

KIV/OS - dodatek A - qemu

Martin Úbl

12. října 2022

1 Emulace

Ne každý má k dispozici svoje Raspberry Pi Zero, a ne každý se chce omezovat jeho dostupností v rámci cvičení nebo ve vyhrazených časech mimo něj. Nabízí se proto možnost nějakým způsobem zajistit, že spustíme kód určený pro toto zařízení v rámci virtuálního stroje na běžném PC.

Na přednáškách si představíte různé volby pro emulaci a virtualizaci: emulaci, paravirtualizaci, plnou/přímou virtualizaci, a tak podobně. V tomto případě je nutné sahnout po emulaci, jelikož se architektura simulovaného PC a hostitelského výrazně liší.

Při emulaci je veškerý binární kód interpretován v rámci virtuálního stroje. Tento proces si lze představit tak, jako kdyby samotný binární kód byl vlastně kódem zdrojovým. Emulátor pak tyto instrukce dekoduje a koná akce, které mění virtuální stav stroje s ekvivalentní architekturou, jako je ten, co bychom měli fyzicky v ruce.

Je tedy vytvořena emulovaná paměťová mapa, registry, sběrnice, periférie a další. Vše je ale čistě v rovině softwarové, a tedy je běh potenciálně hodně pomalý. Daleko lepší výkon poskytuje přímá virtualizace, kdy je kód spuštěn přímo na hostitelském CPU. To by ale vyžadovalo podporu pro danou architekturu v rámci našeho CPU a to nelze zajistit.

Do emulace pak lze z vnějšku v omezené míře zasahovat, dle dispozic emulátoru. Můžeme například vypisovat paměťové bloky, registry, zapisovat uměle do vybraných sektorů paměti a periferních registrů, ale občas můžeme i připojit debugger a program ladit, jako kdyby byl fyzicky spuštěn na našem CPU.

2 Emulátor qemu

Pro emulaci Raspberry Pi Zero budeme používat emulátor `qemu`, který je zdarma k dispozici pod licencí GPLv2. Pro běh `qemu` v součinnosti s tím, co sestavujeme na cvičení je potřeba menší patch, a tedy není možné „tak jak je“ použít balík z distribucí.

Patchované `qemu` lze stáhnout z adresy: <https://github.com/MartinUbl/qemu>

Pokud i tak chcete použít qemu z distribucí např. v balíčkovacím systému apt v Debianu nebo jiném, je to možné pouze za předpokladu, že sestavovanou binární podobu vašeho jádra relokujete ne na adresu 0x8000, ale na 0x10000. To zajistíte v linker skriptu `link.ld`, který používáme.

Za předpokladu, že stáhnete (doporučenou) distribuci qemu z odkazu výše, je třeba ji nejprve sestavit.

Stáhněte tedy distribuci buď jako zip archiv, nebo použijte verzovací systém git:

```
git clone https://github.com/MartinUbl/qemu.git
cd qemu
```

Nyní vytvořte složku `build` a přesuňte se do ní:

```
mkdir build
cd build
```

V repozitáři je připraven `Makefile` a konfigurační skript. Za předpokladu, že máte nainstalovaný libovolný kompilátor jazyka C (`gcc`, `clang`) pro vaši architekturu, je možné qemu sestavit následujícími příkazy:

```
../configure --target-list="arm-softmmu"
make
```

U skriptu `configure` vidíte přepínač, kterým jej instruujeme, aby sestavil pouze emulaci pro `arm-softmmu` architekturu. Je to jediná, která podporuje emulaci Raspberry Pi Zero, a tedy jediná užitečná pro naše potřeby.

Volitelně pak lze přidat k příkazu `make` i přepínač `-j` s počtem sestavovacích paralelních procesů, např. `make -j4`.

Proces sestavení trvá dlouhou dobu, do hodiny by ale na průměrném čtyřjádrovém CPU mělo být vše sestaveno.

2.1 Jiné platformy

Emulátor qemu podporuje širokou škálu platform a architektur, od klasické x86, různé varianty ARM, až po PowerPC, MIPS a další.

Těmito platformami se zabývat nebudeme, je ale dobré vědět, že podpora v emulátoru je. Jde tedy o poměrně univerzální systém.

3 Emulace RPi0

Zkompilovaný kernel máme dostupný na nějaké naší známé cestě. V tomto případě nebudeme používat náš UART bootloader, jelikož není třeba měnit SD kartu. Můžeme rovnou nahrát jádro, které kompilujeme a balíme do souboru s názvem `kernel.img`.

V podsložce `build` je nově vytvořen spustitelný soubor `qemu-system-arm`, kterým spustíme emulaci. Je ale potřeba dodat určitou sadu parametrů:

```
./qemu-system-arm -machine raspi0 -serial null -serial mon:stdio \
                    -kernel /home/dev/kernel.img -nographic
```

Parametry:

- **-machine raspi0** – vybírá konkrétní kombinaci CPU, GPU, obrazu ROM, periférií a dalších
- **-serial null** – první sériový port (UART) nechceme
- **-serial mon:stdio** – druhý sériový port (miniUART) chceme multiplexovat se standardním vstupem a výstupem
- **-kernel /home/dev/kernel.img** – chceme použít obraz jádra uložený na dané adrese
- **-nographic** – qemu podporuje i grafický výstup, který ale nevyužijeme; tímto zamezíme inicializaci okna, a mj. dovolíme fungování např. i pod WSL (Windows Subsystem for Linux)

Zdvojením parametru **-serial** docílíme mapování více UART kanálů. Jelikož je miniUART mapován na UART1, musíme nejprve prvním **-serial null** říct qemu, že UART0 mapovat nechceme, a druhým **-serial mon:stdio**, že UART1 (miniUART) chceme multiplexovat se standardními proudy.

Multiplexing je nutný proto, že qemu má svou příkazovou řádku pro ovládání emulace. Tento parametr bude multiplexovat vstupy a výstupy s touto příkazovou řádkou. Implicitně se však ocitneme v režimu komunikace s hostovaným systémem. Pro přechod mezi tímto režimem a režimem konzole qemu stiskněte ctrl-A a C. Kdybychom použili pouze ctrl-C, byl by předán tento řídicí příkaz do hostovaného systému (který jej mj. momentálně ani neumí zpracovat).

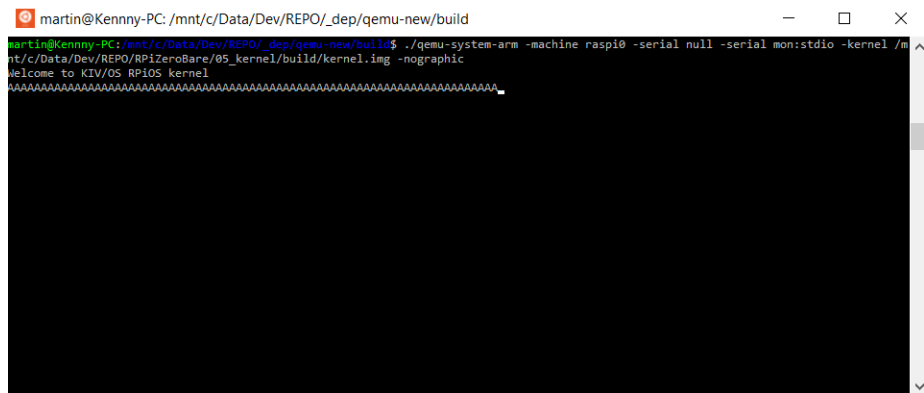
4 Testování

Pro testování použijme kódy ze 3. cvičení, ve kterém jsme implementovali UART driver. Zkompilujme zdrojové soubory a obdržme tedy soubor **kernel.img**. V tomto příkladu jsme nechali vypisovat znak **A** s určitou periodou. Očekáváme tedy, že po spuštění začne konzoli zaplavovat znak **A**, dokud emulátor neukončíme.

Po stisknutí ctrl-A a C se přepne ovládání do konzole qemu, výstupy však pokračují dál. Pro ukončení napíšeme příkaz **quit**, který se nejspíš bude prolínat se zapisovanými znaky **A**, a potvrdíme enterem.

5 Ladění

Pro ladění emulovaného kódu lze využít ladicí nástroj **gdb**. Pro debuggování jiné, než aktuální platformy je však nutné instalovat balíkovou verzi **gdb-multiarch**. Ta je dostupná ze standardních zdrojů vaší distribuce.



```
martin@Kennny-PC: /mnt/c/Data/Dev/REPO/_dep/qemu-new/build
martin@Kennny-PC:/mnt/c/Data/Dev/REPO/_dep/qemu-new/build$ ./qemu-system-arm -machine raspi0 -serial null -serial mon:stdio -kernel /mnt/c/Data/Dev/REPO/RPiZeroBare/05_kernel/build/kernel.img -nographic
Welcome to KIV/OS RPiOS kernel
*****
```

Emulátor qemu podporuje připojení debuggeru gdb. Je ale nutné dodat další parametry.

```
./qemu-system-arm -machine raspi0 -serial null -serial mon:stdio \
    -kernel /home/dev/kernel.img -nographic -S -gdb tcp::1234
```

Nově přibyly dva parametry:

- `-S` – po spuštění je emulace zastavena na první instrukci hostovaného systému, která by se provedla a bude čekat na spuštění z gdb
- `-gdb tcp::1234` – exportuje rozhraní pro gdb na TCP port 1234

Volitelně lze vynechat přepínač `-S`, pak se systém spustí a dovolí pouze připojení gdb do již běžícího systému.

Do emulace se pak lze připojit následujícím příkazem:

```
gdb-multiarch -ex 'set architecture arm' \
    -ex 'file kernel' \
    -ex 'target remote tcp:localhost:1234' \
    -ex 'layout regs'
```

Přepínač `-ex` provede příkaz konzole gdb. My potřebujeme tyto příkazy:

- `set architecture arm` – pro přepnutí na architekturu a instrukční sadu ARM
- `file kernel` – pro propojení běžícího kódu se zkompilevaným binárním souborem
- `target remote tcp:localhost:1234` – pro připojení se k běžící instanci qemu na TCP portu 1234
- `layout regs` – volitelné – přepnutí do pohledu, ve kterém vidíme registry a aktuální kód

Jakmile se nám povede připojit, můžeme ovládat emulaci (a debugging) např. následujícími příkazy (základní sada):

- **continue** (nebo jen **c**) – pokračuje v provádění příkazů, popř. spustí emulaci, pokud jsme ji vytvořili s přepínačem **-S**
- **si** – krok o jednu instrukci
- **s** – krokuje tak dlouho, dokud neopustí aktuální funkci
- **print <spec>** (nebo jen **p <spec>**) – vypíše obsah paměti na dané adrese; **<spec>** může být:
 - identifikátor (např. `moje_promenna`)
 - adresa prefixovaná hvězdičkou (např. `*0x8000`)
 - a jiné
- **print/x <spec>** (nebo jen **p/x <spec>**) – totéž, ale pro výpis v hexadecimální podobě
- **break <spec>** – nastaví (instrukční) breakpoint na dané místo; **<spec>** může být opět identifikátor, adresa prefixovaná hvězdičkou, apod.
- **clear** – vymaže všechny breakpointy
- **clear <spec>** – vymaže zadaný breakpoint
- **quit** – odpojí se od laděné instance qemu

Seznam příkazů pochopitelně není úplný, kompletní dokumentaci lze nalézt zde: https://ftp.gnu.org/old-gnu/Manuals/gdb/html_node/gdb_toc.html

6 Známé problémy

Emulace pomocí qemu není zdaleka dokonalá. Určité věci chybí, nebo jednoduše nefungují jak mají:

- **SYSTIMER** – systémový časovač; efektivně zamezuje použití preemptivního multitaskingu pomocí tohoto druhu časovače
- **bezdrátový adaptér** – momentálně neexistuje oficiální cesta, jak virtualizovat/emulovat WiFi (pro variantu RaspberryPi Zero W)