Libcsp 1.6 recept pre Yocto a Linux4Space

Obsah

Rýchly Yocto build

Na začiatok sa uistite že váš počítač spĺňa nasledujúce požiadavky

- aspoň 90 Gb voľného priestoru
- aspoň 8Gb RAM
- beží na ňom Fedora openSUSE, Debian, CENTos alebo Ubuntu
- má nainštalované nasledujúce aplikácie
    - git
    - tar
    - python
    - gcc
    - GNU make

```
$ sudo apt install git tat python3 gcc make -y
```

Je nutné nainštalovať nasledovné balíčky:

```
$ sudo apt install gawk wget git diffstat unzip texinfo gcc
build-essential chrpath socat cpio python3 python3-pip
python3-pexpect xz-utils debianutils iputils-ping python3-git
python3-jinja2 libegl1-mesa libsdl1.2-dev python3-subunit
mesa-common-dev zstd liblz4-tool file locales libacl1
```

A nastaviť lokalitu:

```
$ sudo locale-gen en_US.UTF-8
```

Poky je referenčná distribúcia Yocto obsahujúca všetko potrebné na vytvorenie vlastného operačného systému. Naklonujte si ju z GitHubu:

```
$ git clone git://git.yoctoproject.org/poky
```

Presuňte sa do naklonovanej zložky a vyberte si vetvu. Pre účely tohoto návodu si vyberieme nanbield:

```
$ cd poky
$ git branch -a
$ git checkout -t origin/nanbield -b my-nanbield
$ git pull
```

Prvý build

Je nutné inicializovať prostredie v ktorom budeme stavať svoj image. Použite `source oe-init-build-env:`

```
~/poky$ source oe-init-build-env

### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
    core-image-minimal
    core-image-full-cmdline
    core-image-sato
    core-image-weston
    meta-toolchain
    meta-ide-support

You can also run generated qemu images with a command like
'runqemu qemux86-64'.

Other commonly useful commands are:
 - 'devtool' and 'recipetool' handle common recipe tasks
 - 'bitbake-layers' handles common layer tasks
 - 'oe-pkgdata-util' handles common target package tasks
```

Začnite svoj prvý build. Pre účely tohoto návodu si vyberieme core-image-sato:

```
$ bitbake core-image-sato
```

Váš prvý build trvá niekoľko hodín. Bitbake musí posťahovať všetky potrebné knižnice a materiály nutné na vytvorenie operačného systému.

Po dokončení buildu spustíte svoj nový operačný systém príkazom:

```
$ runqemu qemux86-64
```

Prihlasovacie meno je root, heslo nie je nastavené.

V prípade nutnosti vytvorenia bootovateľného USB nájdete vytvorený obraz v ~/poky/build/tmp/deploy/images/qemux86-64/.

Stiahnutie receptu libcsp_1.6.bb

Do poky adresára si naklonujte Linux4Space repozitár meta-linux4space:

```
$ git clone git@gitlab.com:linux4space/yocto/meta-linux4space.git
```

Pridajte do svojich vrstiev linux4space vrstvu pomocou bitbake-layers add-layer:

```
$ bitbake-layers add-layer meta-linux4space
```

Ak nemáte poky adresár, je nutné buď prejsť prvou časťou tohoto návodu alebo naklonovať celý Linux4Space repozitár:

```
$ git clone git@gitlab.com:linux4space/linux4space-yocto.git
```

Inštalácia receptu libcsp_1.6.bb

Presuňte sa do zložky build/conf:

```
$ cd build/conf
```

Otvorte si súbor local.conf a na koniec súboru vložte:

```
IMAGE_INSTALL:append = " packagegroup-core-buildessential"
IMAGE_INSTALL:append = " libcsp libcsp-dev"
```

Spustite bitbake na vytvorenie libcsp balíku:

```
$ bitbake libcsp
```

Týmto ste si spravili základný libcsp balík a keď vytvoríte svoj ďalší image pomocou bitbake bude do neho knižnica automaticky pridaná.

V prípade že si chcete nastaviť knižnicu libcsp podľa vlastných potrieb je nutné vytvoriť .bbappend súbor.

Nastavenie knižnice libcsp

Knižnica libcsp používa buildsystém s názvom waf a má možnosti si niektoré funkcie dodatočne povoliť. V tomto návode si spoločne vytvoríme .bbappend súbor ktorými tieto možnosti nastavíme. .bbappend súbor je súbor ktorý je počas stavania balíku bitbakeom pridaný na koniec receptu a dajú sa ním dodatočne modifikovať premenné nachádzajúce sa v recepte.

Presunieme sa do zložky kde sa nachádza libcsp_1.6.bb recept. Tento recept sa tradične nachádza v meta-linux4space/recipes-connectivity/libcsp

```
$ cd meta-linux4space/recipes-connectivity/libcsp
```

Vytvoríme si súbor s názvom libcsp_1.6.bbappend a otvoríme si ho v textovom editori:

```
$ touch libcsp_1.6.bbappend
$ nano libcsp_1.6.bbappend
```

Libcsp recept má v sebe 3 premenné, ktoré sú určené na modifikáciu .bbappend súborom:
DRIVERS
DEPENDS
FLAGS

DRIVERS je premenná v ktorej nastavujete či chcete použiť niektorý z predprogramovaných ovládačov ktoré libcsp ponúka. Momentálna ponuka je USART, CAN a ZEROMQ:

```
DRIVERS += "--with-driver-usart=linux --enable-can-socketcan
--enable-if-zmqhub"
```

DEPENDS je premenná ktorú je nutné nastaviť pokiaľ ste do DRIVERS pridali can alebo zeromq:

```
DEPENDS += " zeromq libsocketcan"
```

Recepty pre zeromq a libsocketcan poky ani linux4space vrstvy neobsahujú. Je nutné si v takomto prípade do poky zložky stiahnuť meta-openembedded a pridať ju do svojich layerov:

```
$ git clone https://github.com/openembedded/meta-openembedded.git
$ bitbake-layers add-layer meta-openembedded
```

V takomto prípade je nutné do local.conf vložiť

```
IMAGE_INSTALL:append = " libsocketcan zeromq"
```

FLAGS je premenná v ktorej sa dajú nastaviť rôzne vlajky pre waf konfiguráciu. Kompletný zoznam vlajok je možné násť v repozitári libcsp v súbore wscript.

```
FLAGS += "--enable-python3-bindings --enable-crc32"
```

Súbor .bbappend uložte a zatvorte. V nasledujúcom postavení balíčku pomocou bitbake budú zmeny nastavené.

Stiahnutie a inštalácia receptu libcsp-demo.bb

libcsp-demo recept je recept určený na otestovanie funkčnosti libcsp knižnice vo vašom vstavanom systéme vytvorenom pomocou bitbake. Recept prekopíruje do rootfs výsledného OS niekoľko programov a kódov v C ktoré je možné si okamžite skompilovať a odskúšať.

Na použitie receptu je nutné na koniec local.conf vložiť:

```
IMAGE_INSTALL:append = " libcsp-demo"
```

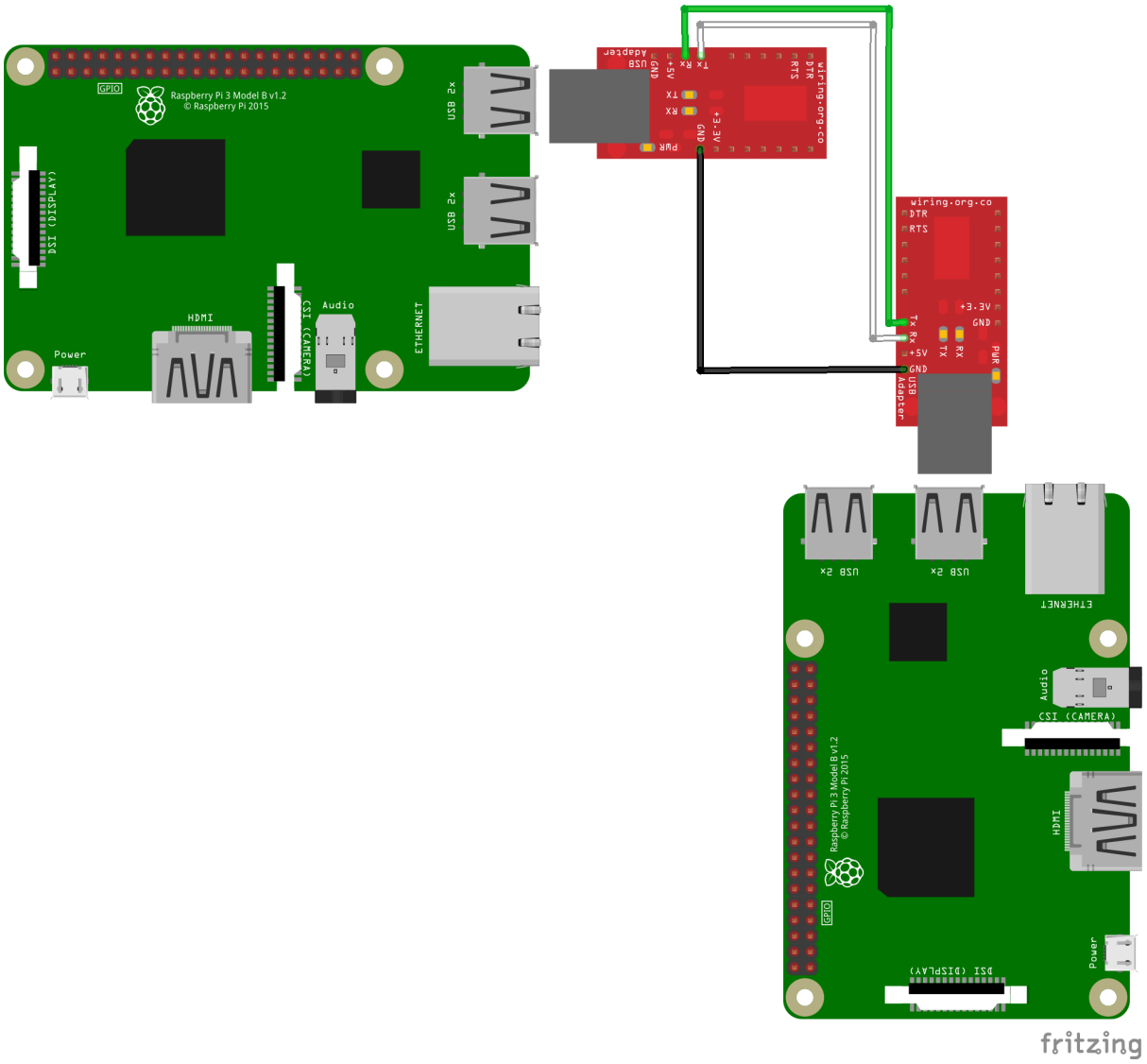
Balík pomocou bitbake vytvoríte nasledovným spôsobom:

```
$ bitbake libcsp-demo
```

Keď vytvoríte nový image systému a spustíte ho, v /home adresári sa bude nachádzať linuxforspace/testdata adresár. Recept vytvoril aj programy server, client a server_client.

Hardwarové zapojenie

Pre účely tohoto návodu sa používajú dve Raspberry Pi 3b dosky a dva USB-na-UART prevodníky. GND musí byť zapojené do GND, Tx a Rx káble musia byť prekrížené.



Testovanie funkčnosti

V prípade použitia libcsp-demo receptu príde váš nový systém s predinštalovanými server a client programami. V prípade že ste recept nepoužili je nutné si napísať vlastný kód a skompilovať ho nasledovným spôsobom:

```
$ gcc váš_program.c -lcsp
```

Program spustíte následovne:

```
$ ./a.out
```

V prípade použitia libcsp-demo receptu test začnete spustením server_client programu v testovacom móde:

```
$ server_client -t
```

Druhý test obsahuje komunikáciu medzi dvoma systémami. Uistite sa že USB-UART prevodníky sú správne pospájané káblami a zapojené do USB portov na obidvoch systémoch.
Do príkazovej riadky vložte:

```
$ ls -l /dev | grep ttyUSB
```

Ak je prevodník správne zapojený a funkčný, uvidíte výstup podobný tomuto:

```
crw-rw---- 1 root dialout   4,  64 Apr 16 15:45 ttyUSB0
```

Na prvom systéme spustite server nasledovým spôsobom:

```
$ server -k/dev/ttyUSB0 -a1
```

```
Initialising CSP
INIT KISS: device: [/dev/ttyUSB0], bitrate: 115200
Connection table
[00 0x14fc560] S:0, 0 -> 0, 0 -> 0, sock: (nil)
Interfaces
LOOP       tx: 00000 rx: 00000 txe: 00000 rxe: 00000
           drop: 00000 autherr: 00000 frame: 00000
           txb: 0 (0.0B) rxb: 0 (0.0B) MTU: 0
KISS       tx: 00000 rx: 00000 txe: 00000 rxe: 00000
           drop: 00000 autherr: 00000 frame: 00000
           txb: 0 (0.0B) rxb: 0 (0.0B) MTU: 252
Route table
1/5 LOOP
0/0 KISS
Server task started
Binding socket 0x55af30cdc338 to port 25
```

Na druhom systéme spustite client:

```
$ client -k/dev/ttyUSB0 -a2 -r1 -t
```

```
Initialising CSP
INIT KISS: device: [/dev/ttyUSB0], bitrate: 115200
Connection table
[00 0x16d0560] S:0, 0 -> 0, 0 -> 0, sock: (nil)
KISS       tx: 00000 rx: 00000 txe: 00000 rxe: 00000
           drop: 00000 autherr: 00000 frame: 00000
           txb: 0 (0.0B) rxb: 0 (0.0B) MTU: 252
Route table
2/5 LOOP
0/0 KISS
CLient task started
```

V prípade úspešnej komunikácie uvidíte clienta posielať pingy a server ich príjmať.

```
Ping address: 1, result 21 [mS]
```

```
SERVICE: Ping received
Packet received on MY_SERVER_PORT: Hello World (1)
```

Časté problémy

Pri vytváraní vlastného obrazu operačného systému sa môžu vyskytnúť určité problémy. V tejto časti sú spomenuté tie najčastejšie:

- zastavenie BitBake procesu kvôli nedostatku pamäte
  BitBake proces je mimoriadne náročný na pamäť. Ak nie je váš PC dostatočne silný, je možné že určité procesy spadnú a nebudú dokončené kvôli nedostatku operačnej pamäte. Najčastejším príkladom takéhoto balíčku je rust-llvm balíček, môže sa to ale vyskytnúť aj pri iných. Ak vám BitBake povie že určitý balíček sa nepodarilo spracovať, spracujte ho samostatne nasledovným spôsobom:

```
$ bitbake <meno_balíčku>
```

  Po dokončení spracovania samostatného balíčku znovu spustite vytvorenie obrazu.

- ak dostanete chybovú hlášku

```
Please use a locale setting which supports utf-8.
```

  Uistite sa že ste zmenili locale na taký ktorý podporuje UTF-8:

```
$ sudo locale-gen en_US.UTF-8
```