Laboratorul 01. Intro & QEMU

Atentie! Pentru rezolvarea laboratorului recomandam folosirea sistemului de operare Ubuntu 18.04. Puteti folosi o masina virtuala Ubuntu 18.04 Bionic Beaver download-ata de pe osboxes [https://www.osboxes.org/ubuntu/] si rulata in VirtualBox [https://www.virtualbox.org/] sau VmWare.

Introducere

Bine ați venit în laboratorul de Sisteme Embedded!

Laboratorul își propune să vă familiarizeze cu sisteme embedded care rulează Linux, de la dezvoltare și configurare, până la mentenanță. Vom trata subiecte precum:

- Rularea/Compilarea de aplicații pe un sistem embedded
- Bootloadere, Kernel, Root FS, Toolchain
- Instalarea și configurarea de servicii
- Construirea unei distribuții Linux pentru sisteme embedded
- Emularea sistemelor

De ce Linux?

Sistemele Linux oferă o mulțime de avantaje dezvoltatorilor de produse, care micșorează timpul de dezvoltare, lucru care este din ce în ce mai important în zilele noastre:

- **versatilitate**: Sistemele Linux nu trebuie să fie single-purpose, se pot adăuga multiple funcționalități cu ușurință (chiar și în etapa de post-producție)
- **codebase mare**: Sistemele Linux abundă de aplicații user-space, drivere pentru o mulțime de dispozitive, suport pentru multe protocoale/sisteme de fișiere/etc.
- **securitate**: Sistemele care folosesc servicii comune în Linux beneficiază de același nivel de securitate ca pe un sistem desktop sau server

De-a lungul anilor Linux a devenit cel mai folosit sistem de operare pentru aplicațiile embedded. Îl puteți găsi folosit în orice:

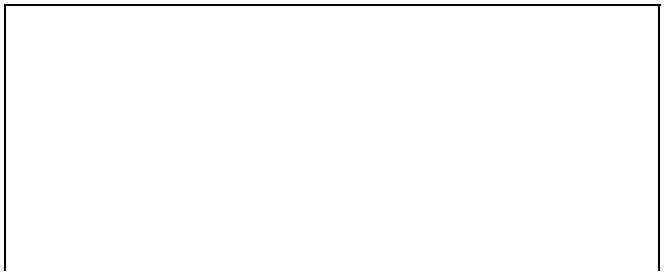
- telefoane mobile (Android)
- router-e
- DVR, NAS
- quadcoptere
- console de jocuri [http://store.steampowered.com/livingroom/SteamMachines/]
- frigidere [http://www.geek.com/chips/this-intelligent-fridge-runs-linux-on-an-arm-chip-1297126/]

După cum se poate vedea, sistemele embedded diferă foarte mult în dimensuni și putere de procesare, unele dintre ele apropiindu-se chiar de puterea de procesare a unui calculator obișnuit. De asemenea, aplicațiile pe care acestea le rulează pot fi foarte variate (ex: smartphone), amestecând diferențele dintre un calculator obișnuit și un sistem embedded. Un lucru care deosebește însă sistemele embedded este modul de interacțiune cu utilizatorii, care foarte rar se face printr-un ecran și o tastatură. Lipsa unui mod tradițional de interacțiune cu utilizatorul este și ceea ce face dezvoltarea unui sistem embedded mai grea, dar și mai interesantă.

Cele mai întâlnite două metode de interacțiune cu un sistem embedded în timpul dezvoltării sunt: consola serială și conexiunea SSH. Dintre acestea, conexiunea SSH este metoda mai robustă și mai simplu de utilizat, însă ea e disponibilă doar pe sistemele care dispun de o interfață de rețea. Consola serială, însă este de obicei prezentă pe orice sistem și permite interacțiunea cu sistemul chiar și înainte ca interfața de rețea să fie disponibilă (ex: în bootloader sau înainte de inițializarea driver-ului de rețea).

Există două concepte importante folosite în dezvoltarea unui sistem embedded: **target** și **host**. *Target*-ul este reprezentat de sistemul embedded pe care îl dezvoltăm și la care ne conectăm (ex: RaspberryPi, Intel Galileo etc.). *Host*-ul este reprezentat de calculatorul pe care îl folosim pentru dezvoltare și prin care ne conectăm cu sistemul embedded. *Host*-ul beneficiază de obicei de o putere de procesare mult mai mare, care micșorează de exemplu timpul de compilare în comparație cu compilarea pe *target*.

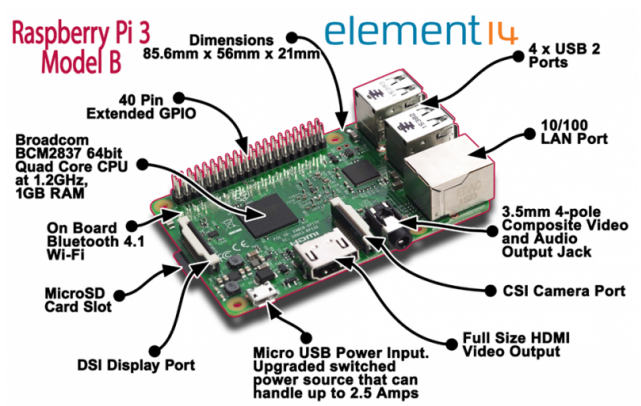
RaspberryPi



RaspberryPi Model B

Vom lucra în principal cu RaspberryPi 3, un sistem de calcul bazat pe un procesor "System on Chip" ARM de la Broadcom. Specificațiile complete sunt:

- procesor: 64-bit quad-core ARM Cortex-A53, 1.2GHz
- 1GB RAM
- 4 porturi USB 2.0
- 1 conector Ethernet
- card microSD
- HDMI, jack audio, RCA
- Diverse alte periferice: GPIO, UART-uri, I²C, SPI, I²S



Schema perifericelor RaspberryPi

Schema bloc

Din punct de vedere hardware, RaspberryPi este un dispozitiv simplu, care expune diferitele periferice pe care le oferă SoC-ul Broadcom. Singura excepție o reprezintă Hub-ul USB, care dublează numărul de porturi USB disponibile și atașează și un dispozitiv Ethernet la SoC-ul Broadcom.

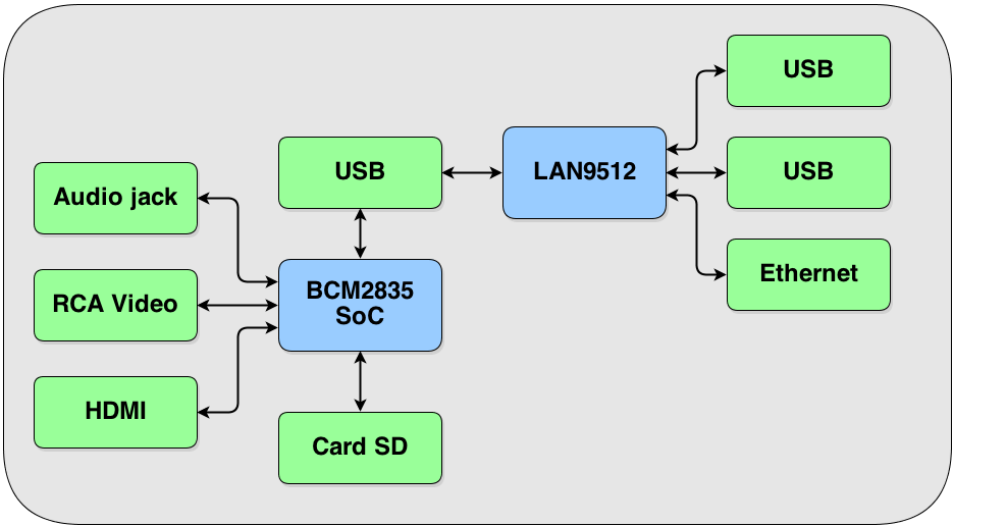


Diagrama bloc

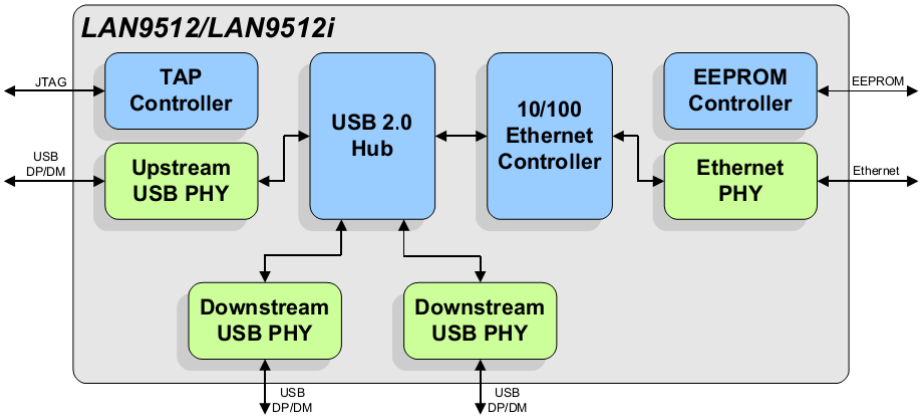


Diagrama block a chip-ului de USB și Ethernet

QEMU

QEMU [http://qemu.org] este un emulator/mașină virtuală care permite rularea unui sistem de operare complet ca un simplu program în cadrul unui alt sistem. A fost dezvoltat inițial de Fabrice Bellard [https://en.wikipedia.org/wiki/Fabrice_Bellard] și este disponibil gratuit, sub o licență open source. QEMU poate rula atât pe Linux, cât și pe Windows [1] [3] [4].

Este o mașină virtuală deoarece poate virtualiza componentele fizice ale unui sistem de calcul, pentru a permite rularea unui sistem de operare, oaspete (*guest*), în cadrul altui sistem de operare, gazdă (*host*). În acest mod de funcționare, atât sistemul *guest*, cât și sistemul *host*, folosesc aceeași arhitectură (ex: x86). QEMU poate folosi un hypervisor (Xen sau KVM) pentru a accelera rularea *guest*-ului atunci când există suport pentru virtualizare în hardware. În acest caz QEMU poate atinge o performanță comparabilă cu sistemul nativ, deoarece lasă mare parte din cod să se execute direct pe procesorul *host*. Folosind KVM sunt suportate diferite arhitecturi, printre care x86, PowerPC și S390 [1].

Este un emulator deoarece poate rula sisteme de operare și programe compilate pentru o platformă (ex: o placă ARM) pe o altă platformă (ex: un PC x86). Acest lucru este făcut prin translatarea dinamică a intrucțiunilor architecturii *guest* în instrucțiuni pentru arhitectura *host*. Ca un emulator, QEMU poate rula în două moduri [2] [4]:

- *User-mode emulation [http://wiki.qemu.org/download/qemu-doc.html#QEMU-User-space-emulator]*, în care un executabil obișnuit (user-space), compilat pentru o arhitectură, este rulat pe o altă arhitectură. În acest mod de funcționare instrucțiunile din executabil sunt translatate în instrucțiuni ale arhitecturii *host*, iar argumentele apelurilor de sistem sunt convertite pentru a putea fi pasate sistemului de operare *host*. Sistemele de operare emulate sunt: Linux, Mac OS X și BSD. Principalele utilizări sunt cross-debugging-ul și cross-compilarea, unde rulăm un compilator nativ al arhitecturii *target*, pe arhitectura *host*.
- *System emulation [http://wiki.qemu.org/download/qemu-doc.html#QEMU-System-emulator-for-non-PC-targets]*, în care este emulat un sistem de calcul complet. QEMU permite emularea unui număr mare de platforme, bazate pe diferite arhitecturi (ex: x86, ARM, PowerPC, MIPS, SPARC, MicroBlaze etc.), împreună cu perifericele lor. În acest mod de funcționare pot fi rulate sisteme de operare întregi, printre care Windows, Linux, Solaris, BSD și DOS.

În dezvoltarea sistemelor embedded, QEMU este folosit deoarece poate emula un sistem de calcul complet, nefiind necesar ca sistemul țintă (*target*) pentru care se face dezvoltarea, și sistemul *host*, pe care se face dezvoltarea, să folosească aceeași arhitectură. Acest lucru permite ca dezvoltarea software-ului pentru un sistem embedded să poată fi făcută în paralel cu proiectarea hardware-ului, lucru crucial pentru obținerea unui timp de dezvoltare scurt. Un alt avantaj pe care il poate avea emularea, mai ales a sistemelor low-end, este o viteză superioară a emulării pe un sistem *host* performant, în comparație cu sistemul *target*.

Instalare

Cel mai simplu mod de instalare pe o distribuție Linux este de a folosi *package manager*-ul. În majoritatea distribuțiilor pachetul principal se numește `qemu` și cuprinde de obicei toate executabilele aferente diferitelor moduri de funcționare ale QEMU. Dacă se dorește doar modul de virtualizare cu KVM poate fi instalat pachetul `qemu-kvm`, iar dacă se dorește modul de emulare a unui sistem ARM poate fi instalat pachetul `qemu-system-arm`.

Ubuntu 18.04

```
sudo apt-get update
sudo apt-get install qemu
sudo apt-get install qemu-kvm
sudo apt-get install qemu-system-arm
```

Dacă distribuția folosită nu oferă un pachet pentru QEMU (ex: Windows) sau pachetul oferit conține o versiune prea veche (ex: Fedora 17), se poate alege instalarea din surse. În acest caz, sursele oficiale pot fi downloadate din repository [https://git.qemu.org/?p=qemu.git;a=summary]-ul de Git, iar pentru instrucțiuni de compilare se poate urmări pagina de manual [http://wiki.qemu.org/download/qemu-doc.html#compilation].

Rulare

Pentru rularea unei mașini virtuale cu KVM se folosește comanda `qemu-kvm` împreună cu imaginea pentru hard disk. În acest caz imaginea hard disk-ului trebuie să conțină un sistem compatibil cu arhitectura *host*, accelerarea oferită de KVM putând fi folosită doar dacă *guest*-ul și *host*-ul folosesc arhitecturi compatibile (ex: x86_64).

Pentru rularea în *user-mode emulation* poate fi folosit unul din executabilele de forma `qemu-<arch>` împreună cu executabilul pe care vrem să-l rulăm [5]. Bineînțeles, acest executabil trebuie să fie compatibil cu arhitectura aleasă, `<arch>`, iar momentan QEMU oferă suport pentru *user-mode emulation* doar pe Linux și BSD. Dintre cele două, suportul pentru BSD nu este însă la fel de complet ca cel pentru Linux [6]

Exemplu de rulare in **user-mode emulation**:

```
qemu-arm -cpu <procesor> <executabil>
```

Pentru rularea în modul *system emulation* se folosește unul din executabilele de forma `qemu-system-<arch>` împreună cu imaginea pentru hard disk [7].

Exemplu de rulare in modul **system emulation**:

```
qemu-system-arm -machine <arhitectura> -cpu <procesor> -kernel <kernel_file> -append "root=/dev/sda2" -drive file=<rootfs_file>,index=0,media=disk,format=raw
```

Configurare

În modul mașină virtuală sau *system emulation* QEMU simulează un întreg sistem de calcul. În lipsa unor alte argumente se folosește însă o configurație implicită de sistem, care este specifică fiecărei arhitecuri în parte. QEMU poate însă simula o gamă largă de configurații de sistem. În limbajul QEMU acestea se numesc *mașini* și pot fi selectate cu opțiunea `-machine`.

Nokia N800 tablet

```
qemu-system-arm -machine n800 <disk image>
```

QEMU oferă însă și un control mai fin asupra configurației sistemului simulat printr-o serie de alte opțiuni, precum [8]:

- `-cpu` - specifică tipul de procesor care va fi emulat
- `-m` - specifică dimensiunea memoriei RAM
- `-hda`, `-hdb` etc. - specifică imaginea pentru primul hard disk, respectiv al doilea hard disk, ș.a.m.d
- `-fda`, `-fdb` - specifică imaginea pentru primul floppy disk, respectiv al doilea floppy disk
- `-cdrom` - specifică imaginea folosită de cdrom
- `-serial`, `-parallel` - specifică porturile seriale, respectiv, paralele și modul de interacțiune a acestora cu *host*-ul

Configurații mai avansate pot fi obținute cu opțiunile `-device`, `-drive`, `-net`, `-soundhw`, `-bt` care adaugă dispozitive periferice, de stocare, plăci de rețea și de sunet și, respectiv, dispozitive bluetooth [8]. Documentația [https://qemu.weilnetz.de/doc/4.2/qemu-doc.html] oferă informații despre toate aceste opțiuni, precum și multe altele.

O altă opțiune utilă este `-kernel`. Aceasta permite specificarea imaginii de kernel folosite de sistemul *guest* direct în comanda QEMU. Astfel, QEMU va încărca kernelul dintr-un fișier aflat pe sistemul *host* în loc de a-l cauta în imaginea de hard disk. Acest lucru poate reduce semnificativ timpul de iterație în momentul dezvoltării unui sistem embedded, deoarece nu mai este necesară recrearea imaginii de hard disk pentru fiecare modificare a kernel-ului.

Pe unele sisteme emulate este chiar obligatoriu ca opțiunea `-kernel` să fie prezentă, deoarece emularea sistemului nu include și un bootloader. Fără un bootloader, sistemul nu știe altfel cum să găsească imaginea de kernel.

De obicei, împreună cu specificarea imaginii de kernel este nevoie să specificăm și linia de comandă a kernel-ului. Pentru aceasta se folosește opțiunea `-append` împreună cu string-ul care vrem să fie pasat kernel-ului la bootare.

O ultimă opțiune, folositoare mai ales pentru debugging, o reprezintă redirectarea monitorului către consolă. Acest lucru se face cu opțiunea `-monitor stdio`. Monitorul oferă o interfață în linie de comandă care permite un control interactiv al modului în care se face emularea.

Networking

Pentru a emula o interfață de rețea, QEMU se bazează pe două componente: *device*-ul prezentat *guest*-ului, configurat cu opțiunea `-device` sau `-net nic`, și *back-end*-ul care leagă acest device de *host*, configurat cu opțiunea `-netdev`. Opțiunea `-device` nu este limitată la a emula doar interfețe de rețea, ea putând configura orice dispozitiv suportat de către QEMU însă, unele plăci de rețea sunt suportate doar de opțiunea `-net nic`.

Pentru *back-end*, QEMU suporta mai multe moduri, printre care:

- `-netdev user` - *user-mode*, rulează în user-space și nu necesită privilegii, însă interacțiunea cu rețeaua *host*-ului este complicată
- `-netdev tap` - *tap*, conectează o interfață TAP a *host*-ului la un VLAN emulat, permițând o configurare detaliată a topologiei folosite de *guest*, însă configurarea este mai complicată
- `-netdev bridge` - *bridge*, conectează o interfață TAP a *host*-ului la un bridge, care permite interacțiunea cu rețeaua fizică a *host*-ului
- `-netdev socket` - *socket*, interconectează VLAN-urile a două sisteme emulate folosind TCP sau UDP.

În mod implicit QEMU emulează un sistem cu o interfață de rețea reprezentată de un *device* specificat de *mașina* selectată, în modul *user-mode*. Aceasta configurare implicită nu ne oferă însă toată flexibilitatea unui *target* real, conectat la o rețea fizică. Din acest motiv în cadrul laboratorului ne vom folosi de modul *bridge*.

Bridge-ul folosit de către *back-end* se configurează cu parametrul `br=<nume bridge>`, iar *device*-ul pentru opțiunea `-net nic` se specifică prin parametrul `model=<device>`. Legatura dintre cele două componente se face prin adăugarea parametrului `netdev=<id>` la *device* și a parametrului `id=<id>` la *back-end*. Valoarea `<id>` trebuie bineînteles să fie identică pentru ca cele două componente să fie legate. În final, cele două opțiuni arată astfel: `-net nic,model=<device>,netdev=<id> -netdev bridge,br=<nume bridge>,id=<id>`.

Înainte de realizarea configurațiilor de rețea, dezactivați conectarea automată din setările sistemului de operare (Settings → Network → Wired → Connect Automatically (off))

Pentru a crea și configura *bridge*-uri se folosește utilitarul **brctl** din pachetul **bridge-utils**. Crearea unui *bridge* care să ofere unui *guest* accesul la rețea fizică a *host*-ului se face astfel:

```
sudo brctl addbr virbr0                                     # creăm bridge-ul
sudo brctl addif virbr0 <interfata fizica>                  # adăugam interfața fizică a host-ului la bridge
sudo ip address flush dev <interfata fizica>                # ștergem adresa IP de pe interfața fizică, doar dacă avem o adresă
                                                            # IP pe interfață. Va șterge și ruta default automat
sudo dhclient virbr0                                        # obținem adresa IP pentru bridge și ruta default prin DHCP
```

Dacă nu merge obținerea adreselor prin DHCP, se poate configura manual adresa și ruta default:

```
ip address show                                             # notăm ip-ul și prefixul interfeței fizice
ip route show                                               # notăm ruta implicită
sudo brctl addbr virbr0                                     # creăm bridge-ul
sudo brctl addif virbr0 <interfata fizica>                  # adaugăm interfața fizică a host-ului la bridge
sudo ip address del <ip>/<prefix> dev <interfata fizica>    # mutăm adresa interfeței fizice
sudo ip address add <ip>/<prefix> dev virbr0                # pe bridge
sudo ip link set dev virbr0 up
sudo ip route add default via <gateway>                     # readăugam ruta implicită
```

Pentru ca *bridge*-ul să fie acceptat de QEMU el trebuie configurat și în fișierul `/etc/qemu/bridge.conf` sub forma:

bridge.conf

```
allow virbr0
```

Exerciții

Atenție! Pentru rezolvarea laboratorului recomandăm folosirea sistemului de operare Ubuntu 18.04. Puteți folosi o mașină virtuală Ubuntu 18.04 Bionic Beaver download-ată de pe osboxes [https://www.osboxes.org/ubuntu/] și rulată în VirtualBox [https://www.virtualbox.org/] sau VmWare.
Hint: ca să meargă copy-paste între host și guest pe VirtualBox, trebuie să activați Devices → Shared Clipboard → Bidirectional și să instalați Guest Additions (Devices → Insert Guest Additions CD Image).

În cadrul exercițiilor vom încerca să emulăm un sistem cât mai apropiat de RaspberryPi. Deoarece QEMU nu are suport pentru SoC-ul Broadcom BCM2835 folosit de RaspberryPi [9], vom folosi o placă Versatile Platform Baseboard [http://infocenter.arm.com/help/topic/com.arm.doc.dui0225d/DUI0225D_versatile_application_baseboard_arm926ej_s_ug.pdf] ca înlocuitor. Această placă conține un procesor ARM, bazat pe core-ul ARM926EJ-S [http://www.arm.com/products/processors/classic/arm9/arm926.php], asemănător cu cel folosit de SoC-ul BCM2835, bazat pe core-ul ARM1176JZ-F [http://www.arm.com/products/processors/classic/arm11/arm1176.php]. Acest ultim procesor implementează setul de instrucțiuni ARMv6 [9].

Ca sistem de operare vom folosi distribuția Raspbian Wheezy [https://www.raspberrypi.org/downloads/raspbian/]. Atât kernelul cât și fișierele distribuției au fost ușor modificate pentru a fi compatibile cu placa *Versatile PB*.

0. Instalați programele și utilitarele necesare:

- Instalați `git`, `vim` și `bridge-utils`.

```
sudo apt-get install git
sudo apt-get install vim
sudo apt-get install bridge-utils
```

- Instalați toolchain-ul necesar pentru a cross-compila programe pentru RaspberryPi: 1) Clonați/download-ați repo-ul [https://github.com/raspberrypi/tools]; 2) Adăugați in $PATH calea către directorul ce conține executabilele necesare.

```
git clone --depth=1 https://github.com/raspberrypi/tools.git
export PATH="/home/osboxes/tools/arm-bcm2708/arm-linux-gnueabihf/bin:$PATH"
```

- Instalați QEMU folosind instrucțiunile prezentate în acest laborator [https://ocw.cs.pub.ro/courses/si/laboratoare/01#instalare].

```
sudo apt-get update
sudo apt-get install qemu
sudo apt-get install qemu-system-arm
```

1. Aflați parametrii pentru opțiunile `-machine` și `-cpu` necesari pentru a emula sistemul nostru.

- Re-citiți despre modalitățile de rulare [https://ocw.cs.pub.ro/courses/si/laboratoare/01?&#rulare].
- Citiți pagina de manual sau documentația [https://qemu.weilnetz.de/doc/4.2/qemu-doc.html] online pentru cele două opțiuni.
- Hint:

```
qemu-system-arm -machine ?
qemu-system-arm -machine <platforma> -cpu ?
```

2. Creați un program **hello world** pentru RaspberryPi linkat *static*. Aflați setul de instrucțiuni folosit de executabilul generat și apoi rulați-l în QEMU folosind *user-mode emulation* și emulând procesorul ARM1176JZ-F. Salvați comanda folosită pentru emulare.

- Utilizaţi compilatorul **arm-linux-gnueabihf-gcc** din toolchain-ul download-at mai sus;
- Folosiți pentru compilator flag-ul `-static` pentru a obține un executabil linkat *static*.
- Utilitarul **file** oferă informații despre conținutul fișierelor primite ca argument.
- Bonus: Ce se întâmplă dacă rulați executabilul direct, fără QEMU? De ce?.

- Re-citiți despre modalitățile de rulare [https://ocw.cs.pub.ro/courses/si/laboratoare/01?&#rulare].
- Citiți introducerea acestui kernel feature [https://www.kernel.org/doc/Documentation/admin-guide/binfmt-misc.rst].
- Veti afla mai multe despre cross-compilare in laboratorul 2.

3. Rulați distribuția Raspbian folosind QEMU în modul *system emulation*. Salvați comanda folosită pentru emulare.

- Kernel-ul modificat pentru a rula pe platforma Versatile PB este disponibil aici [https://drive.google.com/open?id=0B0lgiPZNMMyvaEtfN3V4VVBxRjg].
- Imaginea hard disk-ului (*rootfs*-ul) modificată pentru a rula pe platforma Versatile PB este disponibilă aici [https://drive.google.com/open?id=0B0lgiPZNMMyvOTFMakFuY1N2Q1E].
- Folosiți string-ul `root=/dev/sda2` pentru linia de comandă a kernel-ului.
- La prima pornire distribuția Raspbian va rula un utilitar de configurare. Opțiunile implicite sunt suficiente pentru a porni sistemul.

- Re-citiți despre modalitățile de rulare [https://ocw.cs.pub.ro/courses/si/laboratoare/01?&#rulare].
- Nu uitați să specificați tipul procesorului și al mașinii emulate.
- Revedeți parametrii de configurare ai QEMU și citiți pagina de manual sau documentația [https://qemu.weilnetz.de/doc/4.2/qemu-doc.html] acestora.

4. Adăugați sistemului emulat o interfață serială redirectată către *stdio*. Ce diferență observați? Salvați comanda folosită pentru emulare.

- Rulați din nou distribuția și așteptați până se ajunge la ecranul de login.

- Revedeți parametrii de configurare ai QEMU și citiți pagina de manual sau documentația [https://qemu.weilnetz.de/doc/4.2/qemu-doc.html] acestora.

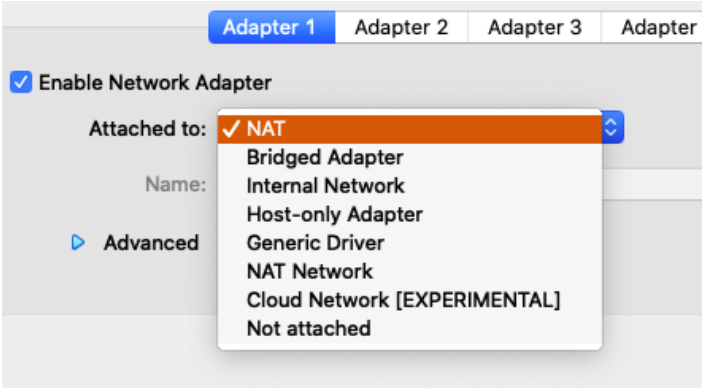
5. Adăugați string-ul `console=ttyAMA0` liniei de comandă a kernel-ului. Ce efect are parametrul adăugat? Adăgați și parametrul `console=tty1` liniei de comandă a kernelului. Ce efect au cei doi parametri combinați? Salvați comanda folosită pentru emulare.

- Parametrul `-append` nu poate fi invocat de mai multe ori, în ciuda numelui. Diferitele opțiuni trimise kernel-ului trebuie concatenate într-un singur șir. Folosiți "" pentru a escapa spațiile.

6. Configurați și testați accesul *guest*-ului la Internet. Salvați comanda folosită pentru emulare.

- Pe host, creați un bridge, care să ofere guest-ului acces la rețeaua host-ului.
- Emulați interfața de rețea pentru placa Versatile PB folosind device-ul *smc91c111*.

- Înainte de realizarea configurațiilor de rețea, dezactivați conectarea automată din setările sistemului de operare (Settings → Network → Wired → Connect Automatically (off)) și puneți placa de rețea a mașinii virtuale Ubuntu in modul de NAT (Devices → Network → Network Settings). Dacă folosiți o rețea wired și nu vă merge cu NAT atunci setați pe modul Bridged Adapter. Atenție, NU setați pe modul "NAT Network".



- Recitiți secțiunea de configurare a rețelei în QEMU.
- Folosiți `sudo` pentru rularea `qemu-system-arm`.

7. Rulați programul compilat la punctul 2 pe *guest*.

- Copiați executabilul folosind utilitarul `scp` de pe host pe guest.

Resurse

- Soluție laborator
- Kernel Raspbian compatibil cu QEMU [https://drive.google.com/open?id=0B0lgiPZNMMyvaEtfN3V4VVBxRjg]
- Rootfs Raspbian compatibil cu QEMU [https://drive.google.com/open?id=0B0lgiPZNMMyvOTFMakFuY1N2Q1E]
- Kernel patch pentru activarea arhitecturii ARMv6 pentru placa Versatile PB
- Kernel patch pentru a putea rula configurația implicită a Versatile PB în QEMU
- Raspbian Wheezy rootfs patch pentru QEMU

Referințe

1. QEMU website [http://wiki.qemu.org/Main_Page]
2. QEMU man page (Introduction) [https://qemu.weilnetz.de/doc/4.2/qemu-doc.html#Introduction]
3. QEMU Wikibooks page [https://en.wikibooks.org/wiki/QEMU]
4. QEMU Wikipedia page [https://en.wikipedia.org/wiki/QEMU]
5. QEMU man page (Linux User space emulator) [https://qemu.weilnetz.de/doc/4.2/qemu-doc.html#Linux-User-space-emulator]
6. QEMU BSD user-mode status page [https://wiki.freebsd.org/QemuUserModeToDo]
7. QEMU man page (ARM System emulator) [https://qemu.weilnetz.de/doc/4.2/qemu-doc.html#ARM-System-emulator]
8. QEMU man page (Invocation) [https://qemu.weilnetz.de/doc/4.2/qemu-doc.html#sec_005finvocation]
9. RaspberryPi hardware description [http://elinux.org/RPi_Hardware]
10. Copy-paste in VirtualBox [https://www.techrepublic.com/article/how-to-enable-copy-and-paste-in-virtualbox/]

si/laboratoare/01.txt · Last modified: 2020/11/14 13:54 by laura.ruse