Tổng quan Yocto, Poky, BitBake, OpenEmbedded

Và các công cụ hỗ trợ debug

GVHD: Anh Phú

Người thực hiện: Thông
Hiếu

Tp. Hồ Chí Minh, Tháng 5/2024

Mục lục

Danh sách hình vẽ

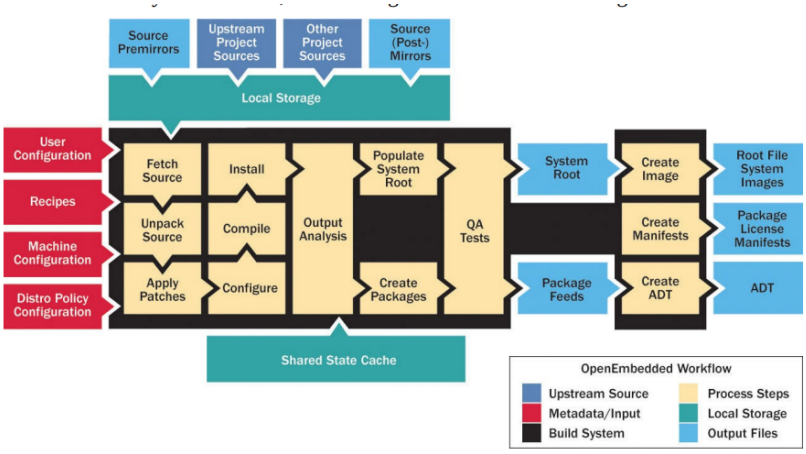
I Yocto Project

Dự án Yocto là một dự án mã nguồn mở cung cấp một hệ sinh thái công cụ và tài nguyên để xây dựng các bản phân phối Linux tùy chỉnh cho các thiết bị nhúng và hệ thống nhúng. Dự án này cung cấp một số thành phần quan trọng như sau:

- OpenEmbedded Core Layer: Đây là lớp cốt lõi của hệ thống xây dựng, chứa các công cụ và các recipe (các tập lệnh mô tả cách xây dựng các thành phần phần mềm) để tạo ra các bản phân phối Linux.
- BitBake: Đây là công cụ xây dựng chính của Yocto Project. BitBake đọc các recipe và quy trình xây dựng để tạo ra các gói phần mềm và các bản phân phối Linux.
- Metadata Layers: Yocto Project sử dụng một cấu trúc lớp (layer) linh hoạt, cho phép người dùng mở rộng và tùy chỉnh dự án bằng cách thêm các lớp metadata mới. Các lớp này có thể chứa các recipe, cấu hình và mã nguồn tùy chỉnh.
- Poky: Poky là bản phân phối tham khảo của Yocto Project. Nó cung cấp một bản mẫu để bắt đầu xây dựng các bản phân phối Linux tùy chỉnh, bao gồm cả các công cụ và cấu hình mặc định.
- BitBake Metadata (BBM) Layer: Đây là một lớp metadata cung cấp các recipe mặc định và các tập lệnh xây dựng cho Poky.
- Toaster: Là một giao diện web đồ họa cho BitBake, giúp người dùng quản lý quá trình xây dựng và theo dõi tiến độ của các bản phân phối Linux.

II OpenEmbedded Build System

OpenEmbedded là một hệ thống xây dựng mã nguồn mở được thiết kế để tạo ra các bản phân phối Linux tùy chỉnh cho các thiết bị nhúng và hệ thống nhúng. Nó cung cấp một cách linh hoạt và mạnh mẽ để tạo ra các bản phân phối Linux được tùy chỉnh cho một loạt các nền tảng phần cứng. OpenEmbedded bao gồm BitBake - một công cụ xây dựng - và một số metadata layers (lớp dữ liệu siêu dữ liệu) chứa các recipe (các tập lệnh mô tả cách xây dựng phần mềm) và cấu hình để tạo ra các hệ điều hành nhúng. Nó được sử dụng trong nhiều dự án nhúng và là một phần quan trọng của Dự án Yocto.

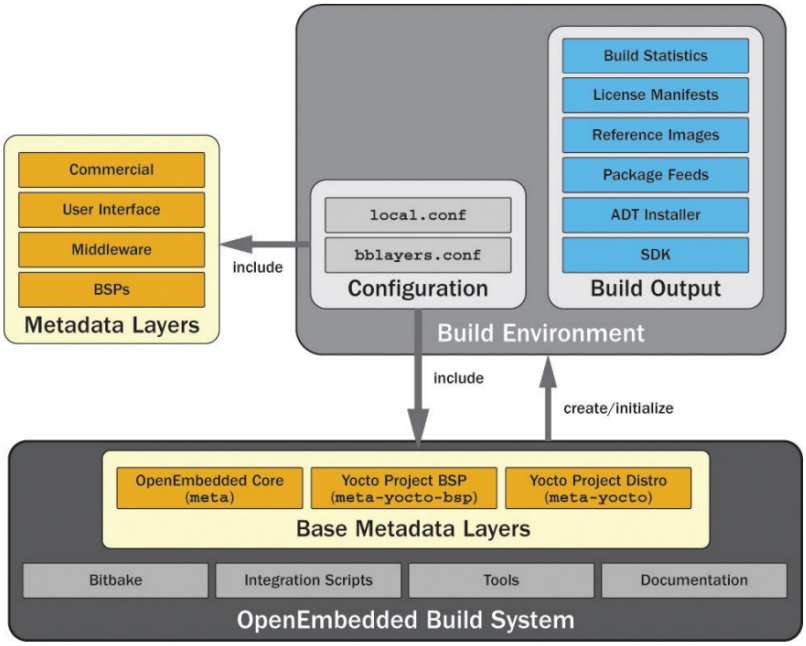


Hình 1: *OpenEmbedded workflow*

1 Các thành phần trong OpenEmbedded Build System

OpenEmbedded Build System gồm 3 thành phần:

- Build system.
- Build environment.
- Metadata layers.



Hình 2: *Kiến trúc Poky*

1.1 Build System

Cấu trúc build system:

```
yocto@yocto-dev:~/yocto$ tree -L 1 poky
poky
├── bitbake
├── documentation
├── LICENSE
├── meta
├── meta-hob
├── meta-selftest
├── meta-skeleton
├── meta-yocto
├── meta-yocto-bsp
├── oe-init-build-env
├── oe-init-build-env-memres
├── README
├── README.hardware
└── scripts

9 directories, 5 files
```

Hình 3: *OpenEmbedded Build System Structure*

Trong đó:

1. meta-hob: Lớp metadata được sử dụng bởi giao diện người dùng đồ họa Hob cho BitBake. Hob là một công cụ giao diện đồ họa được cung cấp bởi Dự án Yocto để hỗ trợ việc tạo và quản lý các bản phân phối Linux tùy chỉnh.

2. meta-selftest: Lớp dành cho việc kiểm thử BitBake, được sử dụng bởi kịch bản oe-selftest. Kịch bản oe-selftest là một bộ kiểm thử tự động được cung cấp bởi Dự án Yocto để đảm bảo tính ổn định và chất lượng của BitBake.

3. meta-skeleton: Lớp mẫu bạn có thể sử dụng để tạo ra các lớp của riêng mình. Nó cung cấp các mẫu và cấu trúc cơ bản để bắt đầu phát triển các lớp metadata tùy chỉnh.

4. meta-yocto: Lớp phân phối của Dự án Yocto. Lớp này cung cấp các cấu hình và recipe cần thiết để xây dựng các bản phân phối Linux tùy chỉnh với các tính năng và công cụ hỗ trợ của Dự án Yocto.

5. meta-yocto-bsp: Lớp hỗ trợ phần cứng (BSP) của Dự án Yocto. Lớp này cung cấp các cấu hình và recipe cần thiết để hỗ trợ và phát triển các bản phân phối Linux cho các bo mạch và thiết bị cụ thể.

1.2 Build Environment

OpenEmbedded build system sẽ làm việc trong 1 môi trường được thiết lập sãn:

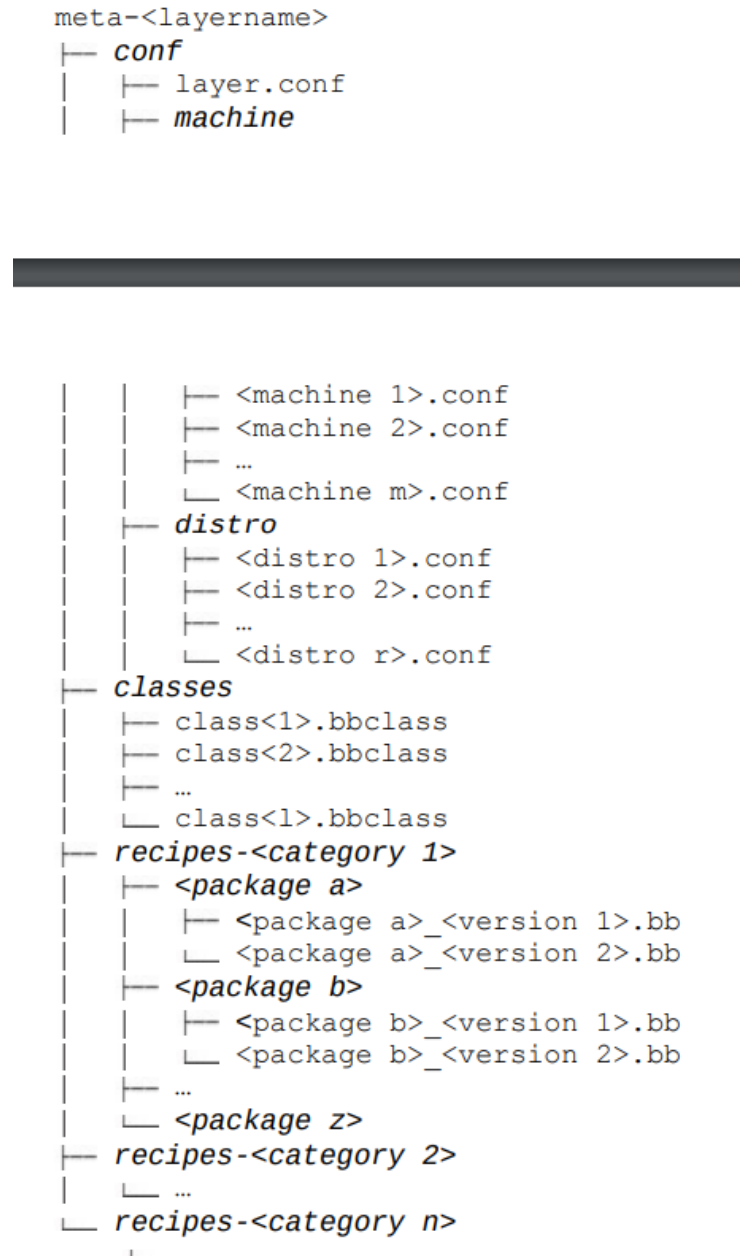
```
yocto@yocto-dev:~/yocto$ tree -L 2 build
x86/
├── bitbake.lock
├── cache
│   ├── bb_codeparser.dat
│   ├── bb_persist_data.sqlite3
│   └── local_file_checksum_cache.dat
├── conf
│   ├── bblayers.conf
│   ├── local.conf
│   └── sanity_info
└── tmp
    ├── abi_version
    ├── buildstats
    ├── cache
    ├── deploy
    │   ├── images
    │   ├── licenses
    │   ├── deb
    │   ├── ipk
    │   └── rpm
    ├── log
    ├── qa.log
    ├── saved_tmpdir
    ├── sstate-control
    ├── stamps
    ├── sysroots
    ├── work
    │   ├── all-poky-linux
    │   ├── i586-poky-linux
    │   ├── qemux86-poky-linux
    │   └── x86_64-linux
    └── work-shared
```

Hình 4: *Build Environment Structure*

2 Metadata Layer

Các lớp metadata là các bộ chứa để nhóm và tổ chức các recipe, lớp, tập tin cấu hình và các dữ liệu siêu dữ liệu khác thành các thực thể logic. Các lớp thường được xây dựng và mở rộng lên nhau. Lớp OE Core tạo nên nền tảng cho kiến trúc lớp của

hệ thống xây dựng Poky. Nó cung cấp các recipe cho một tập hợp cơ bản các gói phần mềm cần thiết cho hầu hết các ngăn xếp hệ điều hành Linux như bootloaders, graphics, networking, và nhiều packages khác. OE Core cũng cung cấp các lớp cơ bản để xây dựng các gói phần mềm, đóng gói phần mềm bằng các hệ thống quản lý gói, tạo các filesystem image và mở rộng chức năng của BitBake.



Hình 5: *Metadata Layer Structure*

III BitBake

BitBake là một thành phần chính được phát triển chung bởi OpenEmbedded và Dự án Yocto, và là một phần lõi của hệ thống xây dựng OpenEmbedded, được chia sẻ bởi cả hai dự án. Các phiên bản của Yocto Project trong Poky bao gồm một phiên bản của BitBake phù hợp với metadata của Poky cho phiên bản cụ thể đó. Điều này đảm bảo rằng BitBake và metadata của Poky tương thích và hoạt động một cách hài hòa trong việc xây dựng các bản phân phối Linux tùy chỉnh dựa trên Poky.

```
yocto@yocto-dev:~/bitbake$ tree -L 3 bbhello
bbhello/
├── classes
│   └── base.bbclass
├── conf
│   ├── bblayers.conf
│   └── bitbake.conf
└── meta-hello
    ├── conf
    │   └── layer.conf
    └── recipes-editor
        └── nano
```

Hình 6: *Ví dụ về cấu trúc bitbake*

IV Troubleshooting

1 Logging

BitBake ghi lại tất cả các sự kiện xảy ra trong quá trình build. Các sự kiện được ghi lại bởi BitBake bao gồm:

- Các câu lệnh Debug được chèn vào trong metadata thực thi.
- Đầu ra từ bất kỳ lệnh nào được thực thi bởi các nhiệm vụ và mã khác.
- Các thông báo lỗi được phát ra từ bất kỳ lệnh nào được thực thi bởi các nhiệm vụ và mã khác.

1.1 General Log Files

BitBake lưu trữ tổng thể log file ở thư mục được gán bởi biến **Log_DIR**. Mặc định, các files log nằm trong **LOG_DIR = "$TMPDIR/log"**

Ví dụ về 1 tệp log ghi lại nhật ký quá trình build:

```
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION        = "1.21.1"
BUILD_SYS         = "x86_64-linux"
NATIVELSBSTRING   = "Fedora-18"
TARGET_SYS        = "i586-poky-linux"
MACHINE           = "qemux86"
DISTRO            = "poky"
DISTRO_VERSION    = "1.5+snapshot-20140210"
TUNE_FEATURES     = "m32 i586"
TARGET_FPU        = ""
meta
meta-yocto
meta-yocto-bsp    = "master:095bb006c3dbbfbdfa05f13d8d7b50e2a5ab2af0"

NOTE: Preparing runqueue
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Running noexec task 2051 of 2914 (ID: 4, /develop/yocto/yocto-git/poky/meta/
recipes-core/images/core-image-minimal.bb, do_fetch)
NOTE: Running noexec task 2052 of 2914 (ID: 0, /develop/yocto/yocto-git/poky/meta/
recipes-core/images/core-image-minimal.bb, do_unpack)
NOTE: Running noexec task 2053 of 2914 (ID: 1, /develop/yocto/yocto-git/poky/meta/
recipes-core/images/core-image-minimal.bb, do_patch)
NOTE: Running noexec task 2910 of 2914 (ID: 9, /develop/yocto/yocto-git/poky/meta/
recipes-core/images/core-image-minimal.bb, do_package_write)
NOTE: Running task 2911 of 2914 (ID: 8, develop/yocto/yocto-git/poky/meta/recipes-
core/images/core-image-minimal.bb, do_populate_lic)
NOTE: Running task 2912 of 2914 (ID: 7, develop/yocto/yocto-git/poky/meta/recipes-
core/images/core-image-minimal.bb, do_rootfs)
NOTE: recipe core-image-minimal-1.0-r0: task do_populate_lic: Started
NOTE: recipe core-image-minimal-1.0-r0: task do_populate_lic: Succeeded
NOTE: recipe core-image-minimal-1.0-r0: task do_rootfs: Started
NOTE: recipe core-image-minimal-1.0-r0: task do_rootfs: Succeeded
NOTE: Running noexec task 2914 of 2914 (ID: 12, /develop/yocto/yocto-git/poky/
meta/recipes-core/images/core-image-minimal.bb, do_build)
NOTE: Tasks Summary: Attempted 2914 tasks of which 2907 didn't need to be rerun
and all succeeded.
```

Hình 7: *Cooker Logfile*

1.2 Task Log Files

BitBake tạo ra một tệp nhật ký cho mỗi nhiệm vụ mà nó chạy cho mỗi recipe. Mặc định, log của các task được cấu hình trong: **T = "$WORKDIR/temp"**.

Đường dẫn trên sẽ áp dụng cho đa số các case. Tuy nhiên với **clean task**, task này sẽ thực hiện xóa đi thư mục và các thư mục con của nó. Vậy nên, quá trình **clean task** sẽ được lưu lại ở log file được cấu hình **T_task-clean = "$LOGDIR/cleanlogs/$PN"**

Các tệp nhật ký cho các nhiệm vụ được đặt tên là log.do_<taskname>.<pid>, trong đó pid là ID quy trình của nhiệm vụ khi nó được chạy bởi BitBake. ID quy trình được sử dụng để phân biệt các nhật ký nhiệm vụ của nhiều lần thực thi của cùng một nhiệm vụ. Điều này làm cho việc so sánh các kịch bản trước và sau, so sánh việc thực thi nhiệm vụ thành công với việc thất bại trở nên dễ dàng. BitBake cung cấp rất nhiều sự trợ giúp khi xác định vị trí tệp nhật ký cho nhiệm vụ thất bại

bằng cách in đường dẫn đầy đủ của nó ra màn hình cùng với thông báo lỗi.

```
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION        = "1.21.1"
BUILD_SYS         = "x86_64-linux"
NATIVELSBSTRING   = "Fedora-18"
TARGET_SYS        = "i586-poky-linux"
MACHINE           = "qemux86"
DISTRO            = "poky"
DISTRO_VERSION    = "1.5+snapshot-20140211"
TUNE_FEATURES     = "m32 i586"
TARGET_FPU        = ""
meta-mylayer      = "<unknown>:<unknown>"
meta
meta-yocto
meta-yocto-bsp    = "master:095bb006c3dbbfbdfa05f13d8d7b50e2a5ab2af0"

NOTE: Preparing runqueue
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
ERROR: This autoconf log indicates errors, it looked at host include and/or
library paths while determining system capabilities.
Rerun configure task after fixing this. The path was
'/develop/yocto/yocto-git/x86/tmp/work/i586-poky-linux/nano/2.3.1-r0/nano-
2.3.1'
ERROR: Function failed: do_qa_configure
ERROR: Logfile of failure stored in:
…/tmp/work/i586-poky-linux/nano/2.3.1-r0/temp/log.do_configure.17865
ERROR: Task 5 (…/meta-mylayer/recipes-apps/nano/nano_2.3.1.bb,
do_configure) failed with exit code '1'
NOTE: Tasks Summary: Attempted 550 tasks of which 545 didn't need to be rerun
and
1 failed.
No currently running tasks (550 of 558)

Summary: 1 task failed:
  …/meta-mylayer/recipes-apps/nano/nano_2.3.1.bb, do_configure
Summary: There were 3 ERROR messages shown, returning a non-zero exit code.
```

Hình 8: *Task Failure*

Các dòng bắt đầu bằng ERROR chứa thông tin liên quan đến sự cố khi build, như gợi ý về nguyên nhân gốc rễ của vấn đề, nhiệm vụ đã thất bại và đường dẫn đầy đủ đến tệp nhật ký.

2 Task Execution

BitBake cung cấp cho người dùng dòng lệnh **listtasks** để hiển thị tất cả các task được thực thi.

Đầu ra là một danh sách tất cả các nhiệm vụ theo thứ tự bảng chữ cái với mô tả ngắn về mỗi nhiệm vụ làm gì. Tuy nhiên, nó không cung cấp thông tin về các nhiệm vụ nào được thực thi trong quá trình xây dựng thông thường và chúng được thực hiện theo thứ tự nào.

```
user@buildhost:~$ bitbake busybox -c listtasks

[… omitted for brevity …]

NOTE: Preparing runqueue
NOTE: Executing RunQueue Tasks
do_build                        Default task for a recipe - depends on all
                                other normal tasks required to 'build' a
                                recipe
do_checkuri                     Validates the SRC_URI value
do_checkuriall                  Validates the SRC_URI value for all recipes
                                required to build a target
do_clean                        Removes all output files for a target
do_cleanall                     Removes all output files, shared state
                                cache, and downloaded source files for a
                                target
do_cleansstate                  Removes all output files and shared state
                                cache for a target
do_compile                      Compiles the source in the compilation
                                directory
do_compile_ptest_base           Compiles the runtime test suite included in
                                the software being built
do_configure                    Configures the source by enabling and
                                disabling any build-time and configuration
                                options for the software being built
do_configure_ptest_base         Configures the runtime test suite included
                                in the software being built
do_devshell                     Starts a shell with the environment set up
                                for development/debugging

[… omitted for brevity …]
```

Hình 9: *Listing Tasks for a Recipe*

2.1 Executing Specific Tasks

Lệnh **bitbake <target> -c <task>** được sử dụng để thực hiện một nhiệm vụ cụ thể cho một công thức (recipe) đã được chỉ định. Đây là cú pháp cơ bản của BitBake, một công cụ được sử dụng trong quá trình xây dựng và quản lý phần mềm trong hệ thống yocto.

Ví dụ: **bitbake busybox -c compile**. Lệnh này sẽ thực hiện nhiệm vụ biên dịch (compile) cho gói busybox.

2.2 Task Script Files

Mỗi lần thực thi một nhiệm vụ, BitBake tạo ra một tệp tập lệnh chứa các lệnh nó thực thi khi chạy nhiệm vụ. Các tệp tập lệnh của nhiệm vụ được đặt trong cùng một thư mục với các tệp nhật ký của nhiệm vụ:

T = "$WORKDIR/temp"

Các tệp tập lệnh nhiệm vụ có tên là:

run.do_<taskname>.<pid>

Phần mở rộng <pid> được sử dụng để phân biệt các tệp tập lệnh cho cùng một nhiệm vụ từ nhiều lần thực thi. Tệp run.do_<taskname> không có phần mở rộng quy trình là một liên kết tượng trưng trỏ đến tệp tập lệnh mà BitBake đã thực thi

cho lần chạy gần nhất của nhiệm vụ.

Các tệp tập lệnh cho các nhiệm vụ Python được thực thi từ môi trường BitBake mà chúng kế thừa và truy cập vào từ điển dữ liệu của BitBake. Các tệp tập lệnh cho các nhiệm vụ shell chứa toàn bộ môi trường và được thực thi bằng cách sinh một quy trình. Trong nhiều trường hợp, chúng có thể được chạy trực tiếp từ dòng lệnh.

Tệp log.task_order chứa một danh sách các nhiệm vụ và các tệp tập lệnh tương ứng của chúng cùng với các ID quy trình cho việc thực thi gần đây nhất.

3 Analyzing Metadata

Toàn bộ quy trình xây dựng của BitBake được điều khiển bởi metadata. Metadata thực thi cung cấp các bước quy trình, được cấu hình thông qua các giá trị của các biến khác nhau. Nhiều biến này phụ thuộc vào ngữ cảnh thực thi của công thức cụ thể. Ví dụ, các biến như SRC_URI được định nghĩa bởi mỗi công thức. Nhiều biến cũng tham chiếu đến các biến khác, được mở rộng trong ngữ cảnh thực thi. Việc thiết lập biến có điều kiện và gắn thêm là các khái niệm mạnh mẽ để điều chỉnh động ngữ cảnh thực thi nhưng cũng thêm phức tạp khi gỡ lỗi các lỗi xây dựng.

Từ các lỗi gõ sai trong tên biến và cài đặt đến sự mở rộng biến không đúng và các phép gán có điều kiện, có nhiều cơ hội cho quá trình xây dựng thất bại hoặc sản xuất đầu ra không đúng. Do đó, quan trọng là có khả năng phân tích cài đặt biến mà hệ thống xây dựng sử dụng trong các ngữ cảnh khác nhau. Lệnh

$bitbake - e$

in toàn bộ từ điển dữ liệu cho môi trường BitBake toàn cục ra màn hình console. Điều này hữu ích để xem các cài đặt mặc định trước khi chúng có thể bị ghi đè bởi ngữ cảnh công thức cụ thể. Sử dụng lệnh với một mục tiêu hiển thị các mục từ điển dữ liệu cho mục tiêu cụ thể đó:

$bitbake - e < target >$

Một nhược điểm của lệnh này là nó liệt kê cả biến lẫn hàm, vì các hàm không gì khác ngoài metadata có thể thực thi. BitBake lưu trữ biến và hàm trong cùng một từ điển dữ liệu. Liệt kê mã của mỗi hàm làm cho đầu ra khá cồng kềnh để phân tích. Thật không may, cả BitBake lẫn OE Core đều không cung cấp chức năng để chỉ liệt kê các biến. Tuy nhiên có thể sử dụng

$bitbake < target > -cshowvars$

để liệt kê tất cả các biến nhưng không liệt kê các hàm của môi trường thực thi của mục tiêu (target) theo thứ tự bảng chữ cái.

4 Development Shell

Những lỗi build xuất phát từ việc biên dịch và liên kết các nguồn thành các đối tượng, thư viện và các tệp thực thi là rất khó để gỡ lỗi khi build chéo. Chúng ta không thể chỉ cần chuyển đến thư mục nguồn, gõ lệnh make, xem thông báo lỗi và sửa chữa vấn đề. Môi trường xây dựng cho các gói phần mềm thường được cấu hình cho việc xây dựng native trên hệ thống host. Một môi trường xây dựng chéo yêu cầu một cài đặt khác biệt và thường là khá phức tạp cho các công cụ, các tệp tiêu đề, thư viện và nhiều hơn nữa để hoạt động đúng cách.

Poky tạo ra các môi trường xây dựng chéo cho quá trình xây dựng BitBake của riêng mình. Thông qua lệnh devshell, nó cũng có thể cung cấp các môi trường xây dựng chéo trong một shell cho nhà phát triển. Lệnh

$bitbake < target > -cdevshell$

mở một cửa sổ terminal với một môi trường xây dựng chéo cho mục tiêu. Thiết lập của môi trường xây dựng chéo hoàn toàn phù hợp với môi trường mà Poky đang sử dụng cho các quá trình xây dựng của riêng mình. Trong shell đó, chúng ta có thể sử dụng các công cụ phát triển như bạn đã làm cho một xây dựng native trên máy host. Môi trường tham chiếu đến các trình biên dịch chéo chính xác cũng như bất kỳ tệp tiêu đề, thư viện và các tệp khác cần thiết để xây dựng gói phần mềm.

5 Debugging layers

Kiến trúc lớp của BitBake cung cấp một cách tổ chức công thức một cách tinh tế. Tuy nhiên, điều này cũng mang lại sự phức tạp, đặc biệt khi nhiều lớp cung cấp cùng một công thức và/hoặc sửa đổi cùng một công thức bằng các tệp nối.

Công cụ bitbake-layers cung cấp một số chức năng giúp phân tích và gỡ lỗi các lớp được sử dụng trong một môi trường xây dựng. Gọi

$bitbake - layershelp$

sẽ hiển thị một danh sách các lệnh có sẵn cho công cụ:

- help: Hiển thị danh sách các lệnh có sẵn hoặc cung cấp thông tin trợ giúp cho một lệnh cụ thể nếu được chỉ định.
- show-layers: Hiển thị danh sách các lớp được sử dụng bởi môi trường xây dựng cùng với đường dẫn và ưu tiên của chúng.
- show-recipes: Hiển thị danh sách các công thức theo thứ tự bảng chữ cái bao gồm lớp cung cấp chúng.
- show-overlayed: Hiển thị danh sách các công thức lớp được nối chồng. Một công thức được nối chồng nếu một công thức khác có cùng tên tồn tại trong một lớp khác.

- show-appends: Hiển thị danh sách các công thức với các tệp nối chúng. Các tệp nối được hiển thị theo thứ tự chúng được áp dụng.
- show-cross-depends: Hiển thị danh sách tất cả các công thức phụ thuộc vào metadata trong các lớp khác.
- flatten <directory>: Giải phẫu hệ thống lớp bằng cách giải quyết nối chồng công thức và ghi đầu ra vào một thư mục đơn lớp được cung cấp bởi tham số thư mục. Có một số quy tắc áp dụng:
  - Nếu các lớp chứa các công thức nối chồng, công thức từ lớp có ưu tiên cao nhất được sử dụng. Nếu các lớp có cùng ưu tiên, thứ tự của các lớp trong biến BBLAYERS của tệp conf/bblayers.conf của môi trường xây dựng quy định công thức nào được sử dụng.
  - Nếu nhiều lớp chứa các tệp không phải công thức (như hình ảnh, bản vá, hoặc tương tự) có cùng tên trong cùng một thư mục con, chúng sẽ được ghi đè bởi tệp từ lớp có ưu tiên cao nhất.
  - Tệp conf/layer.conf từ lớp được liệt kê đầu tiên trong biến BBLAYERS trong tệp conf/bblayers.conf của môi trường xây dựng được sử dụng.
  - Nội dung của các tệp nối đơn giản được thêm vào công thức tương ứng theo ưu tiên của lớp hoặc thứ tự của chúng trong biến BBLAYERS nếu các lớp nối cùng một công thức có cùng ưu tiên.

V Summary

Các công cụ của OpenEmbedded giúp rất hữu ích cho việc sửa lỗi trong quá trình build. Tìm nguyên nhân gốc rễ của vấn đề là bước đầu tiên quan trọng. Sau đó, việc tìm ra giải pháp có thể khó khăn hơn. Tuy nhiên, sử dụng log files, in ra metadata, và công cụ như bitbake-layers có thể giúp trong việc phân tích và gỡ lỗi. Chúng cung cấp cái nhìn tổng quan và các phương tiện cụ thể để đối phó với các vấn đề phức tạp của quá trình xây dựng.