Laboratorul 06. Yocto

Yocto [https://www.yoctoproject.org/] este o colecție de utilitare, metadate și șabloane ce permit construirea/compilarea distribuțiilor de Linux pentru platforme embedded. Acest proiect este dezvoltat de Linux Foundation și condus de către Richard Purdie.

Proiectul Yocto se adresează atât utilizatorilor experimentați cât și utilizatorilor noi. Pentru utilizatorii experimentați, Yocto oferă posibilitatea de a crea distribuții personalizate pornind de la 0 sau de la imagini scheletice. Utilizatorii noi au la dispoziție o serie de exemple și un kernel ce poate fi folosit ca punct de pornire. De asemenea, aceste imagini de bază sunt disponibile pentru diferite platforme: ARM, PPC, MIPS, x86 etc.

Pentru a-și ajuta utilizatorii, Yocto vine cu o serie de aplicații: un sistem de build numit `Bitbake`, o interfață grafică numită `Hob`, un plug-in de Eclipse ce permite lucrul într-un IDE și o documentație.

Sursele Yocto pot fi găsite în repository-ul de la adresa **http://git.yoctoproject.org/cgit/cgit.cgi/poky [http://git.yoctoproject.org/cgit/cgit.cgi/poky]**.

Bitbake

Bitbake [http://docs.openembedded.org/bitbake/html/] este sistemul de build folosit de Yocto. Acesta este asemănător sistemului folosit uzual pe distribuțiile Linux, `make`. Ca și `make`, `bitbake` trebuie să determine ce acțiuni trebuie executate și apoi să le lanseze efectiv în execuție. Aceste acțiuni se determină în funcție de: comenzile date de către utilizator, datele proiectului și starea curentă a build-ului. Toate operațiile ce trebuie executate, dependențele dintre acestea, variabilele și instrucțiunile sunt ținute și citite din fișiere scrise în sintaxa specifică `bitbake`:

HelloWorld.bb

```
DESCRIPTION = "Hello World"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

PR = "r0"
SRC_URI = "file://myhelloworld.c \
           file://README.txt"

TARGET_CC_ARCH += "${LDFLAGS}"

do_compile() {
        ${CC} ${CFLAGS} ${LDFLAGS} ${WORKDIR}/myhelloworld.c -o ${WORKDIR}/myhelloworld
}

do_install() {
        install -d ${D}${bindir}

        install -m 0755 -t ${D}${bindir} ${WORKDIR}/myhelloworld
}
```

Asemănător cu un Makefile, putem folosi variabile de sistem pentru a specifica flag-uri, executabile, căi etc. Principala diferență constă în modul de organizare a fișierului. În aceste fișiere de configurare ne este permis să definim task-uri pentru fiecare etapă a procesului de build ce trebuie executată. Astfel, codul este mult mai bine organizat, mai ușor de urmărit și de depanat.

Un alt avantaj al lui `bitbake` este organizarea sa ierarhică. Atunci când se pornește un build, `bitbake` are o viziune de ansamblu asupra distribuției. Se vor citi mai întâi toate fișierele de configurare (rețetele) ce au legătură cu acea distribuție și abia apoi se va decide care sunt task-urile ce trebuie executate și în ce ordine se vor executa.

Fișiere speciale și variabile de sistem

Observați că fișierul din exemplul de mai sus are extensia `.bb`. Aceasta este o extensie specială specifică unei categorii de fișiere de configurare pentru `bitbake`. Există mai multe categorii de astfel de fișiere, fiecare exprimând un set diferit de metadate (dependențe, patch-uri, instrucțiuni) pentru o anumită componentă. Categoriile importante (în funcție de extensie):

- `.bb`: fișiere concise ce definesc operațiile ce trebuie executate. Poartă numele de **rețete**. O rețetă poate include alte rețete și poate moșteni dintr-o altă rețetă
- `.bbclass`: fișiere folosite pentru a preciza metadatele folosite uzual pentru operațiile comune de build sau împachetare. Aceste fișiere sunt de obicei moștenite de către rețete
- `.inc`: fișiere ce conțin un anumit set de definiții. Sunt folosite atunci când vrem să particularizăm procesul de build pentru o anumită versiune, arhitectură etc.
- `.bbappend`: fișiere ce conțin adăugiri sau componente opționale pentru rețete
- `.conf`: fișierul local de configurare al unui proiect

Pe lângă aceste fișiere, pentru a personaliza un proiect/build, avem la dispoziție și o serie de variabile de sistem:

- `BBFILES`: variabila ce spune sistemului `bitbake` care sunt rețetele disponibile
- `SRC_URI`: identifică fișierele care trebuiesc incluse în directorul de lucru special făcut pentru rețeta respectivă
- `BB_NUMBER_THREADS`: denotă numărul de thread-uri de `bitbake` care să ruleze
- `PARALLEL_MAKE`: modul în care va fi făcută compilarea și numărul de thread-uri de compilare care vor porni. Valoarea acestei variabile este aceeași ca în cazul sistemului `make`: -j <num_threads>
- `MACHINE`: denumirea sistemului (mașinii) pentru care se realizează compilarea
- `BBMASK`: lista de pachete ce vor fi ignorate în momentul compilării

Rețete

Fișierele de configurare prezentate anterior poartă denumirea de rețete și au uzual extensia `.bb`. În cadrul unui proiect (o distribuție) acestea definesc operațiile ce trebuie executate. În funcție de complexitatea și dimensiunea proiectului, acesta va parcurge un anumit număr de etape. Dacă proiectul este mic și presupune, spre exemplu, doar compilarea unor surse, atunci este suficientă o rețetă ce conține un task `do_compile`. Un proiect mai mare va trece însă cel puțin prin etapele de `fetch`, `configure`, `compile` și `install`.

Pentru un astfel de proiect va exista câte o rețetă ce conține un task de tip `do_` pentru fiecare operație. Folosind apoi proprietatea rețetelor de a include sau a specifica dependențe față de alte rețete, se creează o ierarhie ce pornește de la o rețetă top level și parcurge și execută în ordine toate operațiile.

Directivele ce pot fi folosite într-o rețetă pentru a include sau moșteni alte fișiere de configurare sunt:

- include <file_name>: include fișierul cu numele <file_name>. Este folosită variabila BBPATH pentru a căuta fișierul
- require [<path>]<file_name>: un tip special de includere a fișierului <file_name>. Se va încerca includerea fișierului ca și în cazul lui `include`, dar operația va eșua dacă nu există fișierul în locația specificată
- inherit <file_name>: include definițiile din fișierul <file_name>.bbclass, dacă acesta există

Layer

Un set de rețete și alte fișiere specifice `bitbake` ce au legătura cu o anumită funcționalitate sau o anumită componentă pot fi organizate într-un **layer**. Layer-ul este un fișier ce conține referința către o configurație

și setul de rețete ce trebuiesc executate pentru a obține o anumită distribuție.

LayerExample

```
meta-layer:
  - conf
  - recipes-core
      - important_recipe
          - x.bb
  - recipes-category1
      - recipe-1a
          - y.bb
          - t.bbappend
      - recipe-1b
          - ...
  - recipes-category2
      - recipe-2a
          - ...
      - recipe-2b
          - ...
```

Structura unui layer este următoarea: denumirea layer-ului urmată de o serie de categorii de rețete. Fiecare categorie conține lista rețetelor componente și fișierele de configurare aferente fiecarei dintre acestea.

Atunci când parsează un fișier de tip `bblayer`, `bitbake` va configura variabila de sistem BBPATH cu locațiile în care se găsesc rețetele necesare. În cadrul acestui proces de construire a variabilei BBPATH, dacă ierarhia conține rețete cu nume duplicat, pot apărea conflincte sau situații neașteptate în timpul compilării, deoarece fișierele de configurare pentru o rețetă pot fi incărcate dintr-o locație diferită față de cea dorită.

Specificarea layer-elor ce se doresc a fi incluse într-un proiect se face prin scrierea unui fișier numit `bblayers.conf`. Structura acestuia este următoarea:

```
BBLAYERS ?= " \
<path_to>/poky-rasp/meta \
<path_to>/poky-rasp/meta-yocto \
<path_to>/poky-rasp/meta-yocto-bsp \
<path_to>/poky-rasp/meta-raspberrypi \
"
```

Acest fișier pune în variabila de sistem BBLAYERS căile către fiecare locație în care se găsește fișierul `bblayer` al layer-ului dorit.

Instalare Yocto si configurare pentru RaspberryPi

Sursele proiectului Yocto pot fi descărcate urmărind instrucțiunile de aici [https://www.yoctoproject.org/downloads] sau clonate direct din repository-ul mentionat la începutul laboratorului.

Următorul pas este să obținem layer-ele și configurațiile necesare pentru a compila imaginea corespunzătoare pentru sistemul dorit. În cazul nostru, acest sistem este `RaspberryPi`. Acest layer poate fi obținut clonând repository-ul de la adresa: https://github.com/djwillis/meta-raspberrypi.git [https://github.com/djwillis/meta-raspberrypi.git]

După ce avem sursele Yocto și layer-ul necesar pentru dezvoltarea unei imagini de RaspberryPi, trebuie să inițializăm mediul de compilare și directorul de lucru. Pentru aceasta, trebuie rulată următoarea comandă în directorul ce conține sursele Yocto:

```
source oe-init-build-env <path_to_working_directory>/<rpi_build_folder>/
```

Apoi vom adăuga intrări în fișierele de configurare a mediului pentru a ne asigura că parametrii folosiți la compilare sunt corecți și că va fi inclus layer-ul pentru RaspberryPi. În directorul inițializat mai devreme, `<path_to_working_directory>/<rpi_build_folder>/`, se găsește directorul `conf` ce conține fișierul de configurare `local.conf`. Trebuie să ne asigurăm că:

1. `BB_NUMBER_THREADS` are valoarea egală cu numărul de thread-uri de build ce dorim să fie create. Valoarea inițială este 4
2. `PARALLEL_MAKE` are valoarea egală cu numărul de procesoare disponibile.

Valoarea trebuie să aibă forma `-j x`, unde x este numărul de procesoare disponibile.

1. `MACHINE` are valoarea "raspberrypi"
2. `BBMASK` va exclude pachetele ce nu sunt suportate de core-ul Yocto. Spre exemplu, în unele versiuni mai vechi ale layer-ului de RaspberryPi, acestea erau: "meta-raspberrypi/recipes-multimedia/libav|meta-raspberrypi/recipes-core/systemd"

Toate valorile acestor variabile sunt șiruri de caractere! O atribuire precum `MACHINE = my_machine_name` va genera eroare la parsare. Atribuirea corectă este `MACHINE = "my_machine_name"`

Tot în directorul `conf` există și fișierul `bblayers.conf` ce conține informația legată de layer-ele ce vor fi compilate. Trebuie să adăugăm la variabila `BBLAYERS` o cale către layer-ul de RaspberryPi. Revedeți secțiunea **Layers** și exemplul de acolo.

Tot ce mai rămâne acum de făcut este să ne asigurăm că utilitarele folosite de `bitbake` sunt instalate în sistem ca să putem compila imaginea. În cazul în care acestea nu există, se pot instala folosind una din comenzile de mai jos, în funcție de versiunea sistemului:

Fedora

```
sudo yum groupinstall "development tools"
```

Ubuntu

```
sudo apt-get install sed wget cvs subversion git-core coreutils \
unzip texi2html texinfo libsdl1.2-dev docbook-utils gawk \
python-pysqlite2 diffstat help2man make gcc build-essential \
g++ desktop-file-utils chrpath libgl1-mesa-dev libglu1-mesa-dev \
mercurial autoconf automake groff libtool xterm
```

Compilarea unei imagini inițiale pentru o arhitectură poate dura de la cateva ore la zeci de ore, în funcție de performanța sistemului pe care se face compilarea. Procesul este unul îndelungat deoarece sistemul de build Yocto își va construi singur toate componentele necesare precum biblioteci, cross-compiler, etc. Spre exemplu, observați că nu mai este necesară instalarea unui cross-compiler. Acesta va fi creat la începutul build-ului, în două etape: mai întâi va fi compilată biblioteca `glibc` pentru platforma aleasă și apoi se va crea cross-compiler-ul propriu-zis.

Avantajul acestui mod de lucru este că efortul necesar pentru pregătirea și inițializarea unui build este mic și nu există dependențe numeroase față de alte biblioteci externe sau alte utilitare. Dezavantajul major este timpul necesar pentru o compilare.

Exerciții

0. Urmăriți instrucțiunile prezentate în README-ul din resursele laboratorului pentru a crea o imagine de bază folosind Yocto.

1. În urma exercițiului 0 ați creat o imagine de bază. Rulați această imagine în Qemu și verificați că mașina emulată de RaspberryPi pornește până la user login.

2. Pentru a adăuga un layer nou, vă trebuie un director în root-ul Yocto: `meta-labsi` cu fișierul `conf/layer.conf` identic cu cel din `meta-raspberrypi`. Iată apoi scheletul unei rețete pentru o aplicație hello world:

hello.bb

```
DESCRIPTION = "Hello World"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

PR = "r0"

#TODO: scrieți în SRC_URI calea către fișierul dorit.
SRC_URI = ""

S = "${WORKDIR}"

TARGET_CC_ARCH += "${LDFLAGS}"

do_compile() {

}

do_install() {

}
```

Scrieți rețeta în folder-ul corespunzător, pregătiți un fișier hello.c (va afișa "Hello World!") și scrieți funcțiile `do_compile` și `do_install` Layer-ul nou trebuie adăugat în `<yocto_dir>/build/conf/bblayers.conf`

3. Pachetul cu aplicația `hello` nu va fi instalat nicăieri dacă o imagine nu are nevoie de el. Trebuie făcută o nouă imagine, pe modelul `../meta-raspberrypi/recipes-core/images/rpi-basic-image.bb`

- Includeti restul de rețete folosite de `rpi-basic-image` in imaginea voastră. Folosiți directiva `require`. Formatul este: `require poky/<cale_reteta_rpi_basic_image>`.
- Oferiti drepturi de scriere tuturor utilizatorilor pe directorul `/var/tmp/`.
- Această nouă rețetă pentru imaginea `rpi-hello-image` trebuie pusă în layer-ul nou
- Noua imagine trebuie să adauge pachete noi pentru aplicația `hello`, adăugând (cu += în variabila IMAGE_INSTALL)
- Apelați bitbake cu noua imagine și verificați în Qemu rezultatul obținut!

Bonus 4. Pentru a modifica numele pe care îl publică un RaspberryPi este suficient să modificăm `/etc/hostname`. Pentru aceasta am putea modifica fișierul din imagine, dar dezavantajul este că ar trebui să facem această modificare la fiecare recompilare. Am prefera să obținem direct o imagine cu numele nou, iar ca să obținem acest lucru trebuie să modificăm rețetele Bitbake care stau la baza pachetelor instalate în imagine. Fisierul `hostname` ar trebui sa fie plasat aici: `<poky_path>/meta/recipes-core/base-files/`.

Daca intalniti o eroare de tipul `Temporary repodata directory already exists!`, puteti sterge cu incredere directorul `.repodata/` afisat in mesajul de eroare.

Pentru a adăuga un fișier `hostname` în imagine trebuie să:

- aveți un fișier `hostname` în locația unde sunt toate fișierele de inclus
- modificați fișierul `.bbappend` (adăugirea la rețetă), adăugând în variabila SRC_URI noul fișier (cu +=) și implementând o funcție cu numele `do_install_append()`, care se va executa în cadrul rețetei după `do_install`
- Rulați apoi `bitbake rpi-basic-image`

- Verificați rezultatul rulând noua imagine cu Qemu!

Scheletul arata in felul urmator:

example.bbappend

```
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"

SRC_URI += "file://example_file\
            "

do_install_append () {
   <TODO>
}
```

Resurse

- Setup yocto
- Soluție laborator (disponibilă începând cu 30.11.2020)
- Imagine kernel [https://drive.google.com/file/d/0B0lgiPZNMMyvMWEtZS1zME9lQjQ]4
- Reference Manual Yocto [http://www.yoctoproject.org/docs/1.8/ref-manual/ref-manual.html]
- Manual Yocto [https://www.yoctoproject.org/docs/2.0/mega-manual/mega-manual.html]
- Manual BitBake [https://www.yoctoproject.org/docs/1.6/bitbake-user-manual/bitbake-user-manual.html]
- Ubuntu 18 - Yocto common errors and solutions [https://docs.google.com/document/d/1lWhRp5teVswWJnbeSxCSfhxvDoMTJQ5S9ACuc37eDs8/edit?usp=sharing]

si/laboratoare/06.txt · Last modified: 2021/01/08 20:47 by dragos_florin.costea