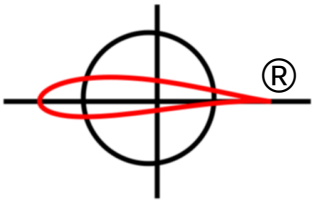
RiscV altında Gömülü Linux Kuruluşu

http://UCanLinux.Com

13 Haziran 2019



Nâzım KOÇ

Bu belge[1], RiscV için Linux işletim sisteminin, power on'dan login'e kadar nasıl kurulacağından ve qemu altında nasıl test edileceğinden bahseder.

Bu belgenin en son sürümü UCanLinux.Com adresinden indirilebilir.

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

[1] `file:MAIN/yazilar/riscv_qemu_kurulus/latex, 31 Temmuz 2019`

Şecere:

13 Haziran 2019 İlk yayın.

31 Temmuz 2019 Yazım hatalarının düzeltilmesi

İçindekiler

Önsöz

ABD'nin yazılımı ve donanımı, açıkça bir tehdit unsuru olarak kullanmasından sonra açık kaynak kodlu yazılımların ve donanımların önemi bir kez daha anlaşılmıştır. Her ne kadar yazılımda açık kaynak kavramı bilinse de, bu kavramın donanımdaki karşılığı pek bilinmemektedir.

Huawei'nin ARM ile lisanslarının askıya alınması[2], gözleri donanımdaki açık kaynak kavramına çevirmiştir. ARM'ın ve risc tabanlı işlemcilerin, şu anki en büyük alternatifi veya rakibi RiscV(risc five okunur) sistemleridir[3]. Bu sistemler açık kaynaklı olup, ARM gibi katı lisans ve kullanım izinlerine tabi değildir. Linux veya BSD gibi açık sistemlerin, donanımdaki karşılığı RiscV olarak düşünülebilir.

RiscV girişimini Aselsan gümüş, Huawei de altın üye olarak desteklemektedir[4]. Ayrıca Alibaba gibi devler de destek vermektedir. Şahsi fikrimiz, ABD'nin Çin'e, özellikle yazılım konusundaki yaptırımları, yazılım konusunda uyuyan bir devi uyandırmaktan başka bir işe yaramayacaktır.

Bu belgede toolchain, bbl, linux kernel, loop device, root file system ve qemu ile baştan sona bir RiscV sisteminin kuruluşu ve testinden bahsedilecektir. RiscV cihazlar henüz aşırı pahalıdır. Bundan dolayı testler emülatör altında gerçeklenecektir.

Bu belgenin güncel hali, pdf formatında, UCanLinux.Com adresinden indirilebilir.

Faydalı olması dileği ile,

-Nâzım KOÇ

[2]https://www.techradar.com/news/arm-stops-all-work-with-huawei-after-us-ban
[3]https://riscv.org/
[4]https://riscv.org/members-at-a-glance/

Bölüm 1

GNU/Linux Kuruluşu

2015 senesinde kurulan ve şu anda 200'den fazla destekçisi bulunan RiscV[1] vakfının amacı, yazılım ve donanım konusunda açık kaynaklı bir mimari oluşturmaktır. Bu projenin orjinal geliştiricisi Berkely üniversitesidir. Oluşum henüz çok yeni olmasına rağmen yavaş yavaş RiscV makineleri ticari ürün olarak piyasaya çıkmaya başlamıştır.

Bu makinelerin üretim adetleri çok kısıtlı olduğundan fiyatları da aşırı pahalıdır. Bundan dolayı OrangePi veya BegaleBone gibi bir makine alıp da üzerinde test yapmamız mümkün değildir. Bunun yerine emülatör altında test yapılacaktır. Test tekniği 2. bölümde verilecektir.

Bu bölümde, bir RiscV makinesine, power-on'dan login'e kadar gelebilen bir GNU/Linux sistemi, tırmalama yöntemi ile kurulacaktır. Yani gerekli olan bütün işler el yordamı ile yapılacaktır ki tarzımız da budur.

"Ben kasmak istemiyorum, yok mu hazır çözüm?" diyenler için aşağıdaki adresler incelenebilir.

Debian port için `https://wiki.debian.org/RISC-V#Debian_port_information`

Fedora port için `https://fedoraproject.org/wiki/Architectures/RISC-V/Installing`

RiscV Tools için `https://github.com/riscv/riscv-tools`

RiscV tools içinde, bu belgede yapılan bütün işlemler tek bir `build.sh` betiği

[1] https://riscv.org, https://wiki.debian.org/RISC-V

ile gerçeklenmektedir.

Ayrıca bu belgede bahsedilecek sistem busybear-linux[2] sistemden esinlenilmiştir. Okuyucu bu sistemi de inceleyebilir. Burada da tırmalayarak bir sistem kurulmuştur.

Bizler de bu çalışmada busybear-linux'deki sistemi örnek aldık. Fakat, çekirdek, açılış betikleri, busybox ve kök dosya sistemi kuruluşu farklı biçimde sunulacaktır.

Bu belgede bahsi geçen konular hemen hemen bütün gömülü Linux sistemlerine uygulanabilir. Sadece boot loader ve çekirdek ile ilişkisi çok farklıdır. Çekirdek, kök dosya sistemi ve açılış betikleri bütün sistemlerde hemen hemen aynı kalacaktır.

Her ne kadar RiscV makineler üzerinde çalışsak da, gömülü Linux sistemleri kurma sistematiği aynı kalacaktır. Daha önce kullanılan çalışma yöntemleri[3] [4] burada da uygulanacaktır. Özetle makine veya mimari değişse de kuruluş yöntemi bakidir.

Bu belgedeki çalışma Ubuntu 16.04 LTS, x86_64 makinesinde ve bash kabuğu altında yapılacaktır. Kullanıcı farklı bir Linux dağıtımı veya sürümü de kullanılabilir. Konular dağıtıma bağlı değildir.

Bu belge üç bölümden oluşmaktadır. Birinci bölümde örnek gömülü sistem imajı kurulacaktır. İkinci bölümde örnek imaj, qemu ile test edilecektir. Son bölümde ise doğrudan emülatör ortamının kendisi incelenecektir.

Sistem kuruluşu için en az 15GB'lık disk alanına ve bolca zamana gerek olacaktır. Vaktim yok diyenler hızlıca aşağıdaki gibi doğrudan sonucu görebilirler. root kullanıcısının şifresi root'tur. `poweroff` komutu ile de emülatör kapatılabilir.

```
$ cd
$ git clone https://github.com/UCanLinux/riscv64-sample.git sample
$ cd sample
$ export PATH=.:$PATH
$ ./run.sh
```

[2] https://www.cnx-software.com/2018/03/16/how-to-run-linux-on-risc-v-with-qemu-emulator/

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

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

1.1 Proje dizininin seçimi

Bütün proje tek bir dizin altında yürütülecektir. Bu hem projeyi takibi hem de yapılan işlemlerin anlatılmasını çok kolaylaştıracaktır.

Projemiz `/opt/riscv/` altına kurulacaktır ve bütün çalışma bu dizin altında yapılacaktır.

Terminalden sürekli bu dizini girmemek ve genelleme yapmak için, `~/.bashrc` dosyasının son satırına, aşağıdaki export tanımı girilmelidir.

```
export RISC=/opt/riscv
```

Okuyucu farklı bir dizin de seçebilir. Bu durumda export tanımına, kendi seçtiği dizin adını girmelidir. Bash dışında kabuk kullananlar da benzer bir tanım yapmalıdır. Bu belge sonuna kadar bütün çalışmalar bash kabuğu altında yapılacaktır.

`~/.bashrc` güncellemesinden sonra, `export` tanımının aktif olabilmesi için terminal kapatılıp tekrar açılmalıdır.

Sonra proje dizinin sahipliği aşağıdaki gibi güncellenir.

```
$ sudo mkdir $RISCV
$ sudo chown nazim:nazim $RISCV
```

Yetki hatalarını bastırmak için, kullanıcılar genelde root kullanıcısı ile çalışmayı tercih ederler. Bu çok tehlikeli bir durumdur. Sürekli root ile çalışıldığında Unix'in yetkilendirme özellikleri perdelenmiş olur. Kazara makineyi kaybetme ihtmali dahi olabilir.

Bundan dolayı yukarıda bulunan `chown` tanımı mutlaka düzgünce girilmeli ve herkes kendi kullanıcısı ile çalışmalıdır. Her kullanıcı `$id` komutu ile kendi kullanıcı ve grup adını bulmalı ve chown komutunda `nazim:nazim` olan yere kendi kullanıcı ve grup adını yazmalıdır.

Çalışmamız boyunca gerekli olacak bazı yardımcı dosyalar aşağıdaki gibi indirilmelidir.

```
$ cd $RISCV/src
$ git clone https://github.com/UCanLinux/riscv64-sample.git sample
```

Bütün yardımcı dosyalar `src` dizini altına inecektir. Ayırca bu belgedeki çalışmamız boyunca gerekli olacak bütün kaynak kodları da `src` altına yüklenecektir.

Çalışma dizinini kurduktan sonraki aşama toolchain kuruluşudur.

1.2 Derleyicinin kurulması

Bu bölümde RiscV derleyici araçları[5] kurulacaktır. Öncelikle toolchain derlemesinde gerekli olacak paketler aşağıdaki gibi kurulmalıdır.

```
$ sudo apt-get install autoconf automake autotools-dev \
        curl libmpc-dev libmpfr-dev libgmp-dev gawk \
        build-essential bison flex texinfo gperf libtool \
        patchutils bc zlib1g-dev libexpat-dev
```

Derleme araçlarını indirme, derleme ve kurma işlemi[6] aşağıdaki gibi yapılabilir.

```
# indir

$ cd $RISCV/src
$ git clone --recursive https://github.com/riscv/riscv-gnu-toolchain toolchain

# derle

$ cd toolchain
$ ./configure --prefix=$RISCV

$ make linux -j$(nproc)
```

`git clone` komutu ile toolchain araçları `$RISCV/src/toolchain` altına inecektir.

`--prefix` sayesinde kurulum işlemi `$RISCV` dizini altına yapılacaktır.

`./configure` komutunda --enable-multilib seçeneği girilmiş olsaydı, 32 bit derleyici araçları da kurulacaktı. Derleme süresi aşırı uzun olduğu ve ihtiyacımız olmadığı için bu seçenek girilmemiştir. Bu durumda sadece 64 bitlik derleyici kurulacaktır.

Aynı anda hem 32 hem de 64 bitlik derleyici kurulursa, kütüphanelerin yerleri ve isimleri değişmektedir. Okuyucu şimdilik sadece 64 bitlik derleyici kurmalıdır. Yani test için de olsa --enable-multilib seçeneği girilmemelidir.

[5] toolchain

[6] https://github.com/riscv/riscv-gnu-toolchain

`-j` seçeneği ile paralel çalışacak iş sayısı verilir. Buraya genelde host makinedeki işlemci sayısı girilir. Makinede tek işlemci varsa bu seçenek girilmez.

`$ nproc` komutu ile makinedeki işlemci sayısı elde edilebilir. Benim tarihi makinemde 2 işlemci vardır. Bu durumda ilgli komut `$ make linux -j2` olacaktır. Birden fazla işlemci varsa, `-j` seçeneği derleme zamanını kısaltacaktır.

Toolchain kuruluşu bitmiştir. Toolchain'in çapraz derleyicileri `$RISCV/bin/` altına kurulmuştur. Derleme araçlarına ait komutların bulunduğu bu dizin, `~/.bashrc` dosyasının son satırına, aşağıdaki gibi, `PATH` değişkeninin önüne eklenmelidir[7]

```
# $HOME dizinine gel
$ cd

# PATH güncelle

$ vi .bashrc
export PATH=$RISCV/bin:$PATH

# test için yeni terminal aç
# riscv gir ve iki kez tab'a bas

$ riscv <tab> <tab>

riscv64-unknown-linux-gnu-addr2line      riscv64-unknown-linux-gnu-gdb
riscv64-unknown-linux-gnu-ar             riscv64-unknown-linux-gnu-gdb-add-index
riscv64-unknown-linux-gnu-as             riscv64-unknown-linux-gnu-gfortran
riscv64-unknown-linux-gnu-c++            riscv64-unknown-linux-gnu-gprof
...

# derleyici doğru yerde mi, incele?

$ which riscv64-unknown-linux-gnu-gcc

/opt/riscv/bin/riscv64-unknown-linux-gnu-gcc
```

Bazı kaynaklarda `$ RISCV/bin` dizini, `PATH` değişkeninin sonuna eklenir. Bu ekleme, pek de sağlıklı değildir. Aynı derleyici araçlarından başka bir dizinde varsa ki benim makinemde 10 civarı toolchain kuruludur, etkin projenize ait

[7] vi editörü yerine, gedit, pico, none vs kullanılabilir. vi'dan kaydederek çıkmak için ESC : wq ve kaydetmeden çıkmak için ESC : q! girilir.

komut değil de başka dizindeki aynı isimli komut gelebilir. Bunu engellemek için ekleme her zaman `PATH` başına yapılmalıdır.

∼`/.bashrc` güncellendikten sonra `exit` ile terminalden çıkılmalı ve yeni bir terminal açılmalıdır. Artık yeni `PATH` tanımı güncel olacaktır. Yeni terminalde güncel `PATH` tanımını test için `$ riscv` girilir ve 2 kez tab tuşuna basılırsa bütün derleme araçlarının isimlerinin gelmesi gerekir. Eğer gelmiyorsa `PATH` değişkeni doğru güncellenmemiş veya yeni terminal açılmamıştır.

`$ which` komutu ile de derleyicinin oturduğu dizinin doğruluğu kontrol edilir. Bu belge boyunca, derleyicimiz `/opt/riscv/bin/` veya `$RISCV/bin` altında oturacaktır.

Aşağıdaki gibi, hemen hello.c kodu yazılıp, derlenip, yürütülebilir dosyanın içeriği incelenebilir.

```
$ cd $RISCV/src/sample

$ cat hello.c

#include <stdio.h>

int main(){
   fprintf(stdout, "hello RiscV\n");
   return 0;
}

# çapraz derle

$ riscv64-unknown-linux-gnu-gcc -Wall hello.c -o hello

# boya bak, ~11K

$ ls -l hello
-rwxrwxr-x 1 nazim nazim 11696 Haz  5 09:55 hello

# strip ile debug ve sembol bilgilerini yok et
# program artık debug edilemez ve boyu daha kısa olacaktır

$ riscv64-unknown-linux-gnu-strip hello

# boya tekrar bak
# epeyce kısaldı, ~6K kadar oldu

$ ls -l hello
-rwxrwxr-x 1 nazim nazim 5992 Haz  5 09:56 hello
```

```
# yürütülebilir dosyayı incele
# kolay okunması için, çıkış alt alt yazılmıştır
# normal çıkış tek satırdır

$ file hello

hello: ELF 64-bit LSB executable,
         UCB RISC-V,
         version 1 (SYSV),
         dynamically linked,
         interpreter /lib/ld-,
         for GNU/Linux 3.0.0,
         stripped
```

`file` komutunun çıkışı çok önemlidir. Bu komut dosya tipini ve özelliklerini tespit eder. Komut çıkışını kısaca inceleyecek olursak...

`ELF 64-bit` ile üretilen kodun 64 bitlik ELF formatında olduğunu söyler. ELF formatı, yürütülebilir dosya ve kütüphane formatıdır ve işlemci mimarisinden bağımsızıdır.

`UCB RISCV` ifadesindeki UCB, `University of California, Berkely` demektir. RiscV'in orjinal tasarımcısı bu üniversitedir.

`dynamically linked` ifadesi ifadesi, bu programın dinamik kütüphanelere bağımlı olduğunu gösterir. Yani program çalışacağı zaman ilgili kütüphaneler de programla birlikte yüklenecektir. Programın bağımlı olduğu kütüphaneler aşağıdaki gibi bulunabilir.

```
$ riscv64-unknown-linux-gnu-objdump -p hello  | grep NEEDED

NEEDED               libc.so.6
```

Yani bu program `libc.so.6` kütüpahenesine bağımlıdır. `hello` kodunu çalıştıracağımız RiscV makineye bu kütüphane de taşınmalıdır. Kök dosya sisteminin kuruluşu sırasında, bu kütüphanenin nasıl temin edileceği anlatılacaktır.

`interpreter /lib/ld-` ifadesi, bu programın çalışması için `/lib` dizini altında `ld.so` kütüphanesine[8] gerek olduğunu söyler. Diğer bir deyişle `libc.so.6`

[8] $ man ld.so

kütüphanesi dışında bir de bu kütüphaneye gerek vardır. Bu çok özel kütüphaneye yorumlayıcı denir.

Yorumlayıcı, bir kütüphane olmasına rağmen, aynı zamanda çalışabilir bir programdır. Yani kullanıcı doğrudan bu kütüphaneyi terminalden çalıştırabilir. Başka hiç bir kütüphaneye bağlılığı yoktur. Bu kütüphaneye dynamic linker/loader denir. Temel bir görevi vardır. Çalışacak programı yükler, bağımlı kütüphaneleri tespit eder ve yükler, sonra da program ve kütüphane içinde bulunan ve atanmamış adresleri çözer, atar ve programı çalışmaya hazır hale getirir ve çalıştırır.

RiscV için kök dosya sistemi kurulurken, bu kütüphane de diğer kütüphanelerle birlikte kurulacaktır.

`stripped` ifadesi ise, program içinde bulunan ve doğrudan yürütmeyi engellemeyecek ifadelerin, yani debug bilgileri, sembol adres ve isimleri gibi bilgilerin program içinden soyulup atıldığını söyler.

Bizler bu işi `riscv64-unknown-linux-gnu-strip` komutu ile yaptık. Bu komutu kullanmamış olsaydık, `file` komutunun çıkışında `not stripped` yazacaktı.

Müşteriye verilecek uygulamalar mutlaka strip edilerek verilmelidir. `strip` edilmemiş ve `-g` ile derlenmiş programlar müşteriye verilirse, kaynak kodu da verilmiş olacaktır.

Host tarafı için bir örnek aşağıda verilmiştir.

```
$ cd $RISCV/src/sample

# native derle

$ gcc -g -o hello hello.c

# incele
# "not stripped" lafını gör

$ file hello

hello: ELF 64-bit LSB executable,
       x86-64,
       version 1 (SYSV),
       dynamically linked,
       interpreter /lib64/l,
```

```
       for GNU/Linux 2.6.32,
       BuildID[sha1]=0060e106c40640a38fd0d12eb6572c1f56fceff8,
       not stripped              <-- !!!


# debug ile kaynak kodunu elde et
# öyle ki açıklamalar dahi çalışabilir kodun içinde mevcut olacaktır

$ gdb ./hello

Copyright (C) 2016 Free Software Foundation, Inc.
...
Reading symbols from ./hello...done.

(gdb) list
1        #include <stdio.h>
2
3        int main(){
4           fprintf(stdout, "hello RiscV\n");
5           return 0;
6        }
7
(gdb) q
```

Şimdi tekrar hello kodunu çapraz derleyelim ve çalıştıralım.

```
$ cd $RISCV/src/sample
$ riscv64-unknown-linux-gnu-gcc -Wall hello.c -o hello

$ ./hello
bash: ./hello: cannot execute binary file: Exec format error
```

Host makinemiz RiscV mimarisinde olmadığı için program hata verecektir.

Artık çapraz derleyicimiz hazırdır. Sonraki adımlarda boot loader, kernel ve kök dosya sistemleri sistematik olarak kurulacak ve qemu altında login'e kadar gelinerek hello kodu çalıştırılacaktır.

1.3 Çekirdeğin derlenmesi

Çekirdeğin kaynak kodu, aşağıdaki komut ile linux dizini altına indirilir ve 4.20 sürümü seçilir.

```
$ cd $RISCV/src
$ git clone https://github.com/riscv/riscv-linux.git linux
$ cd linux
$ git checkout riscv-linux-4.20
```

`defconfig` argümanıyla, kaynak kodu ile gelen ve `linux/arch/riscv/configs` altında bulunan örnek config dosyası, aşağıdaki `make defconfig` komutu ile, `linux/` dizini altına `.config` adı ile kopyalanır.

```
$ cd $RISCV/src/linux
$ make ARCH=riscv defconfig
```

Daha sonra `menuconfig` argümanı ile çekirdeğin config bilgileri incelenebilir veya güncellenebilir.

```
$ cd $RISCV/src/linux
$ make ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu-  menuconfig
```

Mevcut config dosyası kullanılabileceği gibi, bizim örnek config dosyası da istenirse aşağıdaki gibi kullanılabilir.

```
$ cd $RISCV/src/linux
cp ../sample/kernel.config .config
```

Sonra tekrar, aşağıdaki gibi `make menuconfig` komutu girilebilir.

```
$ cd $RISCV/src/linux
$ make ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu-  menuconfig
```

Çekirdek derlemelerinde her zaman `ARCH` değişkeni ile mimari ve `CROSS_COMPILE` değişkeni ile de çapraz derleyicinin ön-eki verilir. Ön-ek her zaman

`riscv64-unknown-linux-gnu-gcc` ifadesinde, `gcc`'den önceki bütün ifadedir. Bu eke, çapraz derleyici ön-eki[9] denir.

`ARCH` ve `CROSS_COMPILE` değişkenleri, bash içinde veya `~/.bashrc` içinde `export` ile de verilebilir. Bu durumda, make komutlarında, `ARCH` ve `CROSS_COMPILE` ifadelerine gerek olmayacaktır. Bu tarz kullanım, esnek değildir. Çünkü bizler, örneğin ARM, MIPS veya PPC işlemcileri için de çapraz derleme yapmaktayız. Her seferinde `~/.bashrc`'deki ilgili iki export tanımının güncellenmesi gerekli olacaktır. Bu da zahmetlidir ve bu güncelleme her zaman gözden kaçabilir, unutulabilir. Genel uygulama, bu değişkenlerin doğrudan make komutu ile verilmesi şeklindedir.

Eğer çekirdek ilk defa derleniyorsa, mevcut seçimlere dokunulmaması yerinde olur. RiscV sistemi açıldıktan sonra, yavaş yavaş çekirdek üzerinde, `make menuconfig` ile güncellemeler yapılabilir.

Çekirdek, aşağıdaki gibi, `all` girişi ile derlenir.

```
$ cd $RISCV/src/linux
$ make ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu-  all -j$(nproc)
```

Kaynak kodu ile gelen defconfig dosyası ile derleme yapılması tavsiye edilmez. En azından `kernel debug` gibi açık olmaması gereken, gereksiz pek çok seçenek işaretlenmiş olabilir.

Derleme sorunsuz biterse hem çekirdek hem de çekirdek modülleri üretilecektir. Yeni çekirdek, kaynak kodunun hemen altında `vmlinux` adı ile üretilmiş olacaktır. `vmlinux` programının incelenmesi aşağıdaki gibi yapılabilir.

Az yer kaplasın diye çıkışlar kırpılmıştır. Kırpılan yerler "..." ile gösterilmiştir.

```
$ cd $RISCV/src/linux

# çekirdeği incele

$ ls -l vmlinux
-rwxrwxr-x 1 nazim nazim 10428912 May 30 07:22 vmlinux

# file ile incele
```

[9] cross compiler prefix

```
$ file vmlinux

vmlinux: ELF 64-bit LSB executable,
         UCB RISC-V,
         version 1 (SYSV),
         statically linked,
         BuildID[sha1]=cb13ee740a3a2e2f5bda4bdbae8a4530df6f0c79,
         not stripped

# üretilen modülleri incele
# ko, kernel object demektir

$ find . -name "*.ko"

./drivers/char/hw_random/rng-core.ko
./drivers/video/backlight/lcd.ko
./drivers/crypto/virtio/virtio_crypto.ko
...
./lib/modules/4.20.0+/kernel/crypto/crypto_engine.ko
./lib/modules/4.20.0+/kernel/crypto/echainiv.ko
./lib/modules/4.20.0+/kernel/crypto/hmac.ko

# rastgele seçilen bir modülü incele

$ modinfo ./crypto/hmac.ko
filename:       /opt/riscv/src/linux/./crypto/hmac.ko
alias:          crypto-hmac
alias:          hmac
description:    HMAC hash algorithm
license:        GPL
depends:
intree:         Y
name:           hmac
vermagic:       4.20.0+ SMP mod_unload riscv
```

`file` komutunun çıkışına bakılırsa, `statically linked` ifadesi görülebilir. Çekirdek hiç bir dış kütüphaneye bağımlı değildir. Tabii ki çekirdeğin içinde de kullanılan kütüphaneler vardır. Hatta bu kütüphanelerin API'leri standard kütüphaneleri andırır. Ama doğrudan çekirdek kodu içinde gömülüdür, ayrıca bulunmazlar.

`not stripped` ifadesi de çekirdeğin içinde symbol ve debug bilgilerinin bulunduğunu gösterir. Bir sonraki bölümde çekirdek, strip komutu ile soyularak ufaltılacaktır.

`*.ko` çekirdek modülleri, kök dosya sistemi kurulurken kullanılacaktır. `menuconfig` sırasında `<M>` veya {`M`} olarak işaretlenen her seçenek `*.ko` şeklinde, çekirdek dosyasından, yani `vmlinux`'ten ayrı bir dosya olarak üretilecektir. Bu `*.ko` dosyaları, ihtiyaç olduğunda, yürütme zamanında çekirdeğe eklenebilir veya atılabilirler.

Dikkat edilirse bu derlemede, ARM derlemelerinde olduğu gibi "device tree blob" üretilmemiştir. Ayrıca çekirdek kodu da uImage değildir. Çünkü bu uygulamada, boot loader olarak u-boot değil, bbl kullanılmıştır.

Sonraki bölümde bbl ile çekirdek bir araya getirilecek ve tek bir dosya oluşturulacaktır.

1.4 Ön-yükleyici

Makine açılır açılmaz, işletim sisteminden önce çalışan programlara ön-yükleyici[10] denir. Ön-yükleyicinin esas görevi Linux çekirdeğini yüklemektir. RiscV sistemlerinin ön-yükleyicileri riscv-pk[11] projesi içinde bulunurlar.

Aslında u-boot projesi de RiscV makinelerini desteklemektedir[12]. Fakat yaygın olmadığı ve daha önceki yazılarımızda da hep u-boot kullandığımız için, burada pk yöntemi kullanılacaktır.

pk, proxy kernel demektir. Buradaki kernel lafının Linux kernel ile bir ilgisi yoktur. proxy, vekil veya aracılık eden demektir.

pk projesi içinde 2 adet ön yükleyici bulunur. Birisi `pk` ve diğeri `bbl`'dir.

`pk` programı statik derlenmiş bir yükleyicidir. Esas görevi, işletim sistemi desteği olmadan program çalıştırmaktır. Çalışacak program ile donanım arasındaki katman olarak düşünülebilir. `pk` programı, çalışacak programın basit i/o taleplerini yerine getirir. Pratikte kullanılan bir yükleyici değildir. Zaten Linux çekirdeğini yükleme özelliğine de sahip değildir.

Diğer yükleyiciye bbl[13] denir. Bu programın esas görevi Linux çekirdeğini veya farklı bir işletim sisteminin çekirdeğini veya pk programını yüklemektir. Bu programa PC BIOS veya HAL[14] gözü ile bakılabilir. Linux çekirdeği ile donanım arasında oturur ve çekirdeğin donanım ile ilgili taleplerini yerine getirir. Bbl olmadan Linux çalışamaz.

riscv-pk projesi aşağıdaki gibi indirilip, derlenip, kurulabilir.

```
$ cd $RISCV/src
$ git clone https://github.com/riscv/riscv-pk.git pk

$ cd pk
$ mkdir build
$ cd build
```

[10] boot loader
[11] https://github.com/riscv/riscv-pk
[12] FOSDEM_RISCV_SBI_Atish.pdf
[13] Berkely boot loader
[14] Harware Abstraction Layer

```
$ ../configure --enable-logo \
               --host=riscv64-unknown-linux-gnu \
               --with-payload=$RISCV/src/linux/vmlinux \
               --prefix=$RISCV

# logo'yu değiş
$ cp $RISCV/src/sample/bbl_logo_file $RISCV/src/pk/build

$ make -j$(nproc)
$ make install
```

`.configure` komutundaki `--with-payload` seçeneği sayesinde bbl ve Linux çekirdeği bir araya getirilerek bbl imajı oluştururlur. Bbl imajına, `bbl + Linux kernel` gözü ile bakılabilir. Bu imaj yürütülebilir, statik bir dosyadır.

`--prefix=$RISCV` seçeneği ile kuruluşun kök dizini verilir. Üretilen dosyalar `$RISCV/riscv64-unknown-linux-gnu/bin` altına kurulur. Bu dosyalar aşağıdaki gibi incelenebilir.

```
$ cd $RISCV/riscv64-unknown-linux-gnu/bin

# pk statik bir ELF programıdır

$ ls -l pk
-r-xr-xr-x 1 nazim nazim    66432 May 28 23:55 pk

$ file pk
pk: ELF 64-bit LSB executable,
        UCB RISC-V,
        version 1 (SYSV),
        statically linked,
        not stripped


# bbl imajı da statik bir programdır
# içinde Linux çekirdeği vardır

$ ls -l bbl
-r-xr-xr-x 1 nazim nazim 10416560 May 28 23:55 bbl

# boyu kısalt

$ chmod 755 bbl
$ riscv64-unknown-linux-gnu-strip bbl

$ ls -l bbl
```

```
-rwxr-xr-x 1 nazim nazim 8349216 Haz  6 10:16 bbl

$ file bbl
bbl: ELF 64-bit LSB executable,
         UCB RISC-V,
         version 1 (SYSV),
         statically linked,
         stripped
```

OpenSBI[15] [16] adlı yeni bir ön-yükleyici üzerinde çalışılmaktadır. Bu ön-yükleyici pk ve bbl'nin yerini alacaktır. Son derece yeni olan bu sistemin 0.1 sürümü 2019, Ocak sonunda duyrulmuştur[17].

Elimizde artık boot edebilecek bir çekirdek mevcuttur. Tek eksiğimiz kök dosya sistemidir.

[15] Open Source Supervisor Binary Interface

[16] https://github.com/riscv/riscv-sbi-doc/blob/master/riscv-sbi.adoc

[17] https://riscv.org/2019/01/risc-v-community-releases-opensbi-to-foster-continued-ecosystem-growth/

1.5 İskelet kök dosya sistemi

Çekirdeğin açılır açılmaz bağladığı dosya sistemine kök dosya sistemi[18] denir. Genelde kök dosya sisteminde `/bin, /sbin, /etc, /tmp` gibi dizinler, Unix komutları, config dosyaları, vb bulunur.

Kök dosya sistemi busybox veya buildroot[19] ile kurulabilir. Bu uygulamada kök dosya sistemi busybox[20] ile kurulacaktır.

Busybox projesinde, sık kullanılan bazı komutlar, daha az seçenek ile tekrar yazılmış ve çalışabilir tek bir dosya altında toplanmıştır. Bu dosyaya busybox denir. Busybox projesi `/bin, /sbin, /usr/bin` ve `/usr/sbin` altında bulunan komutları temin eder. Fakat kök dosya sistemi sadece bu komutlardan ibaret değildir.

Sonraki bölümlerde kök dosya sistemi, sistematik bir biçimde kurulacaktır. Öncelikle boş dizinlerden oluşan iskelet bir dosya sistemi kurulmalıdır.

```
# örnek içindir, uygulamayın!

$ cd $RISCV/src/sample
$ mkdir skeleton
$ cd skeleton
$ mkdir dev etc home lib mnt proc root sys tmp var
```

Daha sonra `/etc` dizininin altı, örnek bir sistemden kopyalanır. Boş dizinler ile `/etc` altındaki dosyaların topluluğuna iskelet kök dosya sistemi denir.

Örnek bir iskelet dosya sistemi `$RISCV/src/sample/skeleton` altında mevcuttur. Bu iskelette bulunan `/etc` altındaki bütün dosyalar, ASCII olduğu için, x86 sistemleri dahil, herhangi bir yerden alınabilir. Bizler iskelet kök dosya sistemini, doğrudan buildroot'un iskelet kök dosya sisteminden alıp, üzerinde ufak güncellemeler yaptık.

Şimdi elimizde sadece `/etc`'nin içinin dolu olduğu, diğer bütün dizinlerin boş olduğu, hemen hemen tamamen boş, iskelet bir kök dosya sistemi bulunmaktadır. Bu iskelet kök dosya sistemi, sistematik bir şekilde, aşağıdaki gibi ete kemiğe bürünecektir.

[18] root file system
[19] https://buildroot.org
[20] https://busybox.net

/bin busybox'dan alınır.

/sbin busybox'dan alınır.

/usr/bin busybox'dan alınır.

/usr/sbin busybox'dan alınır.

/lib toolchain'den alınınır.

/lib/modules çekirdekten alınır.

/etc dosyalar, el yordamı önceden kopyalanmıştır.

/home içi şimdilik boştur. Daha sonra kullanıcıların `$HOME` dizinleri burada yaratılabilir.

/var yazılabilir dizindir. İçindeki dizinler `/tmp`'ye yönlendirilmiştir.

/mnt içi şimdilik boştur. Mount point dizinleri burada yaratılabilir.

/root root kullanıcısının home dizini. Boş sayılır.

/dev çekirdek tarafından açılışta mount edilir.

/proc /etc/rcS tarafından açılışta mount edilir.

/sys /etc/rcS tarafından açılışta mount edilir.

/tmp /etc/rcS tarafından açılışta mount edilir, yazılabilir dizindir.

Projemizdeki kök dosya sistemi, açıldıktan sonra `salt-okunur`[21] bağlanacaktır. Böylece ani kapanmalarda kök dosya sisteminin bütünlüğü garanti altına alınacaktır. Fakat işletim sistemi ayakta iken bazı programlar illa ki bazı bilgileri yazmak isteyecektir.

`/tmp` dizini, sanal bellekte[22] kurulan bir dosya sistemidir. Kök dosya sistemi salt-okunur olsa da `/tmp` dizini her zaman yazılabilir moddadır. Yazılabilir bütün dosya ve dizinler her zaman `/tmp`'ye yönlendirilirler. `/var` altında bulunan bütün dizinler `/tmp`'ye yönlendirilmiştir. Böylece salt-okunur bir dosya sisteminde, yazılabilir dizinler veya dosyalar bulunabilir.

[21] read-only

[22] virtual memory, RAM + swap alanından oluşur

`tmpfs` dosya sistemi[23] sanal bellekte kurulduğu için, tutulan bütün bilgiler güç kesilince yok olur, uçar[24]. Bilgilerin kalıcı olması için, ayrı bir disk bölümü, oku/yaz[25] olarak mount edilip, bilgiler buraya yazılmalıdır. Bu disk bölümü uçucu olmayan bellekte[26] kurulmalıdır.

Örnek kök dosya sisteminin ana yapısını busybox programı oluşturur.

[23] temporary file system
[24] volatile memory
[25] read/write
[26] non-volatile memory

1.6 Busybox

Busybox programı aşağıdaki gibi, indirilip, derlenip, kurulabilir.

```
$ cd $RISCV/src/

# indir
$ git clone git://busybox.net/busybox.git

# her zaman karalı sürüm kullan
# bu belge yazılırken son kararlı sürüm 1.30 idi

$ cd busybox
$ git checkout 1_30_stable

# varsayılan config dosyası ile
% kullanılacak komutları ve özelliklerini seç
$ make menuconfig

# bizim örnek config dosyamız da kullanılabilir
# ama zorunlu değildir

$ cp ../sample/busybox.config .config
$ make menuconfig

# derle

$ make CROSS_COMPILE=riscv64-unknown-linux-gnu- all -j$(nproc)

# _install/ altına kuruluş yap

$ make CROSS_COMPILE=riscv64-unknown-linux-gnu- install

# _install/ altını mutlaka incele
# sembolik link'leri gör

$ cd _install
$ ls -l

$ ls -l bin

total 916
lrwxrwxrwx 1 nazim nazim      7 May 30 08:05 arch -> busybox
lrwxrwxrwx 1 nazim nazim      7 May 30 08:05 ash -> busybox
lrwxrwxrwx 1 nazim nazim      7 May 30 08:05 base64 -> busybox
-rwxr-xr-x 1 nazim nazim 935448 May 30 08:05 busybox
lrwxrwxrwx 1 nazim nazim      7 May 30 08:05 cat -> busybox
```

```
...
lrwxrwxrwx 1 nazim nazim      7 May 30 08:05 vi -> busybox
lrwxrwxrwx 1 nazim nazim      7 May 30 08:05 watch -> busybox
lrwxrwxrwx 1 nazim nazim      7 May 30 08:05 zcat -> busybox
```

_install/bin dizini incelenirse, tek bir busybox programının olduğu ve diğer bütün komutların bu tek komuta sembolik link ile bağlandığı görülebilir. _install altındaki diğer bütün komutlar da /bin/busybox programını, sembolik linkler ile göstermektedirler.

Yaklaşık 400 komut, balık istifi gibi, tek bir busybox programı içine tepiştirilmiştir. Yukarıdaki ls çıkışında da görüleceği gibi, bütün komutlar toplamda 1MB'dan daha ufak bir alan kaplar. Bu çok büyük bir başarıdır. Tabii ki bu komutların kullanım alanlarının çok kısıtlı olduğunu, yani seçeneklerinin çok az olduğunu söylemeye gerek yoktur.

Örneğin Ubuntu 16.04 dağıtımındaki vim programının boyutu 2MB'tan fazladır.

```
$ ls -l /usr/bin/vim.gnome
-rwxr-xr-x 1 root root 2932160 Kas 24  2016 /usr/bin/vim.gnome
```

Busybox'ın içinde de vim benzeri vi komutu vardır. vi dahil, diğer 400'e yakın komutun toplam boyu 1MB'tan azdır. vim seçenekleri kalın bir kitap boyutundadır. Busybox'taki vi'nin seçenekleri ise birkaç tanedir.

Şimdi elimizde skeleton/ altında iskelet kök dosya sistemi ve busybox altında da Linux komutları mevcuttur. Artık sistematik bir biçimde, iskelet dosya sisteminden hareketle kök dosya sistemi inşa edilebilir.

1.7 Kök dosya sisteminin kuruluşu

Öncelikle, iskelet dosya sistemini bozmamak için `RootFS` altında bir kopyası alınacaktır. Örnek kök dosya sistemimiz artık `RootFS` olacaktır. İskelet dosya sistemi, `RootFS` altına aşağıdaki gibi kopyalanır.

```
$ cd $RISCV/src/sample
$ mkdir RootFS
$ cd RootFS
$ cp -a ../skeleton/* .  # komut sonundaki noktayı unutma :)
```

Kopyalama işlemi `-a` seçeneği ile yapılmalıdır. `-a`, arşiv demektir. Bu seçenek ile sembolik link'ler olduğu gibi kopyalanır, link sonuna kadar gidilmez. Alt dizinler de kopyalanır. Aynı zamanda bütün dosya nitelikleri de, sahiplik, mod vs. korunur.

`/bin, /sbin, /usr/bin` ve `/usr/sbin` dizinleri aşağıdaki gibi busybox'tan `RootFS` altına kopyalanır.

```
# busybox kopyala
$ cd $RISCV/src/sample/RootFS
$ cp -a ../../busybox/_install/*  .
```

Daha sonra çekirdekte bulunan modüller `/lib/modules` altına kopyalanmalıdır. Bunun için çekirdek içinde bulunan bütün `*.ko` dosyaları bulunmalı ve `RootFS/lib` altına kopyalanmalıdır. Bu işlem el ile yapılabileceği gibi aşağıdaki gibi `Makefile` yardımı ile de yapılabilir.

```
# modülleri kopyala

$ cd $RISCV/src/linux

$ make ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu-  \
        INSTALL_MOD_PATH=$RISCV/src/RootFS modules_install
```

`INSTALL_MOD_PATH` ile modüllerine kopyalanacağı kök dosya sisteminin kökü verilir. Bütün modüller `/lib/modules/‘uname -r‘` altına kopyalanır.

Daha sonra toolchain içinde bulunan kütüphaneler `/lib` altına kopyalanmalıdır. Bu kütühaneler nerededir?

Kütüphanelerin, toolchain içinde oturduğu yeri öğrenmek için, aşağıdaki gibi `-print-sysroot` seçeneği kullanılır.

```
# kütüphanelerin toolchain'deki yerini tespit et
$ riscv64-unknown-linux-gnu-gcc  -print-sysroot
/opt/riscv/bin/../sysroot
```

Komutun çıkışına göre, RiscV toolhanin'ine ait kütüphaneler `/opt/riscv/sysroot` altındaymış. Derleme yapmak için gerekli include ve kütüphanelerin bulunduğu dizine system root veya kısaca sysroot denir. Bu kütüphaneler her toolchain'de farklı yerde otururlar. Tam yerini öğrenmek için, örneğimizdeki gibi `-print-sysroot` seçeneği kullanılmalıdır.

Yeri belli olan kütüphaneler, aşağıdaki gibi `RootFS` içine kopyalanırlar.

```
$ cd $RISCV/src/sample/RootFS

# kütüphaneleri kopyala

$ cp -a /opt/riscv/sysroot/lib  .
```

Bazı dosyalar ve kütüphaneler şu an için gerekli değildir. Gereksiz dosya ve kütüphaneler aşağıdaki gibi silinebilir.

```
$ cd $RISCV/src/sample/RootFS/lib

# gereksizleri sil

$ rm -f *.a *.la *.spec *fortran* ../linuxrc
```

`*.a` uzantısına sahip statik kütüphanelerin RiscV üzerinde olmasının bir anlamı yoktur. Çünkü bu kütüphaneler program geliştirme sırasında, link aşamasında kullanırlar. Daha sonra gerekli olmazlar. RiscV üzerinde de, şu anki basit RootFS ile program derlemesi yapmak mümkünde değildir. Bu yüzden silinmişlerdir.

Benzer şekilde, `*.la` ve `*.spec` dosyaları, link aşamasında kullanılan, yardımcı ASCII dosyalardır. Bu dosyalar da link aşamalarında kullanıldığı için gerekli değildir, silinebilirler.

Bir de fortran kütüphaneleri vadır. Gençliğimizde bolca kullandığımız bu programala dili ile şimdilik işimiz olmayacaktır. Fortan ile ilgili bütün dosyalar silinmiştir.

Gereksiz bir dosya da `/linuxrc`'dir. `/linuxrc` dosyası, eskiden kullanılan, ramdisk destekli açılışın, çalışacak ilk dosyasıdır. Yani çekirdek, kök dosya sistemini bağladıktan sonra gözü kapalı olarak `/linuxrc` dosyasını çalıştırır. tmpfs destekli initramfs[27] çıktıktan sonra, ramdisk destekli açılış artık kullanılmamaktadır. Bundan dolayı bu dosya da silinebilir.

Yeri gelmişken belirtelim. Ramdisk destekli açılışta, kernel tarafından çalıştırılacak ilk program `/linuxrc`'dir. Tmpfs destekli açılışta `/init` ve diğer açılışlarda ise `/sbin/init` programlarıdır. Bizim açılış tekniğimizde ilk çalışacak program `/sbin/init` olacaktır.

`busybox` temelli bir sistem için aslında çok az kütüphaneye gerek vardır. İlerde gerekli olabilir diye `/lib` altında bulunan paylaşılan kütüphanelerin neredeyse tamamı kopyalanmıştır. Aslında hiç de gerek yoktur.

Eksik kütüphane hatası alınırsa, `lib/` veya `/usr/lib`'den gerekli kopyalamalar daha sonra da yapılabilir. Kopyalama yapılırken toolchain'deki dizinle, kopya yapılan yerdeki dizin birebir aynı olmalıdır. Örneğin toolchain'deki foo kütüphanesi `usr/lib/hede/foo` altında ise, kopyalanan yerde de `usr/lib/hede/foo` olmalıdır. Kütüphanenin `PATH` değeri korunmalıdır. Sebebi basittir.

Çapraz derleme yapılırken, programın bağlı olduğu kütüphaneler kod içinde `PATH` bilgileri ile yazılırlar. Yorumlayıcı da programı yükleyeceği zaman, program içindeki kütüphaneleri, çarpaz derleyicinin yazdığı yerde arar. Bundan dolayı kütüphanelerin `PATH` bilgileri, kopyalanan yerde de birebir aynı olmalıdır. Olmazsa ne olur? Bunun pek çok çözümü vardır. Ama varsayılan değer olarak `PATH` bilgilerini aynı tutmak işleri çok kolaylaştırır.

Github sistemi boş dizinleri kaydedemediği için gerekli boş dizinler `skeleton` içinde yaratılamamıştır. Son olarak aşağıdaki gibi boş dizinler kurulur.

```
# boş dizinleri kur

$ cd $RISCV/src/sample/RootFS
mkdir dev home mnt proc sys tmp var

$ cd etc/network
```

[27] initial ram file system

```
$ mkdir if-down.d  if-post-down.d  if-pre-up.d  if-up.d
```

İskelet kök dosya sistemimiz, ete kemiğe bürünmüş olup, artık açılışa hazırdır. Okuyucu `$RISCV/src/sample/RootFS` altında oturan kök dosya sistemini, gezerek incelemelidir.

Şu anda kadar 4 dizin hariç kök dosya sistemi tamamlanmıştır. `/proc, /sys, /tmp` ve `/dev` dizinleri nasıl oluşturulacaktır?

`/dev` hariç bu dizinler, sistem ayağa kalktıktan sonra `/etc/rcS` betiği tarafından açılışta mount edilirler. Diğer bir deyişle bu dizinler aslında çekirdeğin içinde hazır olarak kurulmuşlardır.

`/etc/rcS` içinde verilen mount komutlarıyla, çekirdek içinde kurulan bu dizinler için user space tarafında bir kapı açılır. Bu kapıların girişleri `/proc, /sys` ve `/tmp` dizinleridir. Bu dizinlere girildiğinde, aslında çekirdek tarafındaki bilgilere veya dosya sistemlerine ulaşılır.

`/proc` ve `/sys` dosya sistemleri sözde dosya sistemleridir. Güç kesilince yok olurlar. `/proc` içinde proses bilgileri, `/sys` içinde sistem bilgileri vardır. Bu sözde dosya sistemlerinin esas görevi user space ile kernel space arasında veri alış verişini sağlamaktır.

`/tmp` dosya sistemi de, içi boş bir dosya sistemidir. Sanal bellekte açılır. Standard bir disk dosyası gibi davranır. Kapasitesi RAM'in tam yarısı kadardır. Ama ramdisk gibi bu kadar belleği hemen kendisine tahsis etmez. İhtiyaç oldukça büyür veya küçülür.

Açılış betiği bölümünde sözde dosya sistemlerinin bağlanması ayrıca incelenecektir.

1.8 Açılış betiği

Çekirdek açılışını tamamladıktan sonra `root=...` parametresi ile verilen cihazı kendisine kök dosya sistemi olarak bağlar. Daha sonra gözü kapalı bir biçimde `/sbin/init` programını yürütür. İlk çalışan program bu olduğu için PID numarası 1'dir. Diğer bütün programlar 1 numaralı bu prosesin çocukları olarak doğarlar.

`/sbin/init` başlar başlamaz `/etc/inittab` dosyasına göre açılışı yönetir. Bu dosya aşağıda verilmiştir.

```
$ cat $RISCV/src/sample/RootFS/etc/inittab
::sysinit:/etc/rcS

::respawn:/sbin/getty 115200 ttyS0

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

Bu dosya, standard olmayan, aşırı basitleştirilmiş bir busybox inittab dosyasıdır. `/sbin/init` programı bu dosyada en önce `sysinit` geçen satırı arar. Hemen karşısındaki programı çalıştırır. Bu örnekte `/etc/rcS` betiği çalışacaktır.

Çalışması bittikten sonra `respawn` olan satırları arar ve karşısındaki her bir programı arka planda çalıştırır. Eğer program durursa, otomatik olarak tekrar çalıştırır. Örnekte `getty` programı çalıştırılacaktır. Bu program seri kanal üzerinden `login:` sağlar ve bir terminal başlatır. `$ exit` ile login'den çıkılırsa getty programı son bulur. `/sbin/init` programı, getty'nın sonlandığını anlar ve ilgili satırdaki `respawn` kelimesi sayesinde getty programını tekrar başlatır. Böylece tekrar `login:` promptu gelir.

Burada `repsawn` yerine, örneğin `once` yazılsaydı `login:` promptu sadece bir kez gelecek ve `$ exit` yapıldıktan sonra bir daha seri kanaldan sisteme giriş yapılamayacaktı.

En genel kullanım için bakınız `http://belgeler.gen.tr/man/man5/man5-inittab.html`

Eğer `shutdown, halt` veya `reboot` gibi komutlar girilirse, `/sbin/init` programı `shutdown` kelimelerinin bulunduğu satırları sıra ile çalıştırır.

Örneğimizde önce `ifdown -a` ile ağ yapılandırması sonlandırılır. Sonra `sync` ile blok cihazların ara belleklerinin boşaltılması sağlanır. Sonra da `umount -a -r` ile mount edilmiş bütün dosyaların unmount edilmesi sağlanır.

Fakat, en azından `/sbin/init` programının çalıştığı disk bölümü unmount edilemez. Çünkü, bu bölüm üzerinde halen `/sbin/init` programı çalışmaktadır ve umount komutu `device busy` hatası alır. İşte `-r` seçeneği bu gibi durumlarda, ilgili disk bölümünü `read-only` olarak tekrar mount eder. Bu adımdan sonra gücü kesmenin, sisteme bir zararı olmayacaktır.

Yukarıda, `/etc/rcS` betiğinin çalışacağını söylemiştik. Bu betik aşağıda verilmiştir. Kolay anlatabilmek için satır başlarına numara eklenmiştir. Numaralar, orjinal betikte mevcut değildir.

```
$ cat $RISCV/src/sample/RootFS/etc/rcS

 1 #!/bin/sh +x
 2 # /etc/rcS

 3 export PATH=/sbin:/bin:/usr/sbin:/usr/bin

 4 mount -t tmpfs  tmpfs  /tmp -o noexec,nosuid,nodev
 5 mount -t sysfs  sysfs  /sys
 6 mount -t proc   proc   /proc

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

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

11 mount -o noatime -t tmpfs  tmpfs /mnt
12 mkdir /mnt/nfs

13 mount -t tmpfs tmpfs /var
14 mkdir /var/log
15 mkdir /var/run
16 mkdir /var/tmp
17 mkdir /var/www
18 mkdir /var/cache

19 hostname UCanLinux

20 syslogd
21 klogd

22 cp /root/index.html /var/www/
23 httpd -h /var/www

24 telnetd -f /etc/issue

25 ifup -a
```

1. satırda, bu betiğin hangi yorumlayıcı ile yürütüleceği yazılır. Bu örnek betik `/bin/sh` programı tarafından yürütülecektir. `#!` ile başlayan bu gösterim tarzına `shebang` denir. Satır sonundaki `+x` işareti, `debug off` demektir.

Bu işaret `-x` yapılırsa, bir satır çalışmadan önce ekrana yazılır sonra çalışır. İlk testlerde `-x` kullanılması tavsiye edilir. Debug ara satırlarda da açılıp kapatılabilir. İlla ki `shebang` ifadesinde kullanılması gerekmez. Debug özelliğini aralarda açıp kapatmak için `set -x` ve `set +x` kullanılabilir.

2. satır açıklama satırıdır. Dikkat edilirse, `shebang` satırı da açıklama gibi başlar. Ama önünde `!` bulunduğu için açıklama gibi değerlendirilmez.

3. satırda, bu betiğin içinde işleyecek komutların PATH bilgisi vardır. Busybox derlenirken, sadece bu betikteki ve `/etc/inittab`'daki komutların `busybox menuconfig` ile seçilmesi yeterli olacaktır. Diğer bir deyişle, busybox içine 400 komut değil de, sadece bu betik ve inittab'da kullanılan `sh, mkdir, mount, syslogd, ...` gibi birkaç komutu seçmek yeterli olacaktır.

Tabii ki uygulama programlarının içinde bulunabilecek system() veya exec() çağrılarında kullanılabilecek komutlar ve diğer betiklerin içinde olabilecek komutlar da ayrıca dikkate alınmalıdır.

4. satırda yazılabilir, geçici bir dosya sistemi bağlanmaktır. Bu tür dosya sistemleri hiç bir zaman kurulmaz, doğrudan mount edilirler.

Unutulmamlıdır ki dosya sistemleri önce `mkfs.ext4 ...` gibi bir komutla kurulur sonra mount edilirler. Fakat `tmpfs` dosya sistemi çekirdek tarafından hazır olarak kuruludur. Sadece mount edilir.

5 ve 6. satırlar, çekirdek içinde bulunan sözde dosya sistemlerini bağlarlar. Böylece çekirdekteki bu dosya sistemleri, user space tarafında görünür, erişilebilir hale gelirler. Unutulmamalıdır ki buradaki iki mount komutu verilmese bile, bu dosya sistemleri çekirdek içinde her zaman mevcuttur. Sadece user space tarafında görünür olmazlar.

Buradaki iki mount komutu iptal edilirse, sistem yine açılır. Ama pek çok Linux komutu, bu iki dosya sistemini kullanacağı için, hata verecektir.

7 ve 8. satırlar sözde terminaller için dosya sistemi kurarlar. Uzaktan yapılan, telnet veya ssh gibi bağlantılarda her bağlantı için bir terminal ve ekran temin edilmelidir. Bu dosya sistemi, uzak bağlantılar için sözde terminal sağlarlar.

Bir gömülü sisteme telnet veya ssh ile bağlandıktan sonra bir türlü `login:` lafı gelmiyorsa, muhtemel sebep bu dosya sisteminin mount edilmemiş olmasıdır. Eğer sisteme uzaktan bağlantı yapılmayacaksa bu iki satıra gerek yoktur.

9 ve 10. satırlar aynen tmpfs gibi bir dosya sistemi sağlar. Sadece bağlantı noktası[28] `/dev/shm` şeklinde standard bir isimdir. Bazı programlar doğrudan bu ismi ararlar. `/tmp` ile tamamen aynıdır. Uçucu bir dosya sistemidir. Güç kesilince tutulan bilgiler yok olur.

11. satırda yine tmpfs dosya sistemi bağlanır. Bazı programlar doğrudan `/mnt` dizinini arar. Genelde mount point noktası olarak bu dizin kullanılır. `/tmp` ve `/dev/shm` ile birebir aynı özelliklere sahiptir. Bazı sistemlerde sadece `/tmp` bağlanır ve `/dev/shm` ile `/mnt` buraya sembolik link ile bağlanır. Bu tür kullanım tarzı da doğrudur.

12. satırda ise, test sırasında kullanılacak bir dizin yaratılmaktadır. 13 ile 18 arasındaki satırlar, bazı programların aradığı dizinleri yaratır.

19. satırda makine ismi doğrudan tanıtılır. Bazı uygulamalarda bu komut `hostname -F /etc/hostname` şeklinde verilir. Sonra da `/etc/hostname` dosyası içine makinenin hostname bilgisi yazılır. `-F` tarzı kullanım daha uygundur. Basit olduğu için biz doğrudan ismi verdik.

20. satırda, syslogd[29]programı başlatılır. Bu program çekirdek veya diğer uygulamalardan gelen log bilgilerini `/var/log/messages` dosyasına yazar. Bir tür sistem kayıtçısıdır.

C programları içinden `syslog()` çağrısı ile, betiklerden de `logger` komutu ile sistem kayıtçısına bilgi gönderilebilir.

Bu program başlatılmazsa, çekirdek mesajları ve log bilgileri tutulmaz.

21. satırdan çekirdek mesajlarını toplayan program başlatılır. Normalde çekirdek mesajları syslogd tarafından toplanır. Fakat çekirdek, user space tarafında çalışan syslogd programına doğrudan veri göndermez. Bunun yerine `klogd`[30] programı, `klogctl()` sistem çağrısı ile veya `/dev/klog` dosyasından ki aslında bir pipe'tır, çekirdek log bilgilerini okur ve `syslogd` programına aktarır. Böylece syslogd programı da çekirdek mesajlarını sanki sıradan bir programdan geliyormuş gibi `/var/log/messages` dosyasında biriktirir. `syslogd` komutunun varsayılan değerleri, `busybox`'ın menuconfig ekranlarında incelenebilir.

22. satırda örnek bir web sayfası, `/var/www` altına kopyalanır. 23. satırda ise

[28]mount point
[29]system logger daemon
[30]kernel logger daemon

bir web sunucusu başlatılır. Dışarıdan,bir browser'dan `http://localhost:8080` girişi ile bağlantı yapılabilir.

24. satırda telnet sunucusu başaltılır. Host makineden RiscV'e `$ telnet localhost 2323` komutu ile bağlantı yapılabilir. Bizler çok basit olduğu için telnet'i seçtik. Çoğu uygulama ssh ile bağlantı yapar. Busybox'daki ssh sunucusun adı dropbear'dir.

Kullanıcının telnet veya http sunucusa ihtiyacı yoksa, 22,23 ve 24. satırları silebilir.

Son olarak 25. satırda ağ yapılandırılır. `ifup` komutu `/etc/network/interfaces` içindeki tanımları kullanarak ip ve route tanımlarını yapar. Örnek `interfaces` dosyası aşağıda verilmiştir.

```
$ cat $RISCV/src/sample/RootFS/etc/network/interfaces

auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
  address     10.0.2.20
  netmask 255.255.255.0
  broadcast  10.0.2.255
  gateway      10.0.2.2
```

`eth0`'ın ip bilgisi aynen yukarıdaki gibi verilmelidir. Çünkü bu IP değerleri daha sonra qemu ile test sırasında kullanılacaktır. Daha sonra okuyucu istediği IP değerini verebilir.

Pek çok IP verme ve ağ yapılandırma yöntemi vardır. Bu örnek çalışmamızda çok basit olduğu için statik IP kullanılmıştır.

`RootFS` altında oturan kök dosya sistemini kullanabilmek için, en azından bir diske yazmamız gerekir. Fakat hayatımız sanal olduğu için diskimiz de sanal olacaktır.

Sonraki bölümde `RootFS` altında oturan dosya sistemi, sanal bir diske taşınarak imaj oluşturulacaktır.

1.9 İmaj oluşturma

Bu çalışmada 64MB'lık[31] bir disk kurulacaktır.

Okuyucu amacına göre diskin kapasitesini güncelleyebilir. Salt-okunur bağlanmış kök dosya sistemlerinde, hemen hemen %100 doluluk olabilir. Çünkü sistem salt-okunur olduğu için hiç bir zaman genişlemeyecektir. Tabii ki güncellemeleri hariç tutmaktayız.

64MB'lik sanal bir disk kurmak için öncelikle içi tamamen sıfırlarla dolu bir dosya yaratmak gerekir. Aşağıdaki `dd` komutu ile içi tamamen sıfırlarla dolu bir dosya kurulur. Buradaki sıfır ifadesi, binary sıfırı gösterir, ASCII sıfır değildir.

```
$ cd $RISCV/src/sample

# için sıfırlarla dolu dosya yarat
$ dd if=/dev/zero of=riscv_disk bs=1M count=64

$ ls -l riscv_disk
-rw-rw-r-- 1 nazim nazim 67108864 Haz  7 00:46 riscv_disk
```

`dd` komutu `device dump` demektir, kopyalama yapar. Bu komut, güvenlik önlemi olarak, mümkünse `root` kullanıcısı ile kullanılmamalıdır. Kazara `dd` komutu `device destroy` gibi çalışabilir, dikkat edilmelidir.

`/dev/zero` cihazı da sonsuz sayıda sıfır üretir.

`if`, input file, `of` output file demektir. `of` ile verilen `riscv_disk` ismi tamamen keyfidir.

`bs` ile block size değeri verilir.

`count` ile block adedi verilir. Sonuçta `bs * count` adet sıfır üretilecek ve `riscv_disk` içine yazılacaktır.

Sonraki adımda bu dosya içine, sanki fiziksel bir cihazmış gibi dosya sistemi kurulur.

[31] Buradaki örnek diskimiz 64MB boyundadır. Fakat github'daki disk 50MB boyundadır. Çünkü github 50MB'tan büyük dosyalara ikaz vermektedir.

```
$ cd $RISCV/src/sample

# dosya sistemi kur

$ mkfs.ext2 -L riscv-rootfs riscv_disk
```

Bizler az yer kapladığı için ext2 dosya sistemi seçtik. Okuyucu ext3 veya ext4 de kullanabilir. Kök dosya sistemi ext2 olacaksa, asla `rw` bağlanmamalıdır. ext2 dosya sistemi ani kapanmaya karşı hiç dayanıklı değildir. Anında bozulur. Fakat `ro` bağlanırsa sorun çıkmayacaktır.

`-L` seçeneği ile disk ismi verilmiştir. Tamamen kozmetiktir. Bu isimler `/etc/fstab` içinde kullanılabilir. Fakat bu dosya örnek sistemimizde kullanılmayacaktır.

Kök dosya sistemiş illa ki rw bağlanacaksa mutlaka ext3 veya ext4 seçilmelidir. Bu dosya sistemleri log tabalı olup ani kapanmalara karış çok dayanıklıdır.

`riscv_disc` sıradan dosyası içinde artık bir dosya sistemi vardır. Bu dosya sistemine erişmek için aşağıdaki gibi mount yapılır ve `RootFS` içindeki bütün bilgiler `riscv_disk` içine kopyalanır.

```
$ cd $RISCV/src/sample

# boş diski /mnt/rootfs dizinine bağla

$ sudo mkdir /mnt/rootfs
$ sudo mount riscv_disk /mnt/rootfs

# RootFS'in içeriğini boş diske kopyala
$ sudo cp -a RootFS/* /mnt/rootfs

# sahiplikleri değiş
# -h ile link'lerin de sahipliği değiştirilebilir

$ sudo chown -R -h root:root /mnt/rootfs

# diskin %57'si kullanılıyor.
# disk ro olarak bağlanacağı için gereksiz yere boş yer kalmıştır

$ df /mnt/rootfs
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/loop0         63461 34185     26000  57% /mnt/rootfs

$ sudo umount /mnt/rootfs
```

```
# sanal diski incele

$ file riscv_disk
riscv_disk: Linux rev 1.0 ext2 filesystem data,
            UUID=60f74e31-1264-4da8-aa58-84c4d0000c1c,
            volume name "riscv-rootfs" (large files)
```

Artık sanal diskimiz qemu tarafından kullanılmaya hazırdır.

`$ df /mnt/rootfs` çıkışına dikkat edilirse, `/mnt/rootfs` dizinine bağlanan cihazın adı `/dev/loop0` olarak geçmektedir. `riscv_disk` isimli sanal diskimiz gözükmemektedir.

Gerçek bir SCSI veya IDE disk gibi bir blok cihaza, Linux kernel tarafından erişileceği zaman, cihazdan talep edilecek bilgi, örneğin 123. sektörün içeriği gibi, aradaki donanıma veya kontrol kartına gönderilir. Aradaki kontrol kartı, 123. sektörü fiziksel diskten okur ve çekirdeğe bir biçimde aktarır.

Fakat sanal disklerde arada bir kontrol kartı veya diski yöneten bir donanım mevcut değildir. İşte bu durumda `/dev/loop`[32] cihazları devreye girer. Bu cihazlar sanal disklerin kontrol kartı gibi davranır. Bu cihazlara, bir tür "feyk kontrol kartı" gibi bakılabilir.

`$ sudo mount riscv_disk /mnt/rootfs` komutu girildiğinde, mount komutu `riscv_disk` cihaz isminin gerçek bir cihaz olmadığını, sıradan bir dosya olduğunu anlar. Bunun için isimlere ait mod bilgisinin ilk karakterine bakar.

```
$ cd $RISCV/src/sample

$ ls -l riscv_disk
-rw-rw-r-- 1 nazim nazim 67108864 Haz  7 12:10 riscv_disk

$ ls -l /dev/sda
brw-rw---- 1 root disk 8, 0 Haz  9 09:44 /dev/sda
```

Yukarıdaki ilk `ls` komutunda, `riscv_disk` isminin mod bilgisi `-rw-rw-r--` şeklindedir. İlk karakter `-` olarak gözükür. Bu da bu ismin sıradan bir dosyayı temsil ettiğini gösterir.

[32] $ man 4 loop

Fiziksel, gerçek bir blok cihaz olsaydı, mod bilgisinin 1. karakteri, yukarıdaki `/dev/sda` modunda görülebileceği gibi `b` olacaktı.

`mount` komutu, bağlanacak cihazı, sıradan bir dosya olarak tespit ettikten sonra, bu cihaza hemen bir feyk kontrol kartı, feyk bir cihaz atar. İşte bu feyk cihazlara "loopback device" denir. Loopback cihazların görevleri, sıradan dosyaları gerçek bir cihazmış gibi çekirdeğe göstermektir.

loopback isminin, ağ yapılandırmasında kullanılan ve localhost'u veya 127.0.0.1 adresini gösteren loopback device ile bir ilgisi yoktur.

`mount` komutu hemen boş bir loopback cihazı numarası tespit eder. Başka hiç bir program loopback cihazlarını kullanmıyorsa, numara sıfırdan başlar. Bu durumda örnek dosyamız için feyk cihaz olarak `/dev/loop0` kullanılacaktır. İşte mount komutunun çıkışında görülen `/dev/loop0` adı buradan gelmektedir.

Çekirdeğin loop device desteğine sahip olabilmesi için çekirdek derlemesi sırasında aşağıdaki gibi uygun seçeneklerin işaretlenmiş olması gerekir.

```
Device Drivers  --->
  [*] Block devices  --->
      <*>   Loopback device support
      (8)     Number of loop devices to pre-create at init time
```

Bu örnekte görüleceği gibi açılışta en fazla 8 adet loopback cihazı üretilecektir. İhityaç oldukça cihaz adedi, çerkidek derlemesi aşamasında veya sistem açıldıktan sonra artırılabilir. Loopback cihazlarının isimleri aşağıdaki gibi host tarafında listelenebilir.

```
$ ls -l /dev/loop?
brw-rw---- 1 root disk 7, 0 Haz 11 17:08 /dev/loop0
brw-rw---- 1 root disk 7, 1 Haz 11 17:07 /dev/loop1
brw-rw---- 1 root disk 7, 2 Haz 11 17:07 /dev/loop2
brw-rw---- 1 root disk 7, 3 Haz 11 17:07 /dev/loop3
brw-rw---- 1 root disk 7, 4 Haz 11 17:07 /dev/loop4
brw-rw---- 1 root disk 7, 5 Haz 11 17:07 /dev/loop5
brw-rw---- 1 root disk 7, 6 Haz 11 17:07 /dev/loop6
brw-rw---- 1 root disk 7, 7 Haz 11 17:07 /dev/loop7
```

RiscV makinesi ayağa kalktıktan sonra, aynı komut girilebilir. Yine aynı sonuç üretilecektir.

Dikkat edilirse, mod bilgisinin başındaki `b` karakterinden bu cihazların blok cihaz olduğu anlaşılabilir. Çekirdek, artık bütün giriş/çıkış işlemlerinde `riscv_disc` dosyası yerine `/dev/loop0` cihazını kullanacaktır. `/dev/loop0`'a yazılan bütün bilgiler, `riscv_disk` dosyasına gidecektir.

`/dev/loop0` cihazı sanki bir kontrol kartı gibi, giriş/çıkış işlemlerini `riscv_disk` dosyası üzerinde gerçekleyecektir. `/dev/loop0` cihazının çekirdek ile sıradan dosya arasında oturduğu düşünülebilir.

`riscv_disc` dosyasının ilişikili olduğu loopback cihazının adı, `losetup`[33] komutu ile aşağıdaki gibi bulunabilir.

```
$ sudo mount riscv_disk /mnt/rootfs

$ losetup -a
/dev/loop0: [2051]:5656764 (/opt/riscv/src/sample/riscv_disk)

$ sudo umount /mnt/rootfs
```

`/dev/loop0` ilişkili isim, `losetup` dışında, `$ cat /sys/block/loop0/loop/backing_file` komutu ile de `/proc` dosya sisteminden elde edilebilir.

Şu anda elimizde hem busybox hem de çekirdeğin config dosyaları mevcuttur. Bu dosyaların yedeklenmesi faydalı olacaktır. Bizler config dosyalarını `$RISCV/src/configs` altına, `busybox.config` ve `kernel.config` olarak yedeklemekteyiz.

[33] $ man losetup

1.10 Busybox nasıl çalışır ?

```
int mkdir_main(int argc, char **argv){
   // burada mkdir komutunun kodu olacaktır.
}


int mkdir_main(int argc, char **argv){
   // burada ls komutunun kodu olacaktır.
}

int main(int argc, char **argv){
   ...
   if ( argv[0]== "mkdir" ) return mkdir_main(argc, argv);
   if ( argv[0]== "ls" )    return ls_main(argc, argv);
   ...
}
```

Yukarıdaki sözde kod[34] busybox'ın içinde bulunan `mkdir` ve `ls` komutlarının yürütme yöntemini temsil eder.

Yukarıdaki gibi bir kod derlenince, bütün komutlar tek bir dosya içinde birikmiş olur. Derleme bitikten sonra aşağıdaki gibi, her bir komut için sembolik linkler kurulur.

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

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

Örneğin, `$ mkdir foo` komutu girildiği zaman sembolik link sayesinde busybox programı çalışır. `if ( argv[0]== "mkdir" ) ...` kodu sayesinde busybox içindeki `mkdir` komutunun çağrılması gerektiği anlaşılır.

C programlarında `argv[0]` elemanı programın adını gösterir. Fakat ilginç bir biçimde, sembolik linklerde, `argv[0]` elemanı, program adı olarak, çalışan

[34] pseudo code

programı değil, sembolik link'in kendisini gösterir. Böylece `busybox` programı, kendisine hangi sembolik ile gelindiğini anlar ve buna göre uygun fonksiyonu çağırır. Her fonksiyon da ilgili komutu kısıtlı bir biçimde icra eder.

1.11 dd nasıl çalışır ?

Gömülü sistemciler için `dd` komutu çok kullanışlıdır. Bu komut basit bir mantığa sahiptir. Bir dosyayı okur ve başka bir dosyaya yazar. `cp` komutundan farklı olarak mount edilmemiş cihazları, blok blok okuyabilir. `cp` komutu, sadece mount edilmiş dosya hiyerarşisini takip ederek çalışabilir.

`$ dd if=/dev/zero of=riscv_disk bs=1M count=64` komutunun sözde kodu aşağıdaki gibi olabilir. Sözde kod çok basit olduğu için ayrıca üzerinde durulmayacaktır. Genel bir fikir vermesi için yazılmıştır.

```
# dd if=/dev/zero of=riscv_disk bs=1M count=64
# komutu, mealen aşağıdaki gibi ifade edilebilir
#
# if, input file
# of, output file

int if= open("/dev/zero",  O_RDONLY); // if=/dev/zero
int of= open("riscv_disk", O_WRONLY); // of=riscv_disk

char bs[1024*1024];                   // bs=1M
int count= 64;                        // count=64
int n;

for(i= 0; i< count; i++){
   n= read(if, bs, sizeof(bs));  // if'den bs[] içine n<=1M adet 0 oku
   write(of, bs, n);             // okunan sıfırları of dosyasına yaz
}
```

Bölüm 2

Qemu ile test

Hacimli miktarda üretilemediği için RiscV makineleri günümüzde aşırı pahalıdır. Bundan dolayı kurmuş olduğumuz GNU/Linux sistemi emülatör altında test edilecektir. Emülatör olarak qemu kullanılacaktır.

Qemu[1] bir makine emülatörü ve sanallaştırma makinesidir. Qemu, "quick emulator" demektir. Baştaki Q harfi, "Quick" kelimesinden gelmektedir.

RiscV toolchain indirildiğinde qemu programının kaynak kodu da beraber gelmektedir. Fakat bizler qemu'nun ana sitesinden programı indireceğiz. Böyle tavsiye edilmektedir, bizler de uyacağız.

Qemu sistemi aşağıdaki gibi indirilir, derlenir ve kurulur.

```
$ cd $RISCV/src

# qemu indir
$ git clone https://git.qemu.org/git/qemu.git

# yapılandır
$ cd qemu
$ ./configure --target-list=riscv64-softmmu,riscv32-softmmu --prefix=$RISCV

# derle ve $RISCV altına kur
$ make -j$(nproc)
$ make install
```

[1] https://www.qemu.org

Yukarıdaki `.configure` seçeneği sayesinde, hem 32 hem de 64 bit için qemu programı üretilmiştir. Üretilen programlar `$RISCV/bin` altına atılırlar. Bu dizin `.bashrc` içine aşağıdaki gibi eklenemelidir.

```
$ vi ~/.bashrc

# son satıra aşağıdaki satırı ekle
export PATH=$RISCV/bin:$PATH

# terminalden çık, yeni terminal aç ki
# PATH tanımı etkinleşsin
```

Kurulumun doğruluğunu kontrol için aşağıdaki komut girilebilir.

```
$ cd
$ qemu-system-riscv64  -machine help

Supported machines are:
none                 empty machine
sifive_e             RISC-V Board compatible with SiFive E SDK
sifive_u             RISC-V Board compatible with SiFive U SDK
spike_v1.10          RISC-V Spike Board (Privileged ISA v1.10) (default)
spike_v1.9.1         RISC-V Spike Board (Privileged ISA v1.9.1)
virt                 RISC-V VirtIO Board (Privileged ISA v1.10)
```

Bu belge yazılırken son sürüm 4.0.5 idi. Bu sürümün desteklediği, emüle edebileceği makinelerin adları, yukarıdaki gibi `-machine help` seçeneği ile listelenebilir.

Bu sürüm şu anda 6 farklı RiscV bordu desteklemektedir. `qemu-system-` öneki ile başlayan qemu komutları, işletim sistemi desteği vermektedir. Yani bir bordu ve altındaki işletim sistemini destekler. İşletim sistemi desteği olmadan, doğrudan bare-metal programları destekleyen qemu programları da vardır. Bunlar konumuz dışındadır.

Bizler test için `virt` bordunu kullanacağız. Bu bord, Mayıs 2017'de yayınlanan ISA v1.10 şartnamesini[2] desteklemektedir.

[2]https://riscv.org/specifications/privileged-isa/

2.1 Qemu'nun terminal destekli başlatılması

Önce kurduğumuz sistem, qemu ile test edilecektir. Sonraki bölümde de qemu'nun seçenekleri üzerinde çalışma yapılacaktır. Örnek sistemimiz aşağıdaki gibi test edilebilir.

```
# Sanal ortamda GNU/Linux testi

$ cd $RISCV/src/sample

$ cp $RISCV/riscv64-unknown-linux-gnu/bbl .

$ sudo qemu-system-riscv64          \
        -nographic                  \
        -machine virt               \
        -kernel bbl \
        -m 128                      \
        -append "root=/dev/vda ro"  \
        -drive file=riscv_disk,format=raw,id=hd0  \
        -device virtio-blk-device,drive=hd0       \
        -netdev user,id=net0,hostfwd=tcp::2323-10.0.2.20:23,hostfwd=tcp::8080-10.0.2.20:80 \
        -device virtio-net-device,netdev=net0

# veya kısaca
$ run.sh
```

Yukarıdaki test komutu, kolay okunması için pek çok satır şeklinde yazılmıştır. Okuyucu bu komutu bir dosya içine yazarak çalıştırabilir. Sonraki bölümde qemu seçenekleri üzerinde durulacaktır.

Çıkışın özeti aşağıda verilmiştir. Çok uzun satırlar kırpılmıştır. root kullanıcısı ile giriş yapılabilir. Şifresi root'tur. `exit` ile kullanıcından çıkılıp tekrar login'e düşülebilir. `poweroff` ile makine kapatılır.

```
bbl loader
 ____  _         __     __
|  _ \(_)___  __\ \   / /
| |_) | / __|/ __\ \ / /
|  _ <| \__ \ (__ \ V /
|_| \_\_|___/\___| \_/

INSTRUCTION SETS WANT TO BE FREE
```

```
...
[    0.000000] OF: fdt: Ignoring memory range 0x80000000 - 0x80200000
[    0.000000] Linux version 4.20.0+ (nazim@nkoc) (gcc version 8.3.0 (GCC)) #2 SMP
[    0.000000] printk: bootconsole [early0] enabled
[    0.000000] initrd not found or empty - disabling initrd
...
[    0.664000] EXT4-fs (vda): mounted filesystem without journal. Opts: (null)
[    0.664000] VFS: Mounted root (ext4 filesystem) readonly on device 254:0.
[    0.672000] devtmpfs: mounted
[    0.708000] Freeing unused kernel memory: 176K
[    0.708000] This architecture does not have kernel memory protection.
[    0.708000] Run /sbin/init as init process


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

UCanLinux login: root
Password: root

root@UCanLinux:~ # poweroff

umount: can't remount tmpfs read-only
umount: devtmpfs busy - remounted read-only
The system is going down NOW!
Sent SIGTERM to all processes
Sent SIGKILL to all processes
Requesting system poweroff
[   40.760000] reboot: Power down
Power off
$
```

"bbl loader" kelimesinden sonraki RiscV logosunu biz ekledik. Normalde burada RiscV'in kendi logosu vardı, pek kabaydı, beğenmedik, figlet ile yeniden yazdık. Bu logo `riscv-pk/build/` altında `bbl_logo_file` adı ile bulunabilir.

2.2 Qemu'nun GUI destekli başlatılması

İlk testimizde, qemu sistemi `-nographic` seçeneği ile başlatılmıştı. Genelde de böyle kullanılır.

Qemu sisteminin bir de kendisine has bir GUI ortamı vardır. Bu gui ortamında çalışmak için, qemu sistemi `-nographic` seçeneği olmadan başlatılır.

Gelen menüden, `View` → `Show tabs` işaretlenir ve `serial0` sekmesine geçilirse login ekranı gözükecektir. GUI bizim için zulümdür. Meraklıları inceleyebilir.

2.3 GNU/Linux Sisteminin incelenmesi

Qemu altında çalışan sisteme root/root ile login olduktan sonra aşağıdaki temel incelemeler yapılabilir.

2.3.1 Mevcut komutların listelenmesi

`$ busybox` komutu ile mevcut komutlar listelenebilir. Bütün komutlar, `/bin/busybox` içinde, tek bir program içinde toplanmışlardır.

```
root@UCanLinux:~ # busybox

BusyBox v1.30.1 (2019-05-30 08:03:16 +03) multi-call binary.
BusyBox is copyrighted by many authors between 1998-2015.
Licensed under GPLv2. See source distribution for detailed
copyright notices.

Usage: busybox [function [arguments]...]
   or: busybox --list[-full]
   or: busybox --show SCRIPT
   or: busybox --install [-s] [DIR]
   or: function [arguments]...

        BusyBox is a multi-call binary that combines many common Unix
        utilities into a single executable.  Most people will create a
        link to busybox for each function they wish to use and BusyBox
        will act like whatever it was invoked as.
```

```
Currently defined functions:
        [, [[, acpid, add-shell, addgroup, adduser, adjtimex, arch, arp,
        arping, ash, awk, base64, basename, bc, beep, blkdiscard, blkid,
        blockdev, bootchartd, brctl, bunzip2, bzcat, bzip2, cal, cat, chat,
        ubiattach, ubidetach, ubimkvol, ubirename, ubirmvol, ubirsvol,
        ...
        ubiupdatevol, udhcpc, udhcpd, udpsvd, uevent, umount, uname, unexpand,
        uniq, unix2dos, unlink, unlzma, unshare, unxz, unzip, uptime, users,
        usleep, uudecode, uuencode, vconfig, vi, vlock, volname, w, wall,
        watch, watchdog, wc, wget, which, who, whoami, whois, xargs, xxd, xz,
        xzcat, yes, zcat, zcip
```

Busybox sisteminde komutlar iki türlü verilebilir. Eğer sembolik varsa ki bizim örneğimizde mevcuttur, komutların doğrudan ismi yazılır, `$ ls -l` gibi.

Busybox komutları sembolik linklere bağlı olmadan da kullanılabilir. Bunu için önce `busybox` yazılır sonra da komut ve varsa argümanları verilir. Bu durumda sistemde hiç bir sembolik linke gerek olmayacaktır. Ama kullanımı pratik değildir, örneğin `$ busybox ls -l`

2.3.2 Proses ağacı

`$ pstree -p` komutu ile proses ağacı listelenebilir. İlk proses `init` prosesidir ve PID değeri 1 dir. Diğer bütün prosesler buradan doğarlar.

```
root@UCanLinux:~ # pstree -p

init(1)-+-httpd(67)
        |-klogd(64)
        |-sh(81)---pstree(89)
        |-syslogd(62)
        `-telnetd(69)
```

2.3.3 Çekirdeğin config dosyası

Çekirdeği derlerken kullanılan `.config` dosyası `/proc/config.gz` altında bulunabilir. Bu dosya aşağıdaki gibi incelenebilir.

```
$ cp /proc/config.gz /tmp
$ cd /tmp
$ gunzip config.gz
$ more config

# veya kısaca
$ zcat /proc/config.gz | more
```

`/proc` altında `config` bilgisinin oluşturulabilmesi için, çekirdek derlemesi sırasında, menuconfig ekranlarından aşağıdaki seçimler yapılmıştır.

```
  General Setup -->
     <*> Kernel .config support
     [*]   Enable access to .config through /proc/config.gz
```

Müşteriye verilecek sistemlerde bu özellikler kapatılabilir. Böylece çekirdek biraz küçülür. Test ve geliştirme sistemlerinde açık olması, yararlı olacaktır.

2.3.4 Kök dosya sistemini rw bağlamak

Ani kapanmalara karşı dayanıklı olması için kök dosya sistemi ro bağlanmıştır. Aşağıdaki basit test ile kök dosya sistemine yazım yapılamayacağı görülebilir.

```
$ cd
$ mkdir hede
mkdir: can't create directory 'hede': Read-only file system
```

Yine de kök dosya sistemi üzerinde güncelleme yapılmak istenirse, aşağıdaki gibi, önce rw bağlanır. Gerekli güncellemeler yapıldıktan sonra tekrar ro bağlanır.

```
$ mount -o remount -o rw /
$ # burada gerek işleri yap
$ mount -o remount -o ro /
```

2.3.5 Uzaktan telnet bağlantısı

Host makineden, aşağıdaki gibi telnet ile RiscV makineye login yapılabilir.

```
# host tarafından telnet ile bağlan

$ telnet localhost 2323
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

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

  UCanLinux login:
```

Qemu tarafında "user space" ağ yapılandırılması kullanılmıştır. Bundan dolayı host makineden, RiscV makinesine erişim için her zaman localhost adı veya 127.0.0.1 adresi kullanılır. RiscV makinesine gelen paketler, portlar yardımı ile uygulamalara yönlendirilirler. Sonraki bölümde bu konuya değinilecektir.

2.3.6 Uzaktan web browser ile bağlanma

Host makinesinde bulunan herhangi bir web browser'dan `http://localhost:8080` girişi ile RiscV makineye web üzerinden erişilebilir. `/var/www/index.html` gösterilecektir.

2.3.7 Disk free

`df` komutu ile dosya sistemlerinin kullanım oranları, cihaz adları ve bağlantı noktaları incelenebilir.

```
# RiscV tarafındayız
```

```
root@UCanLinux:~ # df -a
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/root                63461     34185     26000  57% /
devtmpfs                 91064         0     91064   0% /dev
tmpfs                    91152        68     91084   0% /tmp
sysfs                        0         0         0   0% /sys
proc                         0         0         0   0% /proc
devpts                       0         0         0   0% /dev/pts
tmpfs                    91152         0     91152   0% /dev/shm
tmpfs                    91152         0     91152   0% /mnt
tmpfs                    91152        36     91116   0% /var
```

İlk dosya sistemi `/dev/root` olarak gözükmektedir. Bu dosya sistemini çekirdek açılışta bağlar. Çünkü `-append` komutu ile, çekirdeğe kök dosya sisteminin `/dev/vda` diskinde olduğu, `-append "root=/dev/vda ro"` şekilinde verilmiştir.

Eğer burada "ro" seçeneği verilmeseydi, kök dosya sistemi rw bağlanacaktır. Bu da pek sağlıklı bir durum değildir.

İkinci dosya sistemi de yine çekirdek tarafından `/dev` altına bağlanmıştır. Bu bağlama işleminin otomatik olarak yapılabilmesi için çekirdekte aşağıdaki seçimler yapılmıştır.

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

Çekirdek, kök dosya sistemini mount ettikten hemen sonra `/dev` dosya sistemini bağlar. Bu işlem, aşağıdaki açılış mesajında gösterildiği gibi raporlanır. Satır başındaki rakamlar tarafımızdan eklenmiştir, çıkışa dahil değildir.

```
root@UCanLinux:~ # dmesg|tail
  ...
1 [    0.632000] EXT4-fs (vda): mounted filesystem without journal. Opts: (null)
2 [    0.632000] VFS: Mounted root (ext4 filesystem) readonly on device 254:0.
3 [    0.656000] devtmpfs: mounted
4 [    0.688000] Freeing unused kernel memory: 176K
5 [    0.688000] This architecture does not have kernel memory protection.
6 [    0.688000] Run /sbin/init as init process
```

1 ve 2. satırda kök dosya sisteminin ext4 olarak, ro bağlandığı rapor edilir. 3. satırda ise device dosya sisteminin mount edildiği söylenir. İşte `/dev` dizinine mount işlemi tam burada yapılır.

6. satırda da ilk çalışan programın `/sbin/init` olduğu belirtilir. İlk çalışacak program, `-append` satırında, çekirdek parametresi olarak da verilebilir. Örneğin `-append "root=/dev/vda ro init=/bin/hede"` girilirse, ilk çalışacak program `init=...` ile gösterilen `/bin/hede` olacaktır.

`df` çıkışında görülen, `/` ve `/dev` dışındaki bütün mount işlemleri `/etc/rcS` içinden yapılmıştır.

2.3.8 NFS ile bağlantı

Host makinesindeki bir dizin, RiscV makinesi ile paylaştırılabilir. Bunun için host tarafında NFS server kurulu olmalıdır. Ubuntu 16.04'de NFS server yüklemek için `$ sudo apt-get install nfs-kernel-server` yeterlidir.

`$ service nfs-kernel-server start` ile de sunucu ayağa kaldırılır.

`$RISCV/src/sample` dizini RiscV makinesi ile aşağıdaki gibi paylaştırılabilir. Öncelikle paylaştırılacak dizin, host tarafında, `/etc/exports` içine aşağıdaki formatta yazılmalıdır.

```
# host tarafındayız
$ cat /etc/exports
/opt/riscv/src/sample *(rw,no_subtree_check,insecure,no_root_squash)
```

`/etc/exports` içine yazılan satırı kısaca inceleyecek olursak,

/opt/riscv/src/sample paylaşılacak dizinin adı.

* bu dizini herkes bağlayabilir, paylaşabilir.

rw uzaktaki makine bu dizini okuyup yazabilir.

no_subtree_check link veya mount point'lerle dizin dışına çıkılırsa, karışma, her yere erişmeye izin ver.

insecure gelen port numarası 1024'ten büyükse, yani "secure port" değilse bile kabul et.

no_root_suqash karşı tarafın root yetkisi ile iş yapmasına izin ver.

Tahmin edilebileceği gibi bu yapılandırma hiç ama hiç güvenli değildir. Ama biz bize çalışıldığı için sorun olmayacaktır.

Bu yapılandırmadan NFS sunucusunun haberi yoktur. Haber vermek için, yine host tarafında aşağıdaki gibi `exportfs` komutunun girilmesi yeterlidir.

```
# host tarafında
$ sudo exportfs -avr
exporting *:/opt/riscv/src/sample
```

`/etc/exports` dosyası her güncellendiğinde, `exportfs` komutu kullanılmalıdır ki NFS sunucusu yapılan değişiklerden haberdar olsun.

Bütün makineler, paylaşılan `/opt/riscv/src/sample` dizinini, artık uzaktan mount edebilirler. Şimdi RiscV makinesine geçip, aşağıdaki gibi paylaşılan dizini mount edebiliriz. Bu örnekte host makinenin IP değeri 192.168.1.17'dir.

```
# RiscV makinesindeyiz

root@UCanLinux:~ # mount -t nfs -o nolock 192.168.1.17:/opt/riscv/src/sample /mnt/nfs

root@UCanLinux:~ # df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/root                63461     34185     26000  57% /
devtmpfs                 91064         0     91064   0% /dev
tmpfs                    91152        68     91084   0% /tmp
tmpfs                    91152         0     91152   0% /dev/shm
tmpfs                    91152         0     91152   0% /mnt
tmpfs                    91152        36     91116   0% /var
192.168.1.17:/opt/riscv/src/sample
                     247891968 201715200  33561600  86% /mnt/nfs
```

`df` çıkışının son satırından görülebileceği gibi 192.168.1.17 makinesinde, yani benim host makinemde, `/opt/riscv/src/sample` dizini mount edilmiştir.

`/mnt/nfs` dizini, açılış sırasında, `/etc/rcS` betiği tarafından yaratılmıştı. `mnt` dizini genelde mount işlemlerinde, boş dizin yaratmak için kullanılır. Ama zorunlu değildir. Kullanıcı isterse, herhangi boş bir dizini, mkdir ile yaratıp, mount komutunu yeniden verebilir.

NFS ile dizin paylaşımının çok basit bir yorumu vadır. RiscV makinede `/mnt/nfs` dizini ile host makinede `/opt/riscv/src/sample` dizini aynı yeri gösterir. Birinde yapılacak bir güncelleme diğerinde gözükecektir.

Daha önceki `hello.c` örneğimizi çapraz derleyip, NFS altında test edelim. Öncelikle host tarafında, `hello.c`, aşağıdaki gibi çapraz derlenir.

```
# host tarafındayız

$ cd $RISCV/src/sample

# hello.c'yi çapraz derle
$ riscv64-unknown-linux-gnu-gcc -Wall -o hello hello.c

$ ls -l hello
-rwxrwxr-x 1 nazim nazim 11696 Haz  7 16:45 hello

$ file hello
hello: ELF 64-bit LSB executable, UCB RISC-V, version 1 (SYSV), dynamically linked,
       interpreter /lib/ld-, for GNU/Linux 3.0.0, not stripped

# RiscV kodunu x86'da çalıştırmayı dene

$ ./hello
bash: ./hello: cannot execute binary file: Exec format error
```

Sonra RiscV tarafında, `/mnt/nfs` altında `hello` kodunu doğrudan çalıştırabilir. Dikkat edelim, herhangi bir taşıma veya kopyalama yapılmamıştır. Dizin paylaşıldığı için host tarafındaki değişiklik RiscV tarafında da algılanmaktadır. Bu algılamayı yapan sistem NFS'in ta kendisidir.

```
# RiscV tarafındayız

root@UCanLinux:~ # cd /mnt/nfs

root@UCanLinux:/mnt/nfs # ls -l

total 16
```

```
-rwxrwxr-x    1 1000     1000         11696 Jun  7  2019 hello
-rw-rw-r--    1 1000     1000            85 Jun  7  2019 hello.c

# programı çalıştır

root@UCanLinux:/mnt/nfs # ./hello
hello RiscV

root@UCanLinux:/mnt/nfs #
```

Program çalışmış ve "hello RiscV" lafını ekrana yazmıştır.

Dikkat edilirse, `ls -l` çıkışında sahiplik ve grup adları 1000 olarak gözükmektedir. Sebebi basittir. RiscV tarafında `nazim` kullanıcısı mevcut olmadığı için doğrudan owner ve group numaraları kullanılmıştır.

RiscV tarafında işimiz bitince `$ umount /mnt/nfs` ile bağlantı koparılır.

Eğer bağlantı koparılmadan poweroff yapılırsa `/etc/inittab`'daki `::shutdown:/bin/umount -a -r` satırındaki `umount` komutu sayesinde, bağlı bütün dizinler koparılacaktır.

2.3.9 Dinamik yükleyici/bağlayıcı

Kök dosya sistemini kurarken `ld.so` programından bahsedilmişti. Tekrar edecek olursak, bu programa dinamik bağlayıcı ve yükleyici denir. Bu program aynı zamanda bir kütüphanedir ve başka hiç bir kütüphaneye bağımlı değildir. Esas görevi programları, bağlı kütüphaneleri yüklemek, adları çözmek ve programı çalışmaya hazır hale getirip, çalıştırmaktır.

Bu programın adı genelde ld ile başlar. Her dağıtımıda ismi farklı olabilir. Derlenen program `file` komutu ile incelenirse, `interpreter` kelimesinin karşısında bu programın veya kütüphanenin ismi görülebilir.

Bu program, toolchain derlemesi sırasında üretilir. Bizim RiscV makinesinde bu programın tam adı `/lib/ld-2.26.so` olarak geçer. Kısaca `ld.so` denir. Bu isim her toolchain için farklı olabilir. Hatta aynı toolchain'in 32 veya 64 bitlik sürümlerinde bile farklı olabilir.

`ld.so` programının, bir programı çalışmaya hazır hale getirip, çalıştırdığını söylemiştik.

Eğer `LD_TRACE_LOADED_OBJECTS` çevre değişkeni 1 olarak tanımlanırsa, bu program `ldd` gibi çalışır ve mevcut programın bağlı olduğu kütüphaneleri listeler ve hemen durur.

Örnek olarak, `busybox` programının bağlı olduğu kütüphaneler, aşağıdaki örnekte, `ld.so` programı ile tespit edilmiştir.

```
root@UCanLinux:/bin # LD_TRACE_LOADED_OBJECTS=1 /lib/ld-2.26.so /bin/busybox

        linux-vdso.so.1 (0x000000200001b000)
        libm.so.6 => /lib/libm.so.6 (0x000000200001f000)
        libresolv.so.2 => /lib/libresolv.so.2 (0x00000020000b5000)
        libc.so.6 => /lib/libc.so.6 (0x00000020000c5000)
        /lib/ld-linux-riscv64-lp64d.so.1 => /lib/ld-2.26.so (0x0000002000000000)
```

Kütüphaneler hakkında bir kaç kelime etmek gerekebilir.

`libc.so.6`[3] kütüphanesi, C programları tarafından kullanılan standard fonksiyonları barındırır. Statik olmayan bütün C programları, istisnasız bu kütüphaneye bağımlıdır.

`ld-2.26.so`[4] kütüphanesine youmlayıcı denir. Daha önce bolca bahsettiğimiz ve kısaca `ld.so` ile gösterilen, çalışabilir bir kütüphanedir. Ne tür olursa olsun, bütün ELF kodları bu kütüphaneye bağımlıdır. Bu kütüphane olmadan, çekirdek program çalıştıramaz.

Dinamik derlenmiş bütün programlar, istisnasız `libc.so.6` ve `ld-2.26.so` kütüphanelerine bağımlıdır. Bunun dışındaki kütüphaneler seçimliktir.

En kafa karıştırıcı kütüphane ise, `linux-vdso.so.1` adlı kütüphanedir. Yukarıdaki çıkışa dikkat edilirse, bu kütüphane isminin karışısında sadece adres vardır, bir dosya ismi yoktur. Çünkü bu kütüphane sanaldır. Fiziksel bir dosyaya sahip değildir.

Bu kütüphaneye "sanal, dinamik paylaşılan kütüphane"[5] denir. Kütüphane adında bulunan `vdso` kısaltması, `virtual dynamic shared object` kelimelerinin baş harflerinden gelir. Bu kütüphanenin çok basit bir varlık sebebi vardır.

Genelde sistem çağrıları[6], yani user space tarafından çağrılan çekirdek fonksiyonları maliyetlidir.

Bir sistem çağrısı yapılmadan evvel, çağrıyı yapan user space prosesinin durum bilgileri bir yerde kaydedilir, sonra interrupt üretilerek çekirdek tarafındaki adrese giriş yapılır ve istenen fonksiyon yürütülür. Çekirdek de

[3] $ man 7 libc

[4] $ man ld.so

[5] $ man vdso

[6] open(), close(), read(), socket(), pipe(), vb. Her zaman 2 numaralı man sayfasında bulunurlar, $ man 2 socket gibi.

işini bitirince tekrar bir interrupt üretir ve user space tarafındaki program, saklamış olduğu eski durum bilgilerini kullanarak kaldığı yerden çalışmasına devam eder.

Çok kaba bir biçimde tasvir etmeye çalıştığımız bu işleme, context switch[7] denir. Bu işlem son derece maliyetlidir. Sık kullanılan, özellikle zaman ile ilgili fonksiyonlar, vdso içinde bulunurlar. Böylece, örneğin time ile ilgili bir sistem çağrısı yapıldığında, çekirdeğe başvuru yapılmaz, context switch'e gerek kalmaz ve çok etkin biçimde sistem çağrısı gerçeklenir.

Tekrar `ld.so` programına dönecek olursak, bu programın kullanım bilgisini elde etmek için aşağıdaki gibi, `ld.so` programı argümansız girilir.

```
root@UCanLinux:~ # /lib/ld-2.26.so

Usage: ld.so [OPTION]... EXECUTABLE-FILE [ARGS-FOR-PROGRAM...]
You have invoked ‘ld.so’, the helper program for shared library executables.
This program usually lives in the file ‘/lib/ld.so’, and special directives
in executable files using ELF shared libraries tell the system’s program
loader to load the helper program from this file.  This helper program loads
the shared libraries needed by the program executable, prepares the program
to run, and runs it.  You may invoke this helper program directly from the
command line to load and run an ELF executable file; this is like executing
that file itself, but always uses this helper program from the file you
specified, instead of the helper program file specified in the executable
file you run.  This is mostly of use for maintainers to test new versions
of this helper program; chances are you did not intend to run this program.

  --list                list all dependencies and how they are resolved
  --verify              verify that given object really is a dynamically linked
                        object we can handle
  --inhibit-cache       Do not use /etc/ld.so.cache
  --library-path PATH   use given PATH instead of content of the environment
                        variable LD_LIBRARY_PATH
  --inhibit-rpath LIST  ignore RUNPATH and RPATH information in object names
                        in LIST
  --audit LIST          use objects named in LIST as auditors
```

Çekirdeğin, doğrudan kendisi tarafından, user space tarafında çalıştırdığı programlara, "helper program" denir. `ld.so` programı bir helper programdır. Bu tür programlar çok nadirdir.

[7] https://www.sifive.com/blog/all-aboard-part-7-entering-and-exiting-the-linux-kernel-on-risc-v

Helper program'a bir başka örnek de busybox içindeki `mdev` adlı hotplug programıdır. Gömülü sisteme bir cihaz takıldığı zaman çekirdek, user space tarafındaki hotplug programını çalıştırır. Hotplug programı da içindeki config bilgilerine göre gerekli modülleri yükler veya diğer işlemleri yapar.

Çekirdek, hotplug programının user space tarafında, nerede oturduğunu ve adını bilmez. Genelde açılış sırasında, `/etc/rcS` gibi bir programın içinden, çekirdeğe programın yeri hakkında bilgi gönderilir. Bu bilgi çekirdeğe nasıl gönderilir?

`/proc` dosya sistemi içinde `/proc/sys/kernel/hotplug` isimli bir dosya vardır. Bu dosya içine, aşağıdaki gibi `echo` komutu ile hotplug programının tam adı yazılır.

```
# genelde /etc/rcS içine yazılır
echo /sbin/mdev > /proc/sys/kernel/hotplug
```

Busybox'daki hotplug programının adı `mdev`'dir ve bu program her zaman `/sbin` altında oturur.

Aslında `/proc/sys/kernel/hotplug` dosyası, sıradan bir dosya değildir. Kernel space ile user space arasında bir pipe'tır. Pipe'ın bir ucuna bir bilgi yazıldığında, diğer ucundan okunabilir. Pipe yapısının bir ucu user space, bir ucu kernel space tarafına açılır.

Bu dosya bir pipe yapısı olduğundan, `$ ls -l` ile bakılınca boylar hep 0 gözükür.

```
# host veya RiscV'de denenebilir
$ ls -l /proc/sys/kernel/hotplug
-rw-r--r-- 1 root root 0 Haz  8 20:24 /proc/sys/kernel/hotplug
$ #                    ^
$ #                    dosya boyu 0'dır.
```

Dikkat edilirse, mod bilgisi `rw-` ile başlar. Buradaki `r` karakteri bize, çekirdekten bu isim aracılığı ile bilgi alınabileceğini, okunabileceğini gösterir. Örneğin `$ cat /proc/sys/kernel/hotplug` gibi.

Benzer şekilde `w` karakteri bize, çekirdeğe bu isim aracılığı ile bilgi gönderilebileceğini, yazılabileceğini gösterir. Örneğin `$ echo foo /proc/sys/kernel/hotplug` gibi.

`/proc/sys/kernel/hotplug` dosyasına yani pipe yapısına echo ile `/sbin/mdev` string'i gönderilirse, çekirdek tarafındaki uçta bu bilgi hemen okunur ve gerekli çekirdek yapılarına atama yapılır. `/proc` dosya sisteminin amacı da zaten "user space" ile "kernel space" arasındaki haberleşmeyi sağlamaktır.

`/sbin/mdev`[8] programının, `busybox` projesinin hotplug programı olduğunu söylemiştik. Bu bir helper programdır. Yani çekirdek tarafından çalıştırılacaktır. `echo /sbin/mdev > /proc/sys/kernel/hotplug` ifadesi ile, çekirdeğe çalıştırılacak hotplug programının tam yeri ve adı verilmiştir.

Standard dağıtımlarda bu yöntem kullanılmaz. Bunun yerine net link soketlerini kullanan `udevd` programı kullanılır. Fakat gömülü sistemlerde hotplug yaygındır.

2.3.10 Açılış mesajı ve diğerleri

`login:` gelmeden önce üretilen mesaj `/etc/issue` içindedir.

Login olduktan sonra çalışan ilk betik `/etc/profile`'dır.

Qemu tarafından çekirdeğe `-append` ile aktarılan çekirdek parametreleri `$ cat /proc/cmdline` komutu ile incelenebilir.

CPU bilgisi `$ cat /proc/cpuinfo` ile incelenebilir.

Çekirdeğin destek verdiği dosya sistemlerinin listesi `$ cat /proc/filesystems` ile görülebilir. Bu listede `nodev` olarak gözüken dosya sistemleri sözde dosya sistemleridir, fiziksel bir cihaz değildir. Güç kesilince içinde bilgiler yok olur.

[8] https://git.busybox.net/busybox/plain/docs/mdev.txt

Bölüm 3

Qemu Parametreleri

Qemu[1] [2] [3] sistemi derya deniz bir sistemdir. Bunu görmek için aşağıdaki gibi help komutunu girmek yeterlidir.

```
$ qemu-system-riscv64 --help | more
```

Bir önceki bölümde kurulan örnek sistemi çalıştırmak için aşağıdaki qemu komutu kullanılmıştı.

```
$ sudo qemu-system-riscv64
        -nographic
        -m 128
        -machine virt
        -kernel bbl
        -append "root=/dev/vda ro"
        -drive file=riscv_disk,format=raw,id=hd0
        -device virtio-blk-device,drive=hd0
        -netdev user,id=net0,hostfwd=tcp::2323-10.0.2.20:23,hostfwd=tcp::8080-10.0.2.20:80
        -device virtio-net-device,netdev=net0
```

Bu bölümde bu seçenekler tek tek incelenecek ve bazı önemli kavramlardan bahsedilecektir.

[1] https://github.com/riscv/riscv-qemu/wiki
[2] https://qemu.weilnetz.de/doc/qemu-doc.html
[3] https://www.qemu.org

3.1 qemu-system-

`qemu-system-` ile başlayan programlar, işletim sistemi yüklü bordları emüle edebilirler. Arm işlemci desteği için `qemu-system-arm`, 32 bit x86 desteği için `qemu-system-i386` veya ppc işlemcileri için `qemu-system-ppc`, 64 bir RiscV işlemcileri için `qemu-system-riscv64` programları kullanılabilir. İşlemci desteği yelpazesi çok geniştir.

İşlemciler çeşitli bordlarda çalışırlar. Qemu, özel olarak bazı bordlara destek vermektedir. Desteklenen bordların listesi, aşağıdaki gibi `machine` seçeneği ile listelenebilir. Örnekte arm işlemcili bordların listesi görülmektedir.

```
$ qemu-system-arm -version
QEMU emulator version 2.5.0 ...

$ qemu-system-arm -machine help

Supported machines are:
akita                Sharp SL-C1000 (Akita) PDA (PXA270)
borzoi               Sharp SL-C3100 (Borzoi) PDA (PXA270)
canon-a1100          Canon PowerShot A1100 IS
cheetah              Palm Tungsten|E aka. Cheetah PDA (OMAP310)
collie               Sharp SL-5500 (Collie) PDA (SA-1110)
connex               Gumstix Connex (PXA255)
cubieboard           cubietech cubieboard
highbank             Calxeda Highbank (ECX-1000)
imx25-pdk            ARM i.MX25 PDK board (ARM926)
integratorcp         ARM Integrator/CP (ARM926EJ-S)
kzm                  ARM KZM Emulation Baseboard (ARM1136)
lm3s6965evb          Stellaris LM3S6965EVB
lm3s811evb           Stellaris LM3S811EVB
mainstone            Mainstone II (PXA27x)
midway               Calxeda Midway (ECX-2000)
musicpal             Marvell 88w8618 / MusicPal (ARM926EJ-S)
n800                 Nokia N800 tablet aka. RX-34 (OMAP2420)
n810                 Nokia N810 tablet aka. RX-44 (OMAP2420)
netduino2            Netduino 2 Machine
none                 empty machine
nuri                 Samsung NURI board (Exynos4210)
realview-eb          ARM RealView Emulation Baseboard (ARM926EJ-S)
realview-eb-mpcore   ARM RealView Emulation Baseboard (ARM11MPCore)
realview-pb-a8       ARM RealView Platform Baseboard for Cortex-A8
realview-pbx-a9      ARM RealView Platform Baseboard Explore for Cortex-A9
smdkc210             Samsung SMDKC210 board (Exynos4210)
spitz                Sharp SL-C3000 (Spitz) PDA (PXA270)
sx1                  Siemens SX1 (OMAP310) V2
```

```
sx1-v1               Siemens SX1 (OMAP310) V1
terrier              Sharp SL-C3200 (Terrier) PDA (PXA270)
tosa                 Sharp SL-6000 (Tosa) PDA (PXA255)
verdex               Gumstix Verdex (PXA270)
versatileab          ARM Versatile/AB (ARM926EJ-S)
versatilepb          ARM Versatile/PB (ARM926EJ-S)
vexpress-a15         ARM Versatile Express for Cortex-A15
vexpress-a9          ARM Versatile Express for Cortex-A9
virt                 ARM Virtual Machine
xilinx-zynq-a9       Xilinx Zynq Platform Baseboard for Cortex-A9
z2                   Zipit Z2 (PXA27x)
```

3.2 nographic

`-nographic`

Normalde qemu sisteminin kendi GUI sistemi vardır. Bu seçenek GUI sisteminin başlamasını engeller ve seri kanal çıkışlarını doğrudan terminale yönlendirir.

3.3 m

`-m 128`

Bu seçenek ile sanal Linux sistemindeki RAM miktarı, MB cinsinden verilir. Kullanılmazsa 128MB kabul edilir.

3.4 machine

`-machine virt`

RiscV için yazılan qemu sistemi aşağıdaki bordları destekler.

```
$ qemu-system-riscv64 -machine help

Supported machines are:
```

```
none                 empty machine
sifive_e             RISC-V Board compatible with SiFive E SDK
sifive_u             RISC-V Board compatible with SiFive U SDK
spike_v1.10          RISC-V Spike Board (Privileged ISA v1.10) (default)
spike_v1.9.1         RISC-V Spike Board (Privileged ISA v1.9.1)
virt                 RISC-V VirtIO Board (Privileged ISA v1.10)
```

Üzerinde çalışılmak istenen bord `-machine` seçeneği ile verilir. Tamamen keyfi olarak `virt` bordu seçilmiştir. Bu bord, RiscV'in 1.10 ISA şartnamesini destekler. Virt bordu aslında sanal bir borddur. Gerçek bir bordun karşılığı değildir. Bu bord, network, blok cihaz ve UART gibi cihazları destekler.

3.5 kernel

```
-kernel bbl
```

Linux çekirdeği `-kernel` seçeneği ile verilir. `bbl` bir ELF programdır, önyükleyici ve Linux çekirdeğinden oluşur.

3.6 append

```
-append "root=/dev/vda ro"
```

Çekirdeğe geçirilecek parametreler `-append` ile verilir. Qemu sistemi `-append`'in karşısındaki bilgilerle ilgilenmez. Bu bilgiler, gözü kapalı bir biçimde çekirdeğe aktarılır.

Çekirdek açıldıktan sonra bir kök dosya sistemi bağlar. Ama bu kök dosya sisteminin adını bilemez. Bundan dolayı `root=...` parametresi ile kök dosya sisteminin oturduğu cihaz adı açıkça verilir.

Cihaz adı, diğer bir deyişle disk ismi olarak `/dev/sda` veya `/dev/hda` yerine `/dev/vda` diye alışılmadık bir isim kullanılmıştır.

Diskimiz SCSI olsaydı `/dev/sda`, ide olsaydı `/dev/hda` ismi kullanılacaktı.

Sanal makine yaratan ve çalıştıran programlara hypervisor denir. Diğer bir deyişle işletim sistemi altında işletim sistemi çalıştıran programlardır. Qemu,

VMWare, VirtualBox programları hypervisor sistemlerine örneklerdir.

Qemu sistemini çalıştırırken `root=/dev/sda` demiş olsaydık, qemu sistemi yani hypervisor programı SCSI arabirimini emüle etmek zorunda kalacaktı. Benzer şekilde `root=/dev/hda` parametresini girmiş olsaydık, hypervisor programı bir IDE arabirimini emüle edecekti.

Bunun yerine `root=/dev/vda` girerek, hypervisor programına, bir blok cihaz tanımladığımızı, bu cihazın IDE veya SCSI gibi bir donanım arabirimine sahip olmadığını, arabirim emülasyonuna gerek kalmadan diski kullanabileceğini söylemekteyiz. Böylece qemu'nun hızında çok büyük bir artış meydana gelecektir. Çünkü cihaz bilgileri, donanım emülasyonu olmadan, doğrudan okunmaktadır.

İşte bu tür donanım arabirimine gerek olmadan kullanılan cihazlara virtio cihazları denir. Virtio, network ve disk cihaz sürücüleri için geliştirilmiş bir sanallaştırma standardıdır. Virtio sürücüleri, kendilerinin bir hypervisor altında, sanal bir ortamda çalışacaklarını bilirler ve buna göre davranırlar. Bu da hypervisor programlarının çok verimli ve hızlı çalışmasını sağlar.

Qemu sisteminin desteklediği virtio cihazların listesi aşağıdaki gibi elde edilir.

```
\$ qemu-system-riscv64 -device help | grep virtio

"vhost-scsi", bus virtio-bus
"vhost-user-blk", bus virtio-bus
"vhost-user-scsi", bus virtio-bus
"virtio-blk-device", bus virtio-bus
"virtio-blk-pci", bus PCI, alias "virtio-blk"
"virtio-blk-pci-non-transitional", bus PCI
"virtio-blk-pci-transitional", bus PCI
"virtio-scsi-device", bus virtio-bus
"virtio-scsi-pci", bus PCI, alias "virtio-scsi"
"virtio-scsi-pci-non-transitional", bus PCI
"virtio-scsi-pci-transitional", bus PCI
"virtio-net-device", bus virtio-bus
"virtio-net-pci", bus PCI, alias "virtio-net"
"virtio-net-pci-non-transitional", bus PCI
"virtio-net-pci-transitional", bus PCI
"vhost-user-input", bus virtio-bus
"virtconsole", bus virtio-serial-bus
"virtio-input-host-device", bus virtio-bus
"virtio-input-host-pci", bus PCI, alias "virtio-input-host"
"virtio-keyboard-device", bus virtio-bus
"virtio-keyboard-pci", bus PCI, alias "virtio-keyboard"
```

```
"virtio-mouse-device", bus virtio-bus
"virtio-mouse-pci", bus PCI, alias "virtio-mouse"
"virtio-serial-device", bus virtio-bus
"virtio-serial-pci", bus PCI, alias "virtio-serial"
"virtio-serial-pci-non-transitional", bus PCI
"virtio-serial-pci-transitional", bus PCI
"virtio-tablet-device", bus virtio-bus
"virtio-tablet-pci", bus PCI, alias "virtio-tablet"
"virtserialport", bus virtio-serial-bus
"virtio-gpu-device", bus virtio-bus
"virtio-gpu-pci", bus PCI, alias "virtio-gpu"
"vhost-vsock-device", bus virtio-bus
"virtio-balloon-device", bus virtio-bus
"virtio-balloon-pci", bus PCI, alias "virtio-balloon"
"virtio-balloon-pci-non-transitional", bus PCI
"virtio-balloon-pci-transitional", bus PCI
"virtio-crypto-device", bus virtio-bus
"virtio-crypto-pci", bus PCI
"virtio-rng-device", bus virtio-bus
"virtio-rng-pci", bus PCI, alias "virtio-rng"
"virtio-rng-pci-non-transitional", bus PCI
"virtio-rng-pci-transitional", bus PCI
```

Örneğin PCI mouse için virtio-mouse-device cihazı kullanılabilir. Bu arada standard cihazlar da kullanılabilir. Fakat daha yavaş olacağı aşikardır.

`/dev/vda` cihazı, virtio-blk-device cihazına karşılık gelir. Bu isim daha sonra `-device` seçeneği ile ayrıca verilecektir. Bu cihazın Linux tarafındaki ismi `/dev/vda` olduğu için `root=/dev/vda` girilmiştir.

Linux tarafında, `/dev/vda` cihaz sürücüleri, bir hypervisor altında çalıştığını anlayacak ve bir SCSI veya IDE fiziksel cihazına veri göndermek yerine, virtio standardına göre veri alış/verişi yapacak ve doğrudan süpervisor ile uyumlu çalışacaktır. Bu da hızı çok artırır Çünkü hypervisor programı IDE/SCSI cihazları için emülasyon yapmaya gerek duymayacaktır.

`-device virtio-blk-device ...` seçeneği ile de diskin qemu tarafında virtio standardına göre yönetileceğini söylenir.

`-append "root=/dev/vda ro"` ifadesinde bir de "ro" parametresi[4] mevcuttur. Bu parametre sayesinde `/dev/vda`'da oturan kök dosya sistemi salt-okunur bağlanacaktır. Böylece sistem ani kapanmalara dayanıklı olacaktır. Bu parametre yazılmazsa, varsayılan değer `rw` olacaktır. Kök dosya sistemi

[4]https://www.kernel.org/doc/Documentation/admin-guide/kernel-parameters.txt

oku/yaz modunda bağlanacaktır.

Oku/yaz modunda iken oluşacak ani kapanmalardan sonraki açılışlarda, kök dosya sistemi mutlaka fsck[5] komutları ile tamir edilmelidir. fsck ve tamir komutları genelde açılış betiklerine konur. Kök dosya sistemi `rw` olan sistemlerde, tamir işlemlerinin yapılmaması Rus ruleti oynamak gibidir. Sistemin ne zaman elde kalacağı belli olmaz.

Bütün standard Linux dağıtımlarında bu tamir sistemleri [6] istisnasız mevcuttur. İç yapıları da çok karışıktır. Her ne kadar log tabanlı `ext3, ext4` gibi sistemler dosya sistemi bütünlüğünü sağlasalar da bütün bu sıkıntıları berteraf etmenin en basit yolu, kök dosya sistemini `ro` bağlamaktır.

3.7 drive

```
-drive file=riscv_disk,format=raw,id=hd0
```

Bu seçenek ile sanal diskin ismi ve formatı verilir. Sanal diskimizin adı `riscv_disk`'tir ve tamamen keyfidir. format olarak da genelde `raw` verilir. Eğer `raw` yazılmazsa, qemu sistemi diskin formatını tespit etmeye çalışır. Hatalı bir tespitin önüne geçmek için açıkça `raw` yazılır.

`id=hd0` ile bu diske keyfi bir isim verilir. `hd0` yerine `benim_guzel_diskim` de yazılabilir. İsim tamamen keyfidir, daha sonraki `-device` parametresinde bu isim kullanılacaktır.

3.8 device

```
-device virtio-blk-device,drive=hd0
```

Qemu sistemine bir cihaz sürücüsü eklememizi sağlar. Diğer bir deyişle, `virtio-blk-device` tanımı ile qemu sistemine, virtio standardına uyan bir blok cihaz bağlanacağı haber verilmektedir. `drive=hd0` ile de blok cihazın, yani diskin ismi verilmektedir. Bir önceki `-drive` seçeneğinde ilgili disk,

[5] file system check

[6] Ubuntu 16.04 için bakınız, /etc/init.d/checkroot.sh, /etc/init.d/checkfs.sh

`-drive ...,id=hd0` şeklinde tanımlanmıştı. İşte bu `id` tanımı burada kullanılmakta, `drive` ile `device` eşleştirilmektedir.

Drive kavramı için fiziksel disk, device kavramı için de `open(), read(), write(), ioctl(), fsync(), ...` gibi fonksiyonların topluluğu düşünülebilir.

3.9 netdev

```
-netdev user,id=net0,hostfwd=tcp::2323-10.0.2.20:23, \
                     hostfwd=tcp::8080-10.0.2.20:80
```

Qemu sistemi host ile iki türlü ağ iletişimi kurar. İlk iletişim şekline "user mode network stack" denir. Bu iletişim şekli verimsizdir ama root şifresi ve host tarafında ağ yapılandırması gerektirmez. Basit olduğu için örneğimizde bu yöntem kullanılmıştır.

Diğeri ise bridge yöntemidir. Sanal makine bu yöntemde, sanal bir switch veya sanal bir router ile, gerçek bir makineymiş gibi ağa bağlanır. Bu durumda sanal makinenin fiziksel bir makinden hiç bir farkı yoktur.

Bu işlem, Linux altında bulunan sanal bridge yöntemi ile yapılır. Önce host tarafında sanal bridge kurulur sonra sanal makinde bir tap veya tun cihazı yaratılır. tap cihazları, bildiğimiz "network switch" örneğinde olduğu gibi, layer 2 cihazlarıdır. Tun cihazları da bir layer 3 cihazıdır. Router'lar bu cihaza bir örnektir.

Tun veya tap kurulmuş sanal makine, host'daki bridge'e bağlanır. Bridge'nin bir ucunda tun/tap bağlı sanal makine veya makineler diğer ucunda ise host mevcuttur. Böylece sanal makineler ve host, bridge ile, sanki hepsi bir switch veya router ile birbirlerine bağlıymış gibi çalışırlar.

Bridge, yapı itibari ile birbirlerinden farklı ağları homojen bir ağmış gibi göstermeye yarar. Öyle ki bir ucunda host'daki eth0 diğer ucunda ise tun/tap cihazları vardır. Dışarıdan bakıldığında, brigde bunların hepsini tek bir ağdaymış gibi gösterir.

Aslında "user mode network" yerine bridge yöntemi yazılacaktı ama yazı bitmeyecek, vazgeçtim.

Sonraki parametre "id=net0" ile verilir. Bütün tanımı belirten keyfi bir isimdir. Sonra `-device` seçeneğinde kullanılacaktır.

`hostfwd=tcp::2323-10.0.2.20:23` kuralı, host tarafından gelen ağ paketlerinin sanal makine içinde yönlendirilmesini sağlar. `hostfwd` kuralının genel yazılışı aşağıda verilmiştir.

```
hostfwd=[tcp|udp]:[hostaddr]:hostport-[guestaddr]:guestport
```

Örneğimizde, `tcp::2323` tanımında ortadaki kısımda hostaddr bilgisi olmalıdır. Ama boştur. Bu örnekte, herhangi IP'li makineden, qemu'nun 2323 portuna gelen tcp paketleri yakalacaktır.

Tanımın devamında `-10.0.2.20:23` mevcuttur. Yakalanan paketler qemu altında çalışan 10.0.2.20 IP'li makinenin 23 numaraları portuna yönlendirilecektir.

Özetle, host tarafından, qemu'nun 2323 portuna gönderilen paketler, içerdeki 10.0.2.20 IP'li sanal makinenin 23 numaralı protuna aktarılacaktır.

Sanal makineye 10.0.2.20 adresi tamamen keyfi olarak verilmiştir. Bu sabit IP bilgisi `/etc/network/interfaces` dosyasında bulunabilir. Açılışta ise, `/etc/rcS` içinden `ifup -a` komutu ile ayağa kaldırılmıştır.

Host makinenin bağlı olduğu network 192'li ise sanal makinelere 10'lu IP verilmesi tavsiye edilir.

Sanal makinedeki telnet sunucusu 23 numaraları standard porttan dinleme yapar. Host makineden, yani dışardan `$ telnet localhost 2323` komutu ile sanal makineye giriş yapılabilir.

Genelde içerde çalışan sunucunun port numarası çift yazılarak bağlantı kurulur. Tabii ki bir zorunluluk yoktur ama bu şekilde port numaralarını hatırlamak daha kolaydır.

Peki niçin farklı bir port numarası kullandık? Sebeb çok basit. Öncelikle `$ telnet localhost` yazmış olsaydık, qemu içindeki sanal makineye hiç bir paket gitmeyecekti. Çünkü host'daki telnet sunucusu bağlantıyı kabul edecekti.

`hostfwd=tcp::2323-...` tanımı sayesinde qemu sistemi 2323 numaralı porttan dinleme yapacaktır. `$ telnet localhost 2323` yazınca da telnet bağlantısı

için standard 23 numararlı port yerine 2323 numaralı porta başvuru yapılacaktır. Bu portu da qemu sistemi yukarıdaki tanım sayesinde dinlemektedir. Böylece paketleri, localhost'daki telnet sunucusu değil, qemu sistemi yakalayacaktır.

Sonra da bu paketler `hostfwd=...-10.0.2.20:23` tanımı sayesinde 10.0.2.20 IP'li sanal makinenin 23 numaralı portuna gönderilecektir. Sanal makinenin 23 numaralı portunda da telnet sunucusu dinleme yapmaktadır.

Aşağıdaki gibi sanal makine içinde dinleme yapılan portlar incelenebilir.

```
root@UCanLinux:~ # netstat -l
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 :::http                 :::*                    LISTEN
tcp        0      0 :::telnet               :::*                    LISTEN
...
```

Üçüncü kolonda http ve telnet sunucularının port isimleri geçmektedir. Bu isimlerin sayısal karşılığı `/etc/services` dosyasından `more` komutu ile incelenebilir. Bu isimlere karşı gelen port numaralarının 23 ve 80 olduğu görülebilir.

Sanal makinenin kendi IP değeri olan 10.0.2.20'yi kullanarak telnet bağlantısı yapamayız. Yani `$ telnet 10.0.2.20` veya `$ telnet 10.0.2.20 2323` ile bağlantı kurulamaz. Çünkü bu IP bilgisinin host tarafında bir anlamı yoktur.

Fakat bridge tekniğini kullanmış olsaydık doğrudan 10.0.2.20 ile sanal makineye erişim olacaktı. Bu durumda `hostfwd` tanımlarının hiç birine gerek de olmayacaktı.

Genelde sanal makinelere bağlantı için telnet değil de ssh kullanılır. Çünkü ssh güvenlidir. Daha basit olduğu için telnet kullanılmıştır. Okuyucu çok basit bir kaç değişiklikle ssh kullanabilir. Busybox'ın ssh suncusunun adı dropbear'dir.

Aynen telnet gibi, http için de `hostfwd=tcp::8080-10.0.2.20:80` ifadesi eklenmiştir. Telnet ile aynı mantığa sahip olduğundan ayrıca üzerinde durulmayacaktır.

Dışardan telnet, ssh veya http gibi bağlantılar yapılamayacaksa, hostfwd ve ilgili tanımlara da gerek olmayacaktır.

3.10 device

```
-device virtio-net-device,netdev=net0
```

Qemu tarafında network cihazı olarak `virtio-net-device` kullanılacaktır. Bu cihaz doğrudan hypervisor için yazılmıştır ve çok hızlıdır.

`netdev=net0` tanımı ile de, bir adım evvel `-netdev ...,id=net0...` şeklinde tanıtılan network cihazı `virtio-net-device` sürücüsüne bağlanmıştır. Aradaki bağlantıyı keyif şekilde verilen `net0` ismi sağlamıştır.

Bitti.

Dizin