaramayız. Kök dosya sistemlerinin gömülü sistemlerde kullanılması için envai çeşit teknik vardır. Biz burada çok yeni ve en verimli olan initramfs tekniğini kullanacağız. Bunun için masaüstü sistemde bulunan RootFS/ dizini initramfs yöntemine göre paketlenmelidir.

9. Kök Dosya Sisteminin Initramfs Yöntemi ile Hazırlanması

Initramfs (inital ram file system) yöntemi, initrd'ye almaşık bir çözüm olarak ortaya atılmıştır. Initramfs sistemi esas itibari ile kök dosya sistemi olarak kullanılmaz. Kök dosya sistemine geçiş için kullanılan ara bir sistemdir. Genelde de modül yüklemek için kullanılır. Fakat bizim projemizin gereklerini tam olarak karşıladığı ve içinde pek çok kavram barındırdığı için initrmafs sistemini esas kök dosya sistemi olarak kuracağız.

Öncelikle RootFS dizinine doldurmuş olduğumuz bütün dosyalar aşağıdaki gibi cpio arşivi haline getirilir.

```
# cd RootFS/

# find . | cpio -H newc -o | gzip -v -9 > ../initramfs.cpio.gz
 5915 blocks
 48.3%

# ls -l ../initramfs.cpio.gz
 -rw-r--r-- 1 root root 1564879 Aug 25 20:54 ../initramfs.cpio
```

Bu tür "|" ile birleştirilmiş komutları analiz etmemin en iyi yolu, her birini tek tek prompttan girerek denemektir. "find ." komutu RootFS dizini içinde bulunan bütün dosyaları alt alta listeler. Initramfs sistemi sadece cpio arşivi ile çalışır.

cpio programı, birden fazla formatla arşivleme yapabilir. Initramfs sistemi newc formatını destekler. Bundan dolayı "-H newc" ile initramfs'in formatı açıkça verilir. -o seçeneği "output" demektir. cpio komutu, find komutundan gelen dosya isimlerini newc formatına göre biraraya toplar ve elde ettiği sonuçları ekrana yazar. cpio

komutundan sonraki “|” sayesinde ekrana gidecek bilgiler gzip programına girdi olur. gzip de elde ettiği sıkıştırılmış dosyayı ekrana yazar. Ama “>” işareti sayesinde ekrana yazılacak bilgiler dosyaya yönlendirilir. En nihayetinde RootFS’in initramfs için formatlanmış ve sıkıştırılmış hali, initramfs.cpio.gz dosyasına yazılır. “ls -l” çıkışından da görüleceği gibi, bütün kök dosya sistemimiz yaklaşık 1.5 MBayt yer kaplamaktadır. Diğer bir deyişle, RootFS dizini %50 oranına sıkıştırılmıştır. Çünkü “$ du -ks RootFS” ile baktığımızda, RootFS dizininin boyu 3MBayt civarındaydı. Bu sıkıştırma oranı zaten gzip tarafından %48.3 olarak yukarıda raporlanmıştır.

Kök dosya sistemini, cpio imajı haline getirdik. Fakat hale bu sistemi kullanmamız mümkün değildir. Çünkü kök dosya sistemini açılış sırasında u-boot yükleyecektir. U-boot programı da ne yüklerse yüklesin mutlaka yükleyeceği programın tepesinde 64 baytlık başlık bilgisi arar. 64 baytlık bu başlık bilgisi aşağıdaki gibi mkimage programı ile yazılabilir. Kolay okunması için aralara satır eklenmiştir. “# mkimage -A arm -O linux … uInitrd” komutu tek satırda yazılmalıdır.

```
# mkimage -A arm
          -O linux
          -T ramdisk
          -C none
          -a 0
          -e 0
          -n UcanLinux
          -d initramfs.cpio.gz
             uInitrd

Image Name:   UcanLinux
 Created:      Sat Aug 25 20:55:45 2012
 Image Type:   ARM Linux RAMDisk Image (uncompressed)
 Data Size:    1564879 Bytes = 1528.20 kB = 1.49 MB
 Load Address: 00000000
 Entry Point:  00000000
```

mkimage programı, u-boot paketini derlerken elde edilmiş ve /usr/local/bin altına kopyalamıştı. mkimage programının seçenekleri aşağıdaki gibi listelenebilir.

```
# mkimage --help

Usage: ./mkimage -l image
          -l ==> list image header information

       ./mkimage [-x] -A arch -O os -T type -C comp -a addr -e
                 -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)
            -n ==> set image name to 'name'
            -d ==> use image data from 'datafile'
            -x ==> set XIP (execute in place)

         ./mkimage [-D dtc_options] -f fit-image.its fit-image

         ./mkimage -V ==> print version information and exit

# ./mkimage -V
mkimage version 2012.07
```

mkimage komutunun kullanım bilgisinden de görüleceği gibi bütün seçenekler son derece açıktır. Yukarıda verilen örneğin bazı seçeneklerine kısaca değinelim.

“-T ramdisk” ile elde edilen imajın ne tipten olduğu belirtilir. U-boot, ramdisk tipindeki imajları belleğe kendisi yükler, çekirdeğe bırakmaz. Çekirdeğe sadece başlangıç adresini parametre olarak geçirir.

“-C none” ile imajda sıkıştırma yapılmadığı söylenir. Aslında cpio arşivi sıkıştırılmıştır. Ama açma işi çekirdek tarafında yapılacaktır. U-boot tarafında bir sıkıştırma/açma yapılmayacaktır. U-boot’un açma işlemi yapmaması için “none” girilmiştir.

“-a 0″ ile imajın yükleneceği adres verilir. Fakat ramdiskin yükleneceği adres, daha sonra bootm komutu ile açıkça verilecektir. Bundan dolayı adres 0 verilmiştir. Yani bu adresin gerçekte bir anlamı yoktur.

“-e 0″ ile de yürütmenin başlangıç adresi verilir. Bu adresin yine “-a 0″ daki gibi bir anlamı yoktur.

“-n UcanLinux” ile imaja bir isim verilir. Daha sonra “$ mkimage -l” komutu ile imaj incelendiğinde, ne olduğu kolay anlaşılır. Sadece bilgi içindir. Özellikle birkaç imaj varsa, imajlar bu sayede birbirlerinden kolayca ayırt edilebilirler.

“-d initramfs.cpio.gz” ile, tepesine 64 baytlık başlık bilgisinin yerleştirileceği dosya ismi verilir. mkimage programı dosyanın tipine vs. hiç bakmaz. Gelen dosya ne olursa olsun, gözü kapalı bir biçimde bu dosyanın tepesine başlık bilgisini yazar.

uInitrd ile de u-boot imaj dosyasının adı verilir. Alışkanlık gereği, u-boot başlığı taşıyan

bütün imajlar, “u” harfi ile başlatılırlar. Bizler de bu alışkanlığa sadık kaldık.

Başlık bilgisinde sadece bu kadar bilgi yoktur. Aynı zamanda checksum değeleri de bulunur. U-boot programı, bir imajı yükleyeceği zaman öncelikle checksum değelerini kontrol eder. Eğer bu değerler tutmaz ise yükleme işini hata vererek bırakır.

mkimage, basit bir C programıdır. Daha fazla bilgi için u-boot paketi içinde bulunan, tools/mkimage.c programı incelenebilir.

Özetlersek, önce RootFS/ dizininin içine kök dosya sistemini kurduk. Sonra bu dizini cpio arşivi haline getirdik ve sıkıştırdık. Sonra sıkıştırılmış bu dosyanın tepesine mkimage programı ile u-boot için gerekli olan başlık bilgilerini ekledik. Henüz bitmedi !

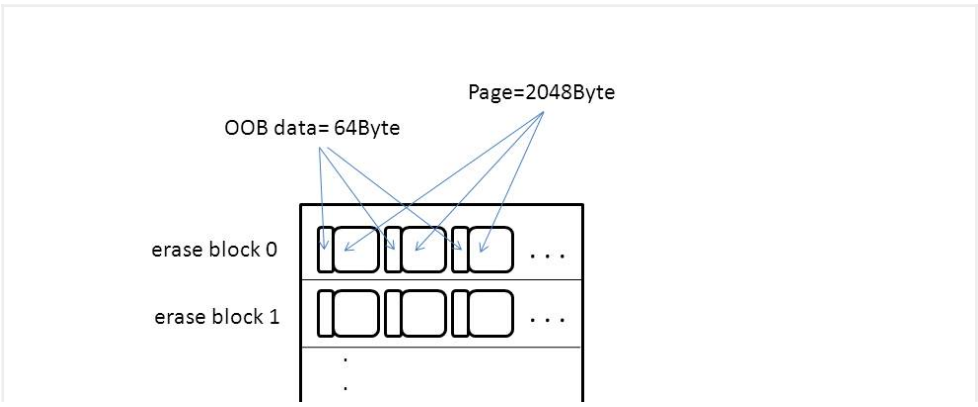
Flash diske kaydedilecek her dosyanın boyunun, diskin page değerinin tam katı olması gerekir. uInitrd programı bu şartı sağlamadan diske yazılırsa, “Attempt to read non page aligned data” hatası alınır. Bu hatayı almamak için kernel dahil flash’a yazılacak her dosya mutlaka page değerine göre hizalanmalıdır.

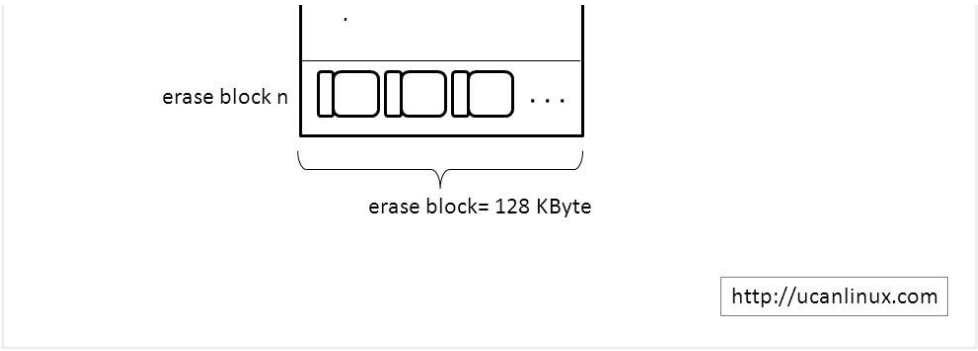
Öncelikle, elimizde bulunan bordun page değerini öğrenmeliyiz. Bunun için el kitaplarına bakılabilir. Ya da u-boot seviyesinde iken aşağıdaki gibi, nand info komutu ile bu değer elde edilebilir.

```
UcanLinux > nand info

Device 0: nand0, sector size 128 KiB
 Page size      2048 b
 OOB size         64 b
 Erase size   131072 b
```

Burada raporlanan değeler çok önemlidir. Öncelikle “Page size” değeri, yani 2048 değeri, flash disk için hizalama değeridir. Diske yazılan her dosya mutlaka 2048′ın tam katı olmalıdır. “Page size” aynı zamanda en küçük okuma/yazma yapılabilecek veri boyudur. Şekil 1.2′de bir flash diskin mealen yapısı verilmiştir.





— Şekil 1.2: Bir flash diskin iç blokları.

Erase veya sector size değeri, 128K olarak verilmiştir. Şekil 1.2′de erase block resmedilmiştir. Bu değeri şu an için kullanmayacağız. Fakat jffs2 gibi bir dosya sistemi kuracak olursak, erase size değeri açıkça bilinmelidir. Eğer erase size değeri düzgün verilmezse, dosya sistemi yine mount edilebilir ve kullanılabilir. Fakat dosya sistemi çok verimsiz çalışır ve aşırı derecede ikaz mesajı üretilir.

Ayrıca NAND diskler içinde genelde hatalı bloklar (bad blocks) bulunur. Her bir hatalı bloğun boyu tam “erase size” değerindedir. Örneğin elimizdeki diskte 4 hatalı blok mevcutsa, 4*erase size değeri kadar bayt diskte kullanım dışıdır. U-boot seviyesinde iken “nand bad” komutu girilirse, hatalı blokların başlangıç adresleri tek tek listelenir.

Beagleboard’un flash diskinin page değerinin 2048 olduğunu öğrenmiştik. Her page değerinin genelde 1/32′si kadar bilgi ayrıca bir blokta tutulur. Bu bloğa OOB (Out Of Band) blok denir. Bu durumda 2048/32= 64 baytlık OOB verisi her page değeri ile birlikte saklanır. Şekil 1.2′de OOB ve Page blokları yan yana resmedilmiştir.

OOB niçin vardır? NAND diskler, NOR diskler veya ROM’lardan farklı olarak çok yoğundurlar. GB seviyesinde kapasiteleri mevcuttur. Külliyatlı miktarlarda üretilirler. Çok ucuzdurlar. Bu güzelin basit bir kusuru vardır. NAND diskler hataya çok açık yapıya sahiptirler. Bu hataları tespit edip düzeltmek için, genelde her 512 baytlık veri için 16 baytlık OOB verisi kullanılır. OOB verisi içinde ECC (Error Correcting Code) bilgisi ve diğer üst (meta) bilgiler mevcuttur. Okunan her veri mutlaka ECC denetiminden geçer ve gerekirse düzeltilir.Her dosya sistemi ECC verisini ve OBB bloğundaki verileri kullanamaz. Bundan dolayı flash disklere VFAT, EXT2 gibi, standard disklere kurulan dosya sistemlerinin kurulması tavsiye edilmez. Doğrudan flash disklere destek veren ve genelde “flash file systems” olarak geçen, jffs2, yaffs2, logfs, ubifs gibi dosya sistemleri kullanılır. Flash diskler için kullanılan dosya sistemlerini ayrı bir yazı konusu yapmak istiyorum. Yoksa bu yazı bitmeyecek.

Bu kadar lafın sonunda, esas dikkat edeceğimiz nokta şudur. Diske yazılacak her dosyanın boyu, page size değerinin tam katı olmalıdır. Örnek diskimizin page size değeri 2048′dir. O halde uInitrd imajının boyu 2048′in tam katı yapılmalıdır. Bunun için

bir kaç yöntem mevcuttur. Bunların içinde en basit olanı dd komutu ile yapılandır. Aşağıdaki örnekte uInitrd, tam 2048'in katı yapılmaktadır.

```
# dd if=uInitrd of=uInitrd.2048 bs=2048 conv=sync

 764+1 records in
 765+0 records out
 1566720 bytes (1.6 MB) copied, 0.00880356 s, 178 MB/s
```

if, input file; of, output file demektir. Nihai dosyanın adı 2048'in tam katı olduğunu belirtmek için uInitrd.2048 olarak verilmiştir. bs, block size demektir. bs değeri "page size" olarak verilmiştir. dd komutu uInitrd dosyasını bs adet bloklarla okur ve uInitrd.2048 dosyasına yazar. Bu kadarı ile bir cp komutundan hiçbir farkı yoktur. Fakat conv=sync seçeneği ile, artık kalan blokların sıfırlar ile doldurulması sağlanır.

Rapor edilen 764+1 değeri, tam 764 adet blok okunduğunu söyler. +1 ise, 2048'den daha az elemana sahip bir blok olduğunu belirtir. İşte conv seçeneği eksik kalan bu bloğun içini doldurur. Böylece çıkışta 765 adet tam blok olacaktır.

uInitrd ile uInitrd.2048 arasında işlevsellik olarak hiç bir fark yoktur. Ayrıca yapılan bu işin u-boot ile de bir ilgisi yoktur. Flash disklerin page değerleri ile okuma/yazma gerekliliğinden dolayı bu işlem yapılmıştır.

Her iki dosya aşağıdaki gibi incelenebilir.

```
# ls -l uInitrd uInitrd.2048

 -rw-r--r-- 1 root root 1564943 Aug 25 20:55 uInitrd
 -rw-r--r-- 1 root root 1566720 Aug 25 20:56 uInitrd.2048
```

Blok hesabı aşağıda özetlenmiştir.

```
1564943 / 2048 = 764.1323 = 764+ 1 blok.
1566720 / 2048 = 765.0000 = 765+ 0 blok.
```

Sırası gelmediği için yapmamıştık. Flash diske yazılacağı için çekirdeğin de page değerine göre hizalanması gerekir. Çekirdeğin hizalanması da aşağıdaki gibi yapılır.

```
# dd if=uImage of=uImage.2048 bs=2048 conv=sync

 1228+1 records in
```

```
1229+0 records out
2516992 bytes (2.5 MB) copied, 0.0105464 s, 239 MB/s
```

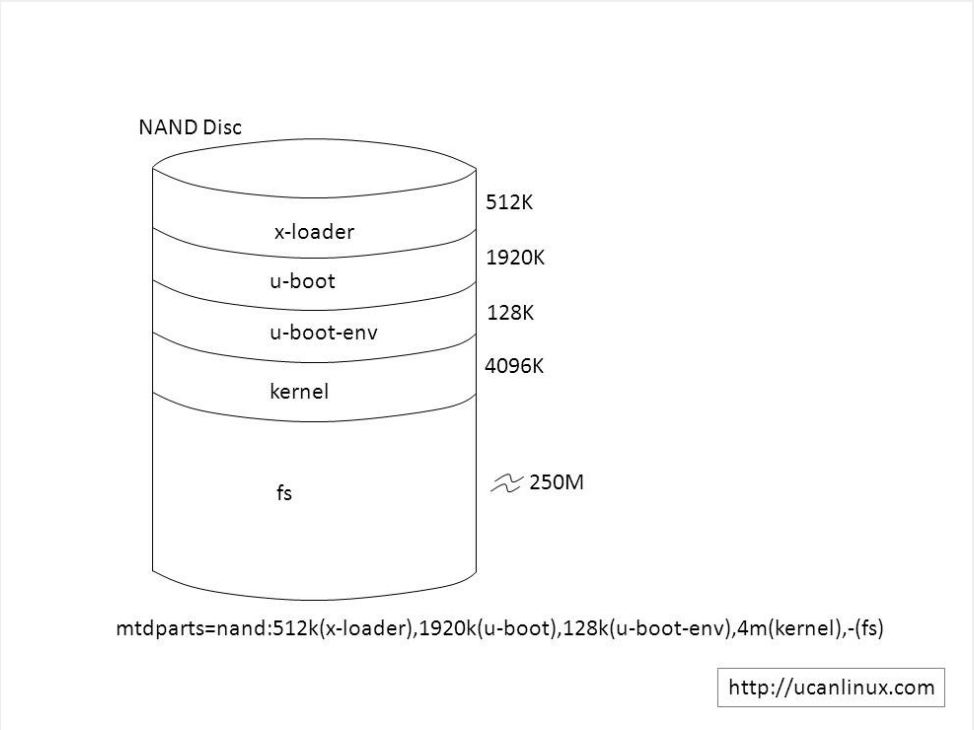
Kök dosya sistemleri çekirdek tarafından çeşitli şekillerde temin edilip bağlanabilirler. Bu konuda inanılmaz bir çeşitlilik vardır. Kök dosya sistemleri

1. Doğrudan initramfs şeklinde çekirdeğin içine gömülebilirler,
2. Çekirdekten ayrı, ama yine initramfs şeklinde kurulabilirler,
3. Flash diskte kurulabilirler,
4. MMC/SD üzerinde kurulabilirler,
5. NFS üzerinden mount edilebilirler, vs.

Bunların her biri ayrı bir yazı konusudur. Kök dosya sistemi mutlaka projenin amacına uygun seçilmelidir. Yukarıdaki listede bulunan 2. çözüm bizim projemiz için biçilmiş kaftandır. Çünkü ani kapanmaya karşı %100 güvenlidir. Dosya boyutu ufak olduğu için doğrudan ram bellekte çalışabilir, vs.

10. NAND Diskin Varsayılan Bölümleri

Beagleboard'da şekil 1.3'de verildiği gibi önceden tanımlanmış, varsayılan disk bölümleri mevcuttur.



— Şekil 1.3: Varsayılan disk bölümleri.

Şekil 1.3'de verilen disk bölümlendirmesi MS-DOS bölümlendirmeleri ile

karıştırılmamalıdır. Flash disklerde bu bölümlendirmeler tamamen mantıksaldır. Ayrıca bu bölümlendirme bilgilerini saklayan MBR gibi özel bir disk sektörü mevcut değildir.Linux çekirdeği bölümlerle ilgili bilgileri mtdparts isimli çekirdek parametresinden alır.

Şekil 1.3′de verildiği gibi BeagleBoard'un flash diski toplam 5 parçaya ayrılmıştır. Bütün parçalar isimlendirilmiştir.

Birinci parçanın ismi x-loader olarak verilmiştir. x-loader esas itibari ile u-boot'tan türetilmiş çok basit bir yükleyicidir. x-loader yükleyicisinin bir program ile imzalanmış haline MLO denir. Bundan dolayı u-boot derlendiğinde MLO programı da üretilmiş olur. Genelde bütün NAND disklerin birinci parçası MLO gibi birinci aşama boot yükleyicileri için ayrılmıştır. Varsayılan bölümlendirmede MLO için toplam 512 K yer ayrılmıştır.

İkinci disk bölümünün adı u-boot ile verilmiştir. u-boot bölümü için toplam 1920 K yer ayrılmıştır.

U-boot çok parametrik bir programdır. U-boot parametreleri için 3. disk bölümü ayrılmıştır. u-boot-env olarak isimlendirilen bu bölüm için 128K yer ayrılmıştır. Bütün kalıcı parameterler bu bölümde saklanır. U-boot içinde iken printenv komutu ile mevcut bütün parametrelerin listesi elde edilir. saveenv komutu ile de bu parametreler kalıcı olarak u-boot-env disk bölümüne yazılır. U-boot bir sonraki boot aşamasında önce bu bölümü okur ve parametreleri atar.128K'lık kapasite u-boot parametreleri için çok büyüktür. Disk sorunu olan gömülü sistemlerde bu bölüm tamamen atılabilir. Gerekli parametreler doğrudan u-boot programında ilgili bordun header dosyasında tanıtılabilir.

Sonraki disk bölümü çekirdek için ayrılmıştır. Bu bölüme kernel adı verilmiş ve 4K'lık bir yer ayrılmıştır. Derlenmiş çekirdek 4K'dan daha büyük olamaz. Daha büyük bir yere gerek olduğunda mtdparts parametresi yeniden tanıtılmalıdır.

Diskin geri kalanı ise fs (file system) ismi ile bölümlendirilmiştir.

Bu disk bölümlendirmesi varsayılan bir bölümlendirmedir. Bu bölümlendirmenin tanımı Şekil 1.3′de mtdparts ile gösterilmiştir. mtdparts parametresi varsayılan bölümler için aşağıdaki gibidir. Kolay okunması için satır bölünmüştür. İlgili ifade tek satırda yazılmalıdır.

```
mtdparts=nand:512k(x-loader),
             1920k(u-boot),
             128k(u-boot-env),
             4m(kernel),
```

```
                -(fs)
```

Yukarıdaki ifade Şekil 1.3′de verilen bölümlendirme için açıklamaya gerek bırakmayacak kadar aşikardır. -(fs)'de bulunan "-" işareti, "diskin geri kalanı" demektir.

Bu disk bölümlendirmelerindeki bütün isim ve kapasitelerin tamamen keyfi olduğunu hatırlatalım. Okuyucu isim ve kapasiteleri amacına göre tanımlayabilir. Fakat mevcut sıranın korunması her zaman tavsiye edilir.Genelde bütün NAND diskler bu sırada bölümlendirilirler. Bu bir tür "önem sırası"dır.

Öncelikle birinci, sonra ikinci aşama boot yükleyicileri diskin en başında bulunmalıdır. Sonra env ve çekirdek bölümleri bulunmalıdır. İsim ve kapasite önemli değildir ama sıra önemlidir. Çünkü hemen hemen bütün NAND disklerde hatalı sektörler bulunur. Ama bu sektörlerin diskin başında bulunmasına izin verilmez. Yani diskin ilk bölümleri her zaman en sağlam yerlerdir.

MLO olmadan, u-boot işe yaramaz. U-boot olmadan kernel işe yaramaz. Kernel olmadan kök dosya sistemi anlamsızdır. Bundan dolayı en önemli program, yani MLO ilk bölüme yazılır. En az önemli olan kök dosya sistemi ise en son disk bölümlerine kurulur.

NAND diski keyfimize göre bölümlere ayırabileceğimizi söylemiştik. Çekirdeğin mtdparts değerini anlayabilmesi için aşağıdakiler yapılmalıdır.

Çekirdeğin mtdparts ifadesini ayrıştırabilmesi için "make menuconfig" aşamasında aşağıdaki özellik çekirdeğe katılmalıdır.

```
Device Drivers  --->
   <*> Memory Technology Device (MTD) support  --->
       [*]   Command line partition table parsing
```

U-boot seviyesinde mtdparts değişkeni yukarıda verildiği formatta, çevre değişkeni olarak tanıtılmalıdır. Çevre değişkeninin ismi de yine mtdparts olarak verilir. Bundan dolayı u-boot tarafında "mtdparts=mtdparts= …" şeklinde garip bir durum ortaya çıkar. Birinci değişken ismi u-boot için ve birinci eşitten sonraki bütün ifade ise çekirdeğe olduğu gibi aktarılmak içindir.

Son olarak Linux çekirdeğinin nand diski nasıl tanıdığına bakmak gerekir. Bunun için tabii ki daha önce en az 1 kere Linux ile açmak gerekir. Aşağıdaki gibi basit bir komut ile çekirdeğin NAND'a verdiği isim tespit edilir.

```
1 root@UcanLinux:~ # dmesg|grep nand
```

```
2 Kernel command line:
3          console=ttyO2,115200n8
4          root=/dev/ram
5          mtdparts=omap2-nand.0:512k(x-loader),
6                                 1920k(u-boot),
7                                 128k(u-boot-env),
8                                 4m(kernel),
9                                 100m(fs1),
10                                50m(fs2),
11                                -(fs3)

12 7 cmdlinepart partitions found on MTD device omap2-nand.0

13 Creating 7 MTD partitions on "omap2-nand.0":

root@UcanLinux:~ #
```

Burada esas aranması gereken ifade, 13. satırda verilen " …. on omap2-nand.0″ ifadesidir. Buradan çekirdeğin nand diski hangi isimle tanıdığı anlaşılır. Sonuçta mtdparts değişkenine isim olarak bu değer girilir.

Biz kendi projemizde diski 7 parçaya böldük. Yukarıdaki çıkışta 2 ve 11. satırların tamamı tek satırdır. Kolay okunması için yeni satırlar eklenmiştir. Bizim projemizde kullandığımız bölümlendirme tanımı, u-boot tarafı için, aşağıda verilmiştir. Bütün ifadenin, hiç enter'a basmadan bir kerede yazılması gerektiğini tekrar hatırlatalım. Kolay okunması için satırlar eklenmiştir.

```
UcanLinux > setenv mtdparts "mtdparts=omap2-nand.0:512k(x-load
             1920k(u-boot),128k(u-boot-env),4m(kernel),
             100m(fs1),50m(fs2),-(fs3) "
```

Bizim yaptığımız tanımdan da görüleceği gibi ilk 4 bölüme sadık kalınmış ama sonraki fs bölümü 3 farklı diske bölünmüştür.

Bu yazıda verilen örnek gömülü sistemde, kök dosya sistemi initramfs tekniği ile kurulmuştur. Initramfs iki türlü kullanılabilir. Birinci tür kullanımda, çekirdek derlemesi sırasında RootFS'in değeri aşağıdaki gibi çekirdeğe tanıtılır.

```
General setup  --->
  [*] Initial RAM filesystem and RAM disk (initramfs/initrd) s
       (/nk/blog/1/RootFS)    Initramfs source file(s)
```

Bu durumda çekirdek derlemesi bittiğinde RootFS içinde bulunan kök dosya sistemi çekirdeğin kaynak kodu içine gömülür. Daha önce yapmış olduğumuz cpio arşivi ve ilgili diğer işleri yapmamıza gerek kalmaz. Literatürde tavsiye edilen yöntem de budur. Fakat biz bunu yapmadık. Daha zahmetli olan diğer yolu seçtik. Yani bu yazının başında anlatıldığı gibi cpio arşivi yaparak, nihayetinde uInitrd imajını elde ettik. Niçin?

Bunun tek sebebi ticari bir uygulama geliştirdiğimiz içindi. Uygulamamızı doğrudan çekirdeğin içine, derleme aşamasında gömecek olsaydık, lisansını da GPL yapmamız gerekecekti. Çünkü çekirdek GPL lisansına sahiptir ve eklenen her programın da GPL veya muadili bir lisansa sahip olması gerekir. Fakat kök dosya sistemi ayrık derlenirse, uygulama özel lisanslı olabilir ve çekirdeğin lisansını ihlal etmez.

Şimdi burada cevaplanması gereken çok basit bir soru vardır. Varsayılan disk bölümlendirmesine göre

```
MLO programı,              x-loader bölümüne,
u-boot.img programı        u-boot   bölümüne,
uImage.2048 çekirkdeği de kernel   bölümüne kazınacaktır.
```

Peki kök dosya sistemini barındıran uInitrd.2048 imajı nerede oturacaktır? Eğer kök dosya sistemini çekirdeğin içine gömerek derleseydik bu soruya gerek kalmayacaktı. Çünkü çekirdeği kernel bölümüne yazdığımızda, çekirdek içine gömülü olan kök dosya sistemi de çekirdek ile birlikte kernel bölümünde oturacaktı.

uInitrd.2048 imajı birkaç yerde olabilir. Biz fs sistemine hiç dokunmadan kök dosyas sistemini çekirdekten arta kalan boş yere yerleştirdik. Sanki çekirdeğin içinde gibi görünüyor ama çekirdeğin içinde değildir.

Bu çok basit bir biçimde, çekirdek ve kök dosya sistemi alt alta eklenerek yapılabilir. Sonuçta tek bir dosya elde edilir. Elde edilen dosya kernel bölümüne tek dosyaymış gibi yazılır. Böylece çekirdek ve kök dosya sistemi, yani uImage.2048 ve uInitrd.2048 programları kernel bölümünde alt alta oturacaklardır. Birleştirme işlemi aşağıdaki başlıkta incelenecektir.

11. Çekirdek ve Initramfs'in Birleştirilmesi

Çekirdek ve kök dosya sistemi aşağıdaki gibi alt alta yapıştırılabilir.

```
$ cat uImage.2048 uInitrd.2048 > kinitrd
```

Nihai dosyaya, yani çekirdek ve kök dosya sisteminin birleşmiş haline kinitrd denilmiştir. Aşağıdaki gibi dosyaların boyutları ve kinitrd'nin boyu ls -l ile incelenebilir.

Dikkat edilirse kinitrd'nin boyu kernel bölümüne sığmalıdır. Eğer sığmaz ise, yani 4M'den büyük olursa, yeni bir mtdparts tanımı ile disk bölümü genişletilmelidir.

```
-rw-r--r-- 1 root root 2516992 Aug 25 21:41 uImage.2048
-rw-r--r-- 1 root root 1566720 Aug 25 20:56 uInitrd.2048
-rw-r--r-- 1 root root 4083712 Aug 25 21:42 kinitrd
```

Bu birleştirme de uInitrd'nin flash diskteki başlangıç adresi çok önemlidir. Öncelikle Şekil 1.5'in NAND Disc ile verilen kernel bölümü dikkatlice incelenmelidir.

x-loader bölümü her zaman diskin 0. adresinde başlar. x-loader+u-boot+u-boot-env bölümlerinin boyları bir hesap makinesinde toplanırsa 0×280,000 olduğu, aşağıdaki gibi görülebilir.

```
kernel bölümünün başlangıç adresi= (512+1,920+128)*1,024 = 2,6
                                                          = 0x2
```

uInitrd.2048'in boyunun, yukarıdaki "ls -l" çıkışından 2,516,992 veya 0×266,800 olduğu görülebilir. O halde çekirdeğin başlangıç adresine çekirdeğin boyunu eklersek,

```
ramdiskin başlangıcı= 0x280,000+ 0x266,800= 0x4E6,800
```

ramdisk başlagıç adresi olarak 0x4E6,800 değerini elde ederiz. Bu toplamayı yapmaya hakkımız vardır. Çünkü çekirdek ve uInitrd arasında hiç boşluk yoktur. cat komutu iki dosyayı tam alt alta yapıştırır.

Çekirdek veya kök dosya sistemi her güncellendiğinde bu hesabı yapmak gerekir. Tembellik gereği, bu hesabı otomatik olarak yapan, kur isimli bir betik yazılmıştır. Ayrıca u-boot için gerekli komutlar da betik tarafından üretilmektedir. Bu komutların nasıl kullanılacağı sonraki bölümlerde incelenecektir. Okuyucu şimdilik "kur" programının sonuçlarını es geçebilir.

kur programı 3 parametre alır. İlk 2 parametre çekirdek ve kök dosya sisteminin isimleridir. Birleşmiş dosyanın ismi son argüman olarak verilir. Nihai çıkışta u-boot içine gerekli bütün komutlar üretilir. Bu komutlar copy/paste ile u-boot tarafına girilebilir. Okuyucu bu betiğe gerek kalmadan yukarıdaki gibi toplam işlemini kendisi yapabilir.

```
# ./kur uImage.2048 uInitrd.2048 kinitrd

-rw-r--r-- 1 root root 2516992 Aug 25 21:41 uImage.2048
```

```
-rw-r--r-- 1 root root 1566720 Aug 25 20:56 uInitrd.2048
-rw-r--r-- 1 root root 4083712 Aug 25 21:42 kinitrd

Flash diske yazım için...

 mmc init
 mmc rescan
 fatload mmc 0:1 80000000 kinitrd
 nandecc sw
 nand erase 280000 400000
 nand write 80000000 280000 400000

Otomatik boot için...
 Eğer adresler değişmemiş ise tekrar tanım yapmaya gerek yoktu

setenv bootcmd "nand read 80000000 280000 266800;
                nand read 81600000 4E6800 17E800;
                bootm 80000000 81600000"
 saveenv
```

Şu ana kadar bord için gerekli bütün programlar hazırlanmıştır. Şimdi altın vuruş yapılacak ve elde edilen bütün programlar NAND diske yazılacak ve bord, MMC olmadan otomatik olarak NAND flash'tan açılacaktır. Mutlu sona ermek için öncelikle NAND disklerle çalışırken bazı kurallara sıkı sıkıya bağlı kalmak gerekir.

12. NAND ile Çalışmak

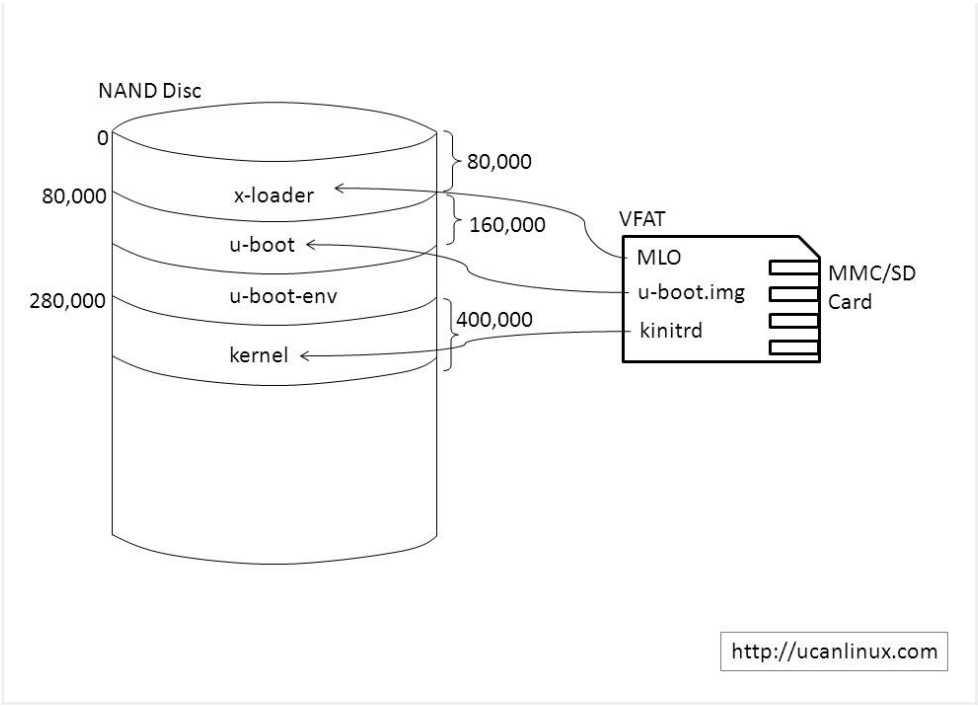
NAND diskler ile çalışırken aşağıdaki 3 kural mutlaka dikkatlice uygulanmalıdır.

1. Diske yazılan her dosya, page değerine göre hizalanmalıdır.
2. ECC değeri olan hw veya sw, doğru seçilmelidir.
3. Diske yazım yapmadan önce mutlaka "erase" işlemi yapılmalıdır.
   Erase işleminden önce mutlaka ECC değeri düzgün seçilmiş olmalıdır.

Bu kurallardan biri gözardı edilirse,
"Attempt to read non page aligned data" veya "CRC Error" gibi hatalar gelebilir.

13. Bütün Dosyaların NAND'a Yüklenmesi

Şu ana kadar üretilen bütün dosyalar MLO, u-boot.img ve kinitrd dosyaları NAND diske yüklenecektir. Yükleme işi MMC kart ile yapılacaktır. Aslında en ideal çalışma ortamı NFS yardımı ile kurulur. Bu da ayrı bir yazı konusu olacak kısmetse. Şekil 1.4'de MMC'den NAND bölümlerine yapılacak kopyalama temsil edilmektedir. Bütün sayılar 16'lık sistemde verilmiştir. Sayı önlerine 0x yazmayı unuttum.



— Şekil 1.4: MMC'deki dosyaların disk bölümlerine aktarılması.

Daha önce MLO ve u-boot.img programları MMC'ye taşınmıştır. Son elde edilen kinitrd programı da MMC'ye aşağıdaki gibi aktarılır.

```
# mount /dev/sdb1 /mnt/mmc

# ls -l /mnt/mmc
 -rwxr-xr-x 1 root root  45780 Aug 23 19:35 MLO
 -rwxr-xr-x 1 root root 335576 Aug 24 09:16 u-boot.img

# cp kinitrd /mnt/mmc

# ls -l /mnt/mmc
 -rwxr-xr-x 1 root root   45780 Aug 23 19:35 MLO
 -rwxr-xr-x 1 root root 4083712 Aug 25 21:45 kinitrd
 -rwxr-xr-x 1 root root  335576 Aug 24 09:16 u-boot.img

# umount /mnt/mmc
```

Kopyalama bittikten sonra MMC borda takılır. Bord, USER butonuna basılarak açılır. Otomatik açılışı durdurmak için "Hit any key to stop autoboot: " mesajı gelince, herhangi bir tuşa basılır ve açılış u-boot seviyesinde durdurulur. Aşağıdaki işlemlerle MMC'de bulunan MLO, u-boot ve kinitrd dosyaları flash diske yazılır.

14. MLO'nun Yüklenmesi

U-boot seviyesinde iken, MMC'nin u-boot tarafından algılanması gerekir. Bunun için "mmc rescan" komutu aşağıdaki gibi girilir.

```
UcanLinux > mmc rescan
```

Bordda tek bir MMC/SD kart okuyucu olduğudan, bu okuyucuya 0 sıra numarası verilir. Bundan sonra, MMC/SD ile ilgili bütün komutlarda "mmc 0″ ifadesi bulunacaktır. Aşağıdaki gibi fatls komutu ile MMC kartın dizin listesi alınabilir.

```
UcanLinux > fatls mmc 0
 45780     mlo
 4140160   flash.img
 335576    u-boot.img

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

U-boot sistemi, hem vfat hem de ext2 dosya sistemini destekler. Ama MLO'dan dolayı her zaman VFAT dosya sistemi kullanılmalıdır. Aşağıdaki gibi fatinfo komutu ile MMC cihazının bilgileri elde edilebilir.

```
UcanLinux > fatinfo mmc 0

Interface:  MMC
 Device 0: Vendor: Man 015041 Snr 1e576822 Rev: 9.2 Prod: S032
 Type: Removable Hard Disk
 Capacity: 29.6 MB = 0.0 GB (60800 x 512)
 Partition 1: Filesystem: FAT12 "NO NAME    "
```

Çıkıştan da görüleceği gibi yaklaşık 30 MB'lık bir MMC kart ile işimi yapmaktayım. Bu kartı eski eşyaların içinde buldum, çok da işe yaradı.

U-boot sisteminin çok zengin bir komut kümesi vardır. Aşağıdaki gibi help komutu ile her bir komuta ait özet kullanım bilgisi elde edilebilir.

```
UcanLinux > help fatload

fatload - load binary file from a dos filesystem

Usage:
 fatload <interface> <dev[:part]>  <addr> <filename> [bytes]
 - load binary file 'filename' from 'dev' on 'interface'
 to address 'addr' from dos filesystem
```

fatload komutu MMC'den bordun belleğine dosya yüklememizi sağlar. Aşağıdaki komut ile MLO programı 0×80,000,000 adresine yüklenmektedir.

```
UcanLinux > fatload mmc 0:1 80000000 MLO
 reading MLO
 45780 bytes read
```

Komutun kullanımı son derece basittir. "mmc 0″ ile birinci MMC kart temsil edilir. Zaten 1 tane kart okuyucu vardır. ":1″ ile birinci VFAT bölümü temsil edilir. 80000000 ile yüklenecek bellek adresi verilir. Bu adresin ne olduğu pek önemli değildir. Alışkanlık gereği 0×80,000,000 verilir. Bu adres RAM'ın başlangıç adresidir. Daha ilerideki adresler de verilebilir. bdinfo komutu ile adresler incelenebilir. Son argüman olarak dosya ismi verilir. İsmi verilen bu dosya 0×80,000,000 adresinden sonra RAM belleğe yüklenir.

Bir sonraki adımda bellekte bulunan bu dosya NAND diskin x-loader ile verilen bölümüne yazılacaktır. Fakat yazım yapmadan evvel hata bulmaya ve düzeltmeye yarayan ECC tipi verilmelidir. x-loader bölümünde oturacak olan MLO programını ROM'daki boot yükleyicisi okuyacaktır. ROM'daki boot yükleyicisi "hw ecc" desteği vermektedir. Bundan dolayı NAND'a yazım yapılmadan evvel aşağıdaki gibi hw ecc seçilmelidir.

```
UcanLinux > nandecc hw
 HW ECC selected
```

Bu sayede, U-boot programı aşağıdaki "nand erase" ve "nand write" komutlarını yürütürken "hw ecc" ile yazım yapacaktır. Sistem açılırken Rom boot yükleyicisi de "hw ecc" yöntemine göre hata analizi yapacağından sorun çıkmayacaktır. hw ve sw sıra ile "hardware" ve "software" kelimelerinden gelir.

Daha önce bahsettiğimiz çalışma prensiplerinden bir tanesinde, yazım yapılmadan evvel, o bölgenin silinmesi ile ilgiliydi. Klasik bir IDE veya SCSI diskte, bir yere bilgi yazılacağı zaman eskisi otomatik olarak ezilir. Fakat flash disklerde yazım işi eski bitlerin hepsini değiştirmez. Bitlerin tamamen değişmesi için önce mutlaka erase işlemi yapılmalıdır. NAND diskte silme işlemi aşağıda verildiği gibi "nand erase" komutu ile yapılır.

```
UcanLinux > nand erase 0 80000

NAND erase: device 0 offset 0x0, size 0x80000
 Erasing at 0x60000 -- 100% complete.
 OK
```

Bizler x-loader bölümünü tamamen silmek istiyoruz. Bunun için ilgili bölümün başlangıç adresi ve uzunluğunu vermek yeterlidir. Şekil 1.4 de gösterildiği gibi x-loader bölümü 0. konumdan başlar ve toplam 0×80,000 bayt yer kaplar. Silme işi bittikten sonra aşağıdaki gibi diske yazım yapılır.

```
UcanLinux > nand write 80000000 0 80000

NAND write: device 0 offset 0x0, size 0x80000
 524288 bytes written: OK
```

“nand write” komutunun parametreleri kolayca tahmin edilebilir. Birinci parametre ile bellek adresi verilir. 0×80,000,000 ile verilen bellek adresini daha önce “nand read” ile kullanmıştık. Yazım işi tamamlandıktan sonra bu adresin bir kıymeti olmayacaktır. Sonraki parametre ise x-loader bölümünün diskteki başlangıç adresi ve bölümün boyudur. Boy olarak her zaman 0×80,000 kullandık. Aslında buna gerek yoktur. Doğrudan MLO’nun boyunu versek de olurdu.

Görüleceği gibi çok basit bir mantıkla, diske yüklenecek dosya önce bellekte bir adrese alınır, sonra da bellekten diske kazınır. u-boot.img ve kinitrd programları da aşağıdaki gibi diske kazınır. Mantık tamamen aynı olduğu için açıklama verilmeyecektir.

15. U-Boot’un NAND’a Aktarılması

```
UcanLinux > fatload mmc 0:1 80000000 u-boot.img
 reading u-boot.img

335576 bytes read

UcanLinux > nandecc hw
 HW ECC selected

UcanLinux > nand erase 80000 160000

NAND erase: device 0 offset 0x80000, size 0x160000
 Erasing at 0x1c0000 -- 100% complete.
 OK

UcanLinux > nand write 80000000 80000 160000

NAND write: device 0 offset 0x80000, size 0x160000
 1441792 bytes written: OK
```

16. kinitrd’nin NAND’a Aktarılması

```
UcanLinux > fatload mmc 0:1 80000000 kinitrd
 reading kinitrd

4083712 bytes read

UcanLinux > nandecc sw
 SW ECC selected

UcanLinux > nand erase 280000 400000

NAND erase: device 0 offset 0x280000, size 0x400000
 Erasing at 0x660000 -- 100% complete.
 OK

UcanLinux > nand write 80000000 280000 400000

NAND write: device 0 offset 0x280000, size 0x400000
 4194304 bytes written: OK
```

Bu adımdan sonra, daha önce “kur” betiği ile üretilen u-boot komutları artık anlam kazanacaktır. “kur” programı dosyaları diske yazabilmek için gerekli u-boot komutlarının parameterlerini hesaplamakta ve raporlamaktadır. O raporda açıklanmayan sadece bootcmd parametresi kalmıştır.

U-boot sistemi bootcmd içine yazılan her komutu gözü kapalı işletir. Eğer çekirdeği ve kök dosya sistemini yükleyecek ve boot edecek komutları bootcmd içine yazarsak, bordun otomatik olarak açılmasını sağlayabiliriz.

17. Otomatik Açılış

Bu adıma gelindiğinde, NAND diskte gerekli bütün dosyalar mevcut olacaktır. Fakat otomatik açılış için u-boot’a yeni bir çevre değişkeni girmek ve onu u-boot-env bölümü içine kaydetmek gerekir. Bunun için MMC kart borddan çıkarılır, artık işimiz kalmamıştır. RESET düğmesine basılarak bord tekrar başlatılır ve herhangi bir tuşa basılarak açılışın u-boot aşamasında kalması sağlanır. Daha sonra aşağıdaki bootcmd (boot command) komutu girilir ve saveenv komutu ile diske kalıcı olarak kaydedilir.

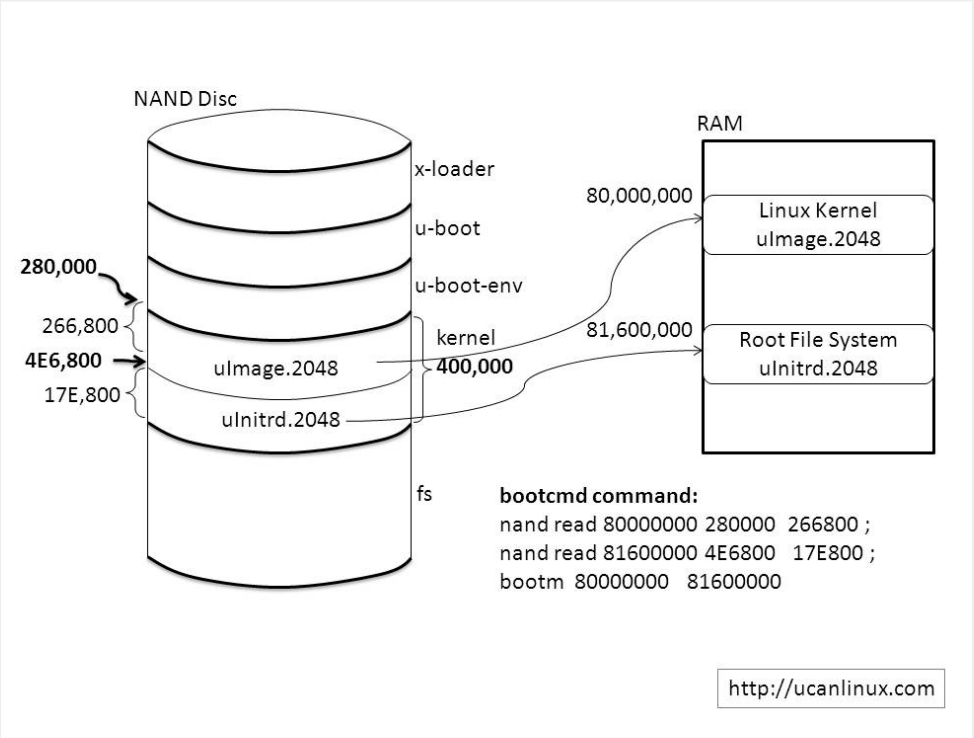
```
UcanLinux > setenv bootcmd "nand read 80000000 280000 266800 ;
                            nand read 81600000 4E6800 17E800 ;
                            bootm 80000000 81600000"

UcanLinux > saveenv

Saving Environment to NAND...
 Erasing Nand...
```

```
Erasing at 0x260000 -- 100% complete.
Writing to Nand... done
```

Yukarıdaki bootcmd komutu, okumayı kolaylaştırmak için parçalı yazılmıştır. Gerçekte tek satır olmalıdır. Buradaki bootcmd komutu Şekil 1.5′de temsil edilmiştir.



— Şekil 1.5: Otomatik açılış.

Şekil 1.5 bütün bootcmd komutunu açıkça resmetmektedir.

“nand read 80000000 280000 266800″ komutu, 0×280,000 disk adresinden itibaren, yani kernel bölümün başından itibaren 0×266,800 bayt okuma yapar. 0×266,800 bayt aslında uInitrd.2048 dosyasının boyudur. Bu komut ile, NAND diskte bulunan çekridek 0×80,000,000 adresinden itibaren belleğe okunacaktır.

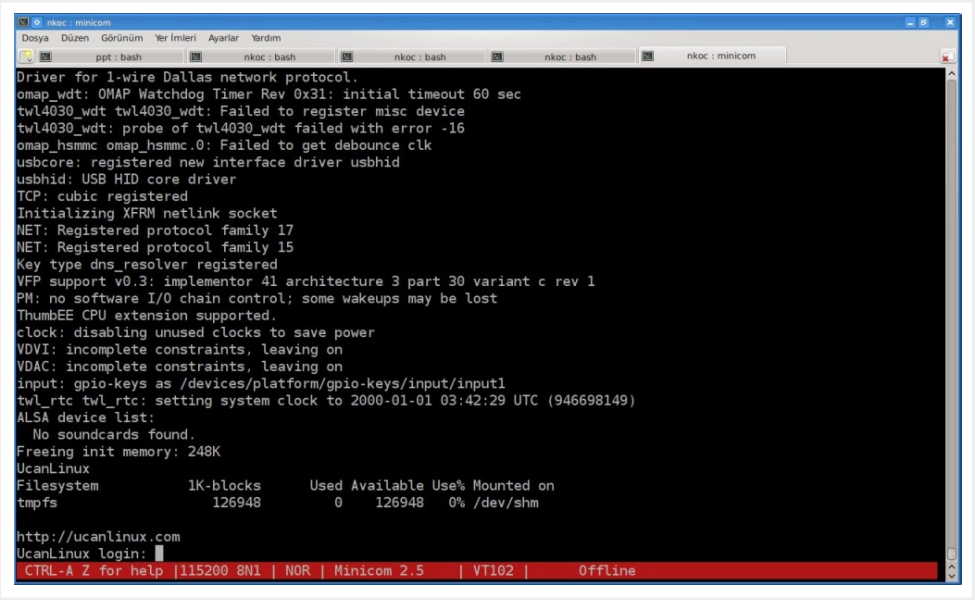
“nand read 81600000 4E6800 17E800″ komutu da benzer bir iş yapar. Bu komut kök dosya sistemini, yani uInitrd.2048 dosyasını 0×81,600,000 adresine okur. Bu adresin bir önemi yoktur. Yeter ki veriler birbirlerini ezmesinler. 0x4E6,800 adresi daha önce hesaplanmıştı. Bu adres hemen çekirdeğin altında bulunan kök dosya sisteminin başlangıç adresi idi. 0x17E,800 değeri ise uInitrd.2048 dosyasının, yani kök dosya sisteminin boyudur.

bootcmd içine birden fazla komut girildiğinde, komutlar birbirlerinden “;” ile ayrılmalıdırlar. Son olarak “bootm 80000000 81600000″ komutu işler. “bootm” kelimesi

“boot from memory” den gelir.İsminden de anlaşılacağı gibi 80000000 adresine yürütmeyi verir. Burada da çekirdek olduğu için bord açılmaya başlar. bootm komutu aynı zamanda 81600000 adresini, çekirdeğe ramdisk_addr parametresi olarak geçirir. Çekirdek de açıldıktan sonra bu adreste bir ramdisk olduğunu anlar ve mount ederek açılış işlemlerini devam ettirir.

Dikkat edilirse 81600000 adresini biz tamamen keyfi olarak verdik. Bu adres daha sonra çekirdeğe parametre olarak aktarılacaktır. İşte bu sebepten dolayı kök dosya sistemi mkimage ile u-boot imajı haline getirilirken -a ve -e ile ilgili adresler 0 verilmişti.

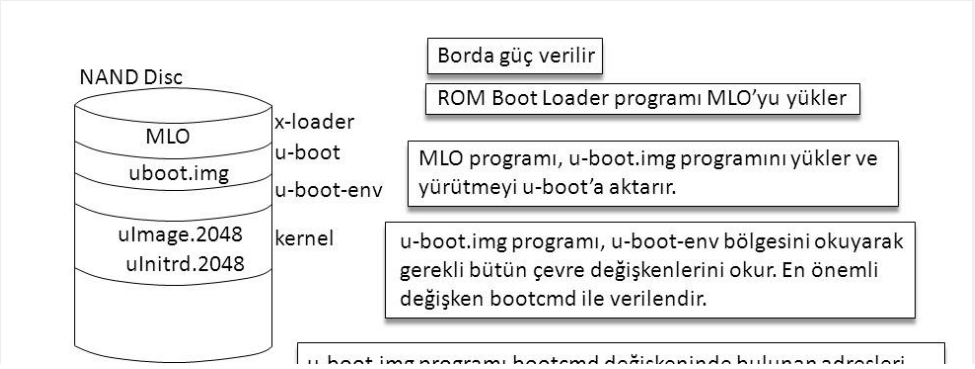
Sistemimiz otomatik olarak açılacak ve aşağıdaki gibi çok güzel, kapkara bir ekran gelecektir. root kullanıcısının şifresi yine root’tur.

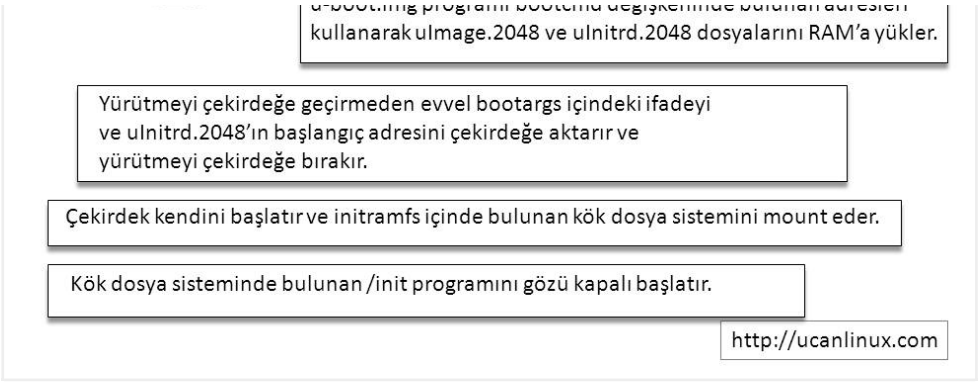


— Linux’un güzel yüzü.

18. Login’e Kadar Geçilen Aşamalar

Çekirdek kök dosya sistemini mount ettikten hemen sonra gözü kapalı bir biçimde /init programını başlatır. /init’e gelen kadar geçilen aşamalar Şekil 1.6’da özetlenmiştir.





— Şekil 1.6: /init'e gelen kadar geçilen aşamalar.

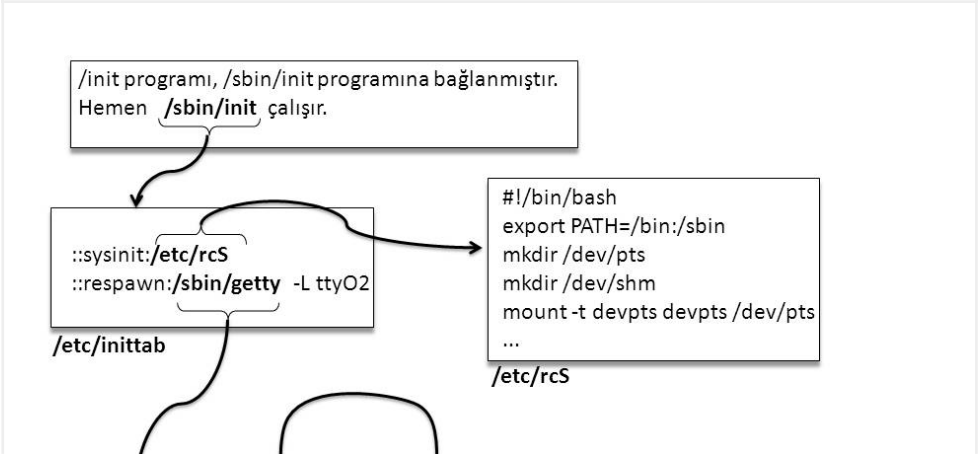
Buraya kadar geçilen aşamalar nerede ise bütün gömülü sistemlerde aynıdır. Kök dosya sistemini kurarken /init programına /sbin/init'i bağlamıştık. Böylece açılışın /sbin/init programı tarafından yönlendirilmesini sağlamıştık. Burada pek çok almaşık çözüm denenebilir.

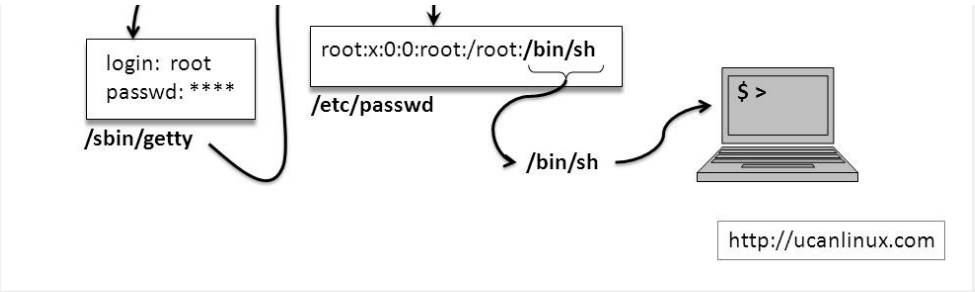
Örneğin /init programı /bin/sh programına bağlanır. Böylece Linux açılır açılmaz kabuk programı gelir.

Uygulama programının adı init yapılır ve doğrudan köke atılır. Çekirdek /init programını gözü kapalı işleyeceği için, uygulamamız sistem açılır açılmaz çalışacaktır.

Peki uygulama son bulursa çekirdek ne yapacaktır? Initramfs esas kök dosya sistemi değildir. Initramfs'in çalışması bitince çekirdek initramfs ile ilgili herşeyi bellekten siler ve root parametresi ile verilen diski mount etmeye çalışır. Fakat biz henüz böyle bir çalışma yapmadığımız için Linux çökecektir. Initramfs'den gerçek kök dosya sistemine geçiş ayrı bir inceleme konusudur. Fakat şu an için bu geçişin hiç bir önemi yoktur. Çünkü Initramfs sistemi bizim bütün ihtiyaçlarımızı karşılamaktadır.

/sbin/init programını çalıştığında, Şekil 1.7'de adım adım verildiği gibi, login promptu ve kabuk elde edilecektir. Çok açık olduğu için Şekil 1.7 ayrıca incelenmeyecektir.





— Şekil 1.7: Kabuğa gelene kadar geçilen aşamalar.

19. Açılış Betiği

Daha önce gömülü bir sistemin 4 ayak üzerine oturduğundan bahsetmiştik. Bunlar sıra ile,

1. Boot yükleyicileri
2. Çekirdek
3. Kök dosya sistemi
4. Açılış betikleri

başlıklarında özetlenebilir. Şu anda kadar 1. başlık içim, MLO ve U-Boot sistemleri kullanıldı. Çekirdek için, Linux 3.5.2 çekirdeği üzerinde çalışıldı. Kök dosya sistemi olarak initramfs tercih edildi. Bu çalışmaların tamamı bu adıma kadar yapıldı. Şimdi 4. başlıkta verilen açılış betikleri üzerinde kısaca durulacaktır.

Standard bir Linux dağıtımında /etc/rc.d veya /etc/init.d gibi dizinler altında onlarca açılış betiği bulunur. Bu açılış betiklerini, en üst tarafta, genelde /sbin/init programı tetikler. Fakat gömülü sistemlerde bu kadar karmaşık açılış betiklerine gerek yoktur. Bazı gömülü sistemlerde standard bir dağıtımda bulunan açılış betikleri nerede ise olduğu gibi kullanılmaktadır. Bizce bu şımarıklıktır.

Gömülü sistem, yapısı itibari ile yapacağı işi ve donanımı hemen hemen çok katı bir biçimde önceden tespit edilmiş bir sistemdir. /etc/init.d/rc.network betiği için, “cihazda eternet var mı, varsa ağ tanımlarını yapayım” diyerek yola çıkılmamalıdır. Sistemde ağ özelliği ya vardır ya yoktur. Çünkü donanım her hali ile açık seçik bellidir. Tabii ki çok özel durumlardan bahsetmiyoruz.

Bu kadar laftan sonra, tek bir açılış betiği yazılacaktır. Adet üzerine bu betiğe rcS ismi verilmiştir. rcS ismi, “run command script” veya “run command single” kelimelerinden geliyor olabilir, emin değilim.

Şekil 1.7'de gösterildiği gibi, /etc/rcS betiği /sbin/init tarafından başlatılır. /sbin/init ayağa kalkar kalkmaz /etc/inittab dosyasını okur ve oradaki her satırları etiket değelerine göre tek tek işler. Fakat satırları işlemeden önce, ilk olarak sysinit etiketli

satırı arar. Eğer bulursa bu satırda tanımlı programı çalıştırır. Bu satıra /etc/rcS betiği yazılmıştır. Böylece /sbin/init programı ayağa kalkar kalmaz hemen /etc/rcS betiğini yürütecektir.

Örnek açılış betiği aşağıda verilmiştir.

```
 1 #!/bin/sh -x
 2 export PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin

 3 mount -t sysfs  sysfs  /sys

 4 mkdir /dev/pts
 5 mkdir /dev/shm
 6 mdev -s

 7 mount -t proc   proc   /proc
 8 mount -t devpts devpts /dev/pts
 9 mount -t tmpfs  tmpfs  /dev/shm

10 hostname $SISTEM_ADI

11 syslogd -C
12 klogd

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

15 telnetd
16 df
```

Öncelikle burada yazılan her komutun busybox içinde olması gerektiğini hatırlatalım. Nihai sisteme geçildiğinde, busybox'a sadece rcS içindeki komutları eklemek yeterli olacaktır. Ama test sisteminde bu hiç tavsiye edilmez.

Açılış betiğinin en tepesine, 1. satırda verildiği gibi -x ile debug seçeneğini eklemek çok faydalıdır. Böylece her komut işlemeden önce, komutun kendisi ekrana yazılacaktır. Başlangıçta hata bulmak için çok faydalıdır. -x ile debug açılır, +x ile kapatılır.

3. satırda sys dosya sistemi bağlanır. Bu dosya sistemi de /proc gibidir ama daha çok donanıma yönelik bilgiler barındırır. Bu dosya sistemi mutlaka mdev komutundan önce bağlanmalıdır.

4 ve 5. satırda /dev dizininin içinde olması gereken 2 dizin açılmıştır.Aslında buna gerek yoktur. Dosya sistemi kurulurken mkdir ile de yapılabilirdi.

6. satırda verilen mdev komutu, /sys dosya sistemini tarar ve tanımlı bütün cihazlar için /dev dizini altında düğümleri yaratır. mdev komutundan önce /dev dizininin içi boştur. mdev'den sonra dizin içinde cihazların isimleri ve diğer bilgiler bulunur. Eğer mdev komutundan önce cihaz ismi gerektiren bir komut girilecekse, örneğin /dev/ttyUSB0 ile ilgili bir iş yapılacaksa, ilgili cihaz ismi açıkça mknod komutu ile yaratılabilir.

7. adımda, daha önce bahsettiğimiz /proc dosya sistemi bağlanır.

8. adımda pty sistemi bağlanır. Dışarıdan telnet veya ssh gibi programlarla gömülü cihaza bağlanmak için pty cihazlarına gerek vardır. /dev/pty dosya sistemi bu cihazlara ihtiyacı olan programlara cihaz ismi temin eder.

9. adımda /dev/shm dosya sistemi bağlanır. Burası geçici disk olarak kullanılabilir. Dosya sisteminin varsayılan kapasitesi, fiziksel belleğin yani RAM belleğin tam yarısıdır. Ama bu bellek hiç bir zaman ramdisk gibi çalınmaz. Dosya sistemi ihtiyaç oldukça büyür, ihtiyaç oldukça küçülür.

/dev/shm dosyasındaki bütün bilgiler, cihaz kapanınca gider. Bir tür ramdisk gibi düşünülebilir. Ama çok daha yeteneklidir. Kök dosya sistemini read/only bağlayan sistemler, genelde /tmp ve /var dizinlerini, yani yazılabilir dizinleri buraya yönlendirilirler.

/dev/shm cihazı normal bir disk gibi kullanılabilir. Örneğin, $ vi /dev/shm/dene.c ile test kodu burada yazılabilir. Ya da $ mkdir /dev/shm/falan gibi dizin yaratılabilir.

10. adımda, hostname komutu ile makine ismi tanıtılır. Bizler SISTEM_ADI değişkenini vermeyi unutmuşuz!

11. adımda sistem kayıtçısı başlatılır. -C ile kayıtçı, mesajları çevrimsel saklar. Diğer bir deyişle mesajlar belirli bir boydan daha fazla yer kaplayamaz. Böylece zaman içinde diskin veya belleğin mesajlarla dolması engellenir.

12. adımda, klogd programı çekirdekten gelen mesajları syslogd'ye yönlendirir. Böylece çekirdek mesajlarının syslogd tarafından yakalanması sağlanır.

13 ve 1.4 adımlarda localhost tanımı yapılır.

15. adımda telnet sunucusu başlatılır ki, borda ağ üzerinden erişelim. Bizler USB/Ethernet çeviricisi ile borda ağ üzerinden bağlandık. Bu satır gözardı edilebilir. Çünkü bu yazıda ne u-boot ne de linux tarafında ağ özellikleri üzerinde hiç durmadık. Belki başka bir yazının konusu olabilir.

16. adımda df ile mount edilmiş dosyalar listelenir. Bizler testlerimiz NFS üzerinden yaptığımız için bu satır, bilgi babından gerekli idi. Şu anki uygulamada gözardı edilebilir.

20. Örnek Programın İşletilmesi ve strace

Bu yazının en başında ilk.c isimli bir program yazmış, çapraz olarak derlemiş ve derleme sonunda ilk.dynamic isimli bir dosya elde etmiştik. Kök dosya sistemi kurulurken bu dosyayı /usr/bin altına atmıştık. Bu dosya aşağıdaki gibi işletilebilir.

```
root@UcanLinux:/usr/bin # ./ilk.dynamic
ARM işlemci için ilk program.
Merhaba ARM.
```

Ayrıca her test sisteminde strace programı mutlaka bulunmalıdır. Biz de strace programını kök dosya sistemine ekledik. Bu program Linux sistem çağrılarını, yani çekirdek tarafından yapılan çağrıları listeler. Gömülü sistemde çalışmayan bir program, *$ strace program_adı* ile işeltilirse, eksik kütüphane gibi bazı hatalar çok kolaylıkla bulunabilir. Örnek programınız aşağıdaki gibi izlenebilir.

```
$ strace ilk.dynamic
```

strace programı aynı zamanda halen çalışmakta olan programları izlemek için de kullanılabilir. Bunun için -p seçeneği ile pid numarasını vermek yeterlidir. Örneğin telnet sunucusunun pid değeri 528 ise,

```
$ strace -p 528
```

komutu ile arka planda çalışan bir program incelenebilir. Bu tür incelemeler özellikle config dosyaları veya yetkilerle ilgili hataları bulmak için çok faydalıdır.

21. Bazı Çekirdek Mesajlarının İncelenmesi

Login geldikten sonra root/root ile sisteme giriş yapılır. Bir gömülü sistem ilk defa açılıyorsa, çekirdek mesajları çok dikkatlice incelenmelidir. Bu inceleme sırasında gereksiz bir sürücü veya can sıkan uyarılar aranmalıdır. Aşağıda dmesg çıkışının bazı önemli satırları verilmiştir.

Aşağıdaki çıkış çekirdeğin hangi tool chain ve hangi sürüm ile ne zaman derlendiğini gösterir.

```
Linux version 3.5.2-ucan-linux (root@nkoc_ev)
```

```
        (gcc version 4.6.3 (Sourcery CodeBench Lite 2012.03-57)
        Sat Aug 25 14:00:01 EEST 2012
```

Aşağıdaki çıkış bord hakkında bilgi verir.

```
CPU: ARMv7 Processor [411fc083] revision 3 (ARMv7), cr=10c53c7
Machine: OMAP3 Beagle Board
```

Aşağıdaki çıkış, u-boot tarafından çekirdeğe aktarılan parametreleri gösterir. Çekirdeğe aktarılacak bütün parametreler u-boot tarafında bootargs değişkeninde saklanır. Aşağıdaki çıkışta dikkat edilirise mtdparts değişkeni de vardır. Çekirdek bu değişken sayesinde diskin mantıksal olarak nasıl bölündüğünü anlar. Bazı parametreler istenirse derleme sırasında da verilebilir. Çekideğin bu özelliği ile şimdiye kadar ilgilenmedik.

```
Kernel command line: console=ttyO2,115200n8
                     root=/dev/ram
                     mtdparts=omap2-nand.0:512k(x-loader),
                              1920k(u-boot),128k(u-boot-env),
                              4m(kernel),100m(fs1),50m(fs2),-(
```

Fiziksel bellek hakkındaki bilgi aşağıdaki gibi raporlanır.

```
Memory: 255MB = 255MB total
```

Üzerinde hiç durmadık, fakat devtmpfs isimli çok önemli bir cihaz dosya sistemi mevcuttur. Bu dosya sistemi çekirdek ayağa kalktıktan sonra kurulur ama initramfs seviyesinde mount edilmez. Aşağıda bu dosya sisteminin kurulduğu rapor edilir. Bu dosya sistemi initramfs bittiken sonra, eğer istenirse otomatik olarak çekirdek tarafından bağlanır.

```
devtmpfs: initialized
```

Bordun sürümü aşağıdaki gibi raporlanır.

```
OMAP3 Beagle Rev: C1/C2/C3
```

Aşağıdaki mesaj ile initramfs'in kullanıma alındığı görülebilir. Initramfs ve initrd ayrı tekniklerdir. Çekirdek imajın tipine bakarak initramfs mi yoksa initrd mi olduğuna karar verir. initrd sistemi artık çok eski kaldığından kullanılmamalıdır.

```
Trying to unpack rootfs image as initramfs..
```

Başka bir yazıda jffs2'nin kullanılmasından bahsedeceğiz. Çekirdeğe jffs2 desteği verdiğimizden dolayı aşağıdaki gibi bir mesaj alınır.

```
jffs2: version 2.2. (NAND) (SUMMARY)  Â© 2001-2006 Red Hat, In
```

Çekirdek mesajlarının yazıldığı cihaza konsol denir. Genelde konsollarda line disiplini bulunmaz. Bundan dolayı login işlemlerinin konsollardan yapılması pek tavsiye edilmez. Linux çekirdeği mesajları nereye basacağını bilemez. Çekirdek parametresi ile konsol ayrıca tanıtılır. Beaglebordun konsol cihazının adı ttyO2'dir. Genelde her bordda bu tanım farklıdır. El kitaplarından ya da örnek dağıtımlardan konsol tanımı elde edilmelidir. Aşağıdaki çıkışta konsolun başlatıldığı raporlanmaktadır.

```
console [ttyO2] enabled
```

Aşağıda NAND cihazın fiziksel özellikleri ve bölümleri ile ilgili bilgiler raporlanmıştır.

```
NAND device: Manufacturer ID: 0x2c, Chip ID: 0xba
             (Micron NAND 256MiB 1,8V 16-bit),
             page size: 2048, OOB size: 64

7 cmdlinepart partitions found on MTD device omap2-nand.0

Creating 7 MTD partitions on "omap2-nand.0":

0x000000000000-0x000000080000 : "x-loader"
0x000000080000-0x000000260000 : "u-boot"
0x000000260000-0x000000280000 : "u-boot-env"
0x000000280000-0x000000680000 : "kernel"
0x000000680000-0x000006a80000 : "fs1"
0x000006a80000-0x000009c80000 : "fs2"
0x000009c80000-0x000010000000 : "fs3"
```

22. /proc Dosya Sisteminin İncelenmesi

/proc dosya sistemi prosesler ve bazı donanımlar hakkında bilgi tutan, pusedo bir dosya sistemidir. Bu dosya sisteminin esas amacı user space için, kernel space tarafından bilgi sunulmasıdır. Ya da user space tarafından, kernel space tarafına geçmeden, oradaki bilgilere ulaşmaktır. Aşağıdaki bazı dosyaların incelenmesi yararlı olacaktır.

cpuinfo işlemci hakkında bilgi verir.

```
root@UcanLinux:/proc # cat cpuinfo

Processor       : ARMv7 Processor rev 3 (v7l)
BogoMIPS        : 575.59
Features        : swp half thumb fastmult vfp edsp thumbee neo
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x1
CPU part        : 0xc08
CPU revision    : 3

Hardware        : OMAP3 Beagle Board
Revision        : 0020
Serial          : 0000000000000000
```

cmdline, çekirdek komut satırını verir. Bu çok faydalı bir dosyadır. User space'de çalışan bir program, bu dosyayı sıradan bir dosya gibi açarak, çekirdeğin hangi parametreler ile açıldığını bulabilir. Özellikle ağ destekli açılışlarda bazı ağ bilgileri bu dosyadan temin edilir. Kolay okunması için yeni satırlar eklenmiştir.

```
root@UcanLinux:/proc # cat cmdline
console=ttyO2,115200n8
root=/dev/ram
mtdparts=omap2-nand.0:512k(x-loader),1920k(u-boot),128k(u-boot
         4m(kernel),100m(fs1),50m(fs2),-(fs3)
```

/proc/**config.gz** dosyası, çekirdek derlemesi sırasında kurulan .config dosyasının sıkıştırılmış halidir. Bu dosyanın nihai sistemde bulunmasına hiç gerek yoktur. Çünkü çok yer kaplar. config.gz dosyası iki türlü kullanılabilir.

Birinci kullanım tarzında, $ *cd /tmp ; gunzip /proc/config.gz* şeklinde dosya açılır ve incelenir. Bu dosya aynı zamanda doğrudan çekirdek kaynak kodunun hemen kök dizinine .config ismi ile kopyalanarak çekirdek derlemesi için kullanılabilir.

İkinci kullanmı tarzında doğrudan zcat komutu ile dosya içinde arama yapılabilir. Örneğin

$ zcat /proc/config.gz | grep LOOP

komutu girilerek çekirdeğin LOOP cihaz desteği ile derlenip derlenmediği test edilebilir. zcat komutu, gunzip ve cat komutlarından oluşur. Bizler busybox tarafına,

“nasıl olsa gerek olmaz” diye zcat komutunu eklememişiz. Bu yüzden örnek çıkış veremedik.

filesystems dosyası içinde çekirdeğin destek verdiği bütün dosya sistemlerinin isimleri bulunur.

devices dosyası içinde, işletim sisteminde bulunan bütün karakter ve blok cihazların majör numaraları bulunur.

mounts dosyası içinde, bağlı bütün dosya sistemlerinin özellikleri saklıdır. /etc/mtab dosyası doğrudan bu dosyaya bağlanmıştır. Herhangi bir dosya sistemi mount/umount yapıldığında /proc/mounts dosyasına bir satır eklenir veya silinir. Eski Linux uygulamalarında /etc/mtab ayrı bir dosya idi. Artık /etc/mtab dosyası doğrudan /proc/mounts dosyasına bağlıdır.

mtd dosyası içinde flash disk bölümlerinin bilgisi bulunur. Örnek dosya aşağıda verilmiştir. Linux açılırken bu bilgileri aynı zamanda konsola yazmaktadır. Bütün sayıların 16′lık sistemde olduğunu hatırlatalım. Onlar da sayıların başına 0x eklemeyi unutmuşlar

```
root@UcanLinux:/proc # cat mtd

dev:    size   erasesize  name
mtd0: 00080000 00020000 "x-loader"
mtd1: 001e0000 00020000 "u-boot"
mtd2: 00020000 00020000 "u-boot-env"
mtd3: 00400000 00020000 "kernel"
mtd4: 06400000 00020000 "fs1"
mtd5: 03200000 00020000 "fs2"
mtd6: 06380000 00020000 "fs3"
```

Yukarıdaki çıkışta, cihaz ismi olarak mtd0, mtd1, … kullanılmıştır. Örneğin fs1′e erişmek için /dev/mtd4 kullanılacaktır. /dev/mtdX ismi ile, NAND diske karakter cihaz olarak erişilir. Aynı disk bölümüne blok olarak erişmek için /dev/mtdblockX cihaz adı kullanılır. NAND disk bölümlerinin blok adlarına ulaşmak için /proc/partitions dosyası kullanılır.

partitions dosyası NAND disklerin blok cihaz cinsinden bilgilerini sağlar. Örnek çıkış aşağıda verilmiştir. Bu blok adları, özellikle dosya sistemi kurarken veya dosya sistemine blok erişimi yaparken gerekli olacaktır.

```
root@UcanLinux:/proc # cat partitions
```

```
major minor  #blocks  name

  31        0        512 mtdblock0
  31        1       1920 mtdblock1
  31        2        128 mtdblock2
  31        3       4096 mtdblock3
  31        4     102400 mtdblock4
  31        5      51200 mtdblock5
  31        6     101888 mtdblock6
```

uptime dosyası içinde iki adet sayı bulunur. Birinci sayı, işletim sisteminin açıldığından beri geçen toplam süreyi gösterir. İkinci sayı ise idle durumda beklenilen toplam süredir. Süreler saniye cinsindedir. Örnek çıkış aşağıda verilmiştir.

```
root@UcanLinux:/proc # cat uptime
 2806.15 2804.28
```

version dosyası içinde çekirdek ve derleme ile ilgili bilgiler bulunur. Örnek çıkış aşağıda verilmiştir.

```
root@UcanLinux:/proc # cat version

Linux version 3.5.2-ucan-linux (root@nkoc_ev)
            (gcc version 4.6.3(Sourcery CodeBench Lite 2012.03
            #2 Sat Aug 25 14:00:01 EEST 2012
```

/proc sistemi Linux işletim sistemine inanılmaz bir esneklik kazandırmıştır. Burada sadece birkaç dosyadan bahsedilmiştir. İlerleyen yazılarda yeri geldikçe diğer özelliklerden bahsedilecektir.

/proc sistemi esas itibari ile proseseler hakkında bilgi tutar. Her prosesin PID numarası, /proc dizini altında bir dizine karşı gelir.

23. Özet

Tool cahin aşağıdaki gibi kurulur.

```
arm-2012.03-57-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz
paketi mentor.com adresinden indirilir.

$ cd /cross
$ tar jxvf arm-2012.03-57-arm-none-linux-gnueabi-i686-pc-linux
$ ln -s arm-2012.03 arm-none-linux-gnueabi
```

```
export PATH=$PATH:/cross/arm-none-linux-gnueabi/bin
```

Bağımlılıklar aşağıdaki gibi test edilebilir.

```
$ arm-none-linux-gnueabi-objdump -x ilk.dynamic | grep NEEDED
```

U-Boot aşağıdaki gibi indirilip derlenir. MLO, u-boot.img ve mkimage elde edilir. MLO ve u-boot.img dosyaları MMC'ye aktarılır.

```
$ wget ftp://ftp.denx.de/pub/u-boot/u-boot-2012.07.tar.bz2
$ tar jxvf u-boot-2012.07.tar.bz2
$ cd u-boot-2012.07

$ vi include/configs/omap3_beagle.h

$ make CROSS_COMPILE=arm-none-linux-gnueabi- omap3_beagle_conf
$ make CROSS_COMPILE=arm-none-linux-gnueabi-
# cp tools/mkimage /usr/local/bin

# mkdir /mnt/mmc
# mount /dev/sdb1  /mnt/mmc
# cp    MLO        /mnt/mmc
# cp    u-boot.img /mnt/mmc
# umount /mnt/mmc
```

Linux çekirdeği aşağıdaki gibi derlenir.

```
# tar jxvf /tmp/ftp/linux-3.5.2.tar.bz2
# cd linux-3.5.2
# export CROSS=arm-none-linux-gnueabi-
# make ARCH=arm CROSS_COMPILE=$CROSS omap2plus_defconfig

# make ARCH=arm CROSS_COMPILE=$CROSS menuconfig
# make ARCH=arm CROSS_COMPILE=$CROSS uImage -j2
```

Busybox aşağıdaki gibi indirilir ve derlenir. Kök dosya sistemi RootFS altına kurulur.

```
$ wget http://busybox.net/downloads/busybox-1.20.2.tar.bz2
$ tar jxvf busybox-1.20.2.tar.bz2
$ cd busybox-1.20.2
$ make menuconfig
```

```
# make -j2
# make install
```

Initramfs destekli kök dosya sistemi aşağıdaki gibi kurulur.

```
# cd RootFS/
# find . | cpio -H newc -o | gzip -v -9 > ../initramfs.cpio.gz
```

Çekirdek ve kök dosya sistemi aşağıdaki gibi hizalanır.

```
# dd if=uInitrd of=uInitrd.2048 bs=2048 conv=sync
# dd if=uImage  of=uImage.2048  bs=2048 conv=sync
```

mtdparts tanımı aşağıdaki gibi yapılabilir.

```
UcanLinux > setenv mtdparts "mtdparts=omap2-nand.0:512k(x-load
             1920k(u-boot),128k(u-boot-env),4m(kernel),
             100m(fs1),50m(fs2),-(fs3) "
```

Çekirdek ve kök dosya sistemi aşağıdaki gibi birleştirilip MMC'ye aktarılabilir.

```
$ cat uImage.2048 uInitrd.2048 > kinitrd
# mount /dev/sdb1 /mnt/mmc
# cp kinitrd /mnt/mmc
# umount /mnt/mmc
```

MLO aşağıdaki gibi diske kazınabilir.

```
UcanLinux > mmc rescan
UcanLinux > fatload mmc 0:1 80000000 MLO
UcanLinux > nandecc hw
UcanLinux > nand erase 0 80000
UcanLinux > nand write 80000000 0 80000
```

u-boot.img aşağıdaki gibi diske kazınabilir.

```
UcanLinux > fatload mmc 0:1 80000000 u-boot.img
UcanLinux > nandecc hw
UcanLinux > nand erase 80000 160000
UcanLinux > nand write 80000000 80000 160000
```

kinitrd aşağıdaki gibi diske yazılabilir.

```
UcanLinux > fatload mmc 0:1 80000000 kinitrd
UcanLinux > nandecc sw
UcanLinux > nand erase 280000 400000
UcanLinux > nand write 80000000 280000 400000
```

Otomatik başlatma değişkeni aşağıdaki gibi tanımlanabilir.

```
UcanLinux > setenv bootcmd "nand read 80000000 280000 266800 ;
                            nand read 81600000 4E6800 17E800 ;
                            bootm 80000000 81600000"
```

24. Çalışma Paketi

Bu çalışmada elde edilen bazı dosyalar buradan indirilebilir. Özellikle config dosyaları yazıyı okurken yapılacak denemelerde vakit kazandıracaktır.

25. İletişim

Yazara **nazim** at **ucanlinux.com** adresinden ulaşılabilir. Ya da bu yazı doğrudan blog'dan okunuyursa blogun altına bulunan mesaj kutusu kullanılabilir. Bu kutu belirli bir süre açık kalacaktır.

26. Güncellemeler

Bu yazının en güncel hali her zaman http://ucanlinux.com adresinin blog sekmesinden elde edilebilir. Mevcut güncellemeler aşağıda verilmiştir.

29.Ağustos.2012: ilk yayın

27. Sürümler

Bu örnek projede kullanılan programların web adresleri ve sürümleri aşağıda özetlenmiştir.

```
tool chain   : mentor.com,  2012.03-57,
               gcc 4.6.3, arm-none-linux-gnueabi-

u-boot       : denx.de,     2012.07
linux kernel : kernel.org,  3.5.2
busybox      : busybox.net, 1.20.2
rootfs       : hand made,   generic
```

28. Kullanım Hakları

— yazı sonu —