OpenEmbedded in the Real World

Scott Murray
Senior Staff Software Engineer

Konsulko Group
http://konsulko.com
scott.murray@konsulko.com

Who am I?

- Long time Linux user (over 20 years)
- Have done Linux software development for over 15 years
- Have been doing embedded Linux development on and off since the start of my professional career
- Have previously done some kernel driver development, wrote and maintain the Linux CompactPCI hotswap framework
- Have been using OpenEmbedded / Yocto Project for close to three years on several projects ranging from small sensor style devices to rack mount network equipment.
- Currently working for Konsulko Group providing professional services
- Long time conference attendee, first time presenter

Caveats

- While I consider myself experienced with using OpenEmbedded, I’ve still got lots to learn!
- My apologies in advance if some of this seem obvious.
- I’m going to gloss over some details, as I’d like to get to the issues I feel you might not be aware of.
- Any recommendations I make are based on what has worked for projects I’ve worked on, and on discussions with coworkers and various community members.

Agenda

- Quick recap of OpenEmbedded & The Yocto Project
- Starting a project with OpenEmbedded
- Builds
- Packaging and Upgrade
- Workflow
- Security
- Support

scott.murray@konsulko.com

OpenEmbedded & The Yocto Project in 1 Slide!

- OpenEmbedded (OE) is a build system and associated metadata to build embedded Linux distributions.
- The Yocto Project is a collaboration project that was founded in 2010 to aid in the creation of custom Linux based systems for embedded products. It is a collaboration of many hardware and software vendors, and uses OpenEmbedded as its core technology. A reference distribution called “poky” (pock-EE) built with OE is provided by the Yocto Project to serve as a starting point for embedded developers.

An aside about naming

- People do and will get the naming wrong
- A typical example is referring to “Yocto Linux”
- My recommendation is to pick your battles...

scott.murray@konsulko.com

Konsulko Group

Starting a Project

- No matter the target, you’ll need to start by setting up your OpenEmbedded base layers, and then add any required BSP layers or layers for specific functionality
- You can do this by either:
  - Piecing together oe-core and bitbake repositories
  - Using the all-in-one Yocto Project poky repository
- Then add additional layers on top
  - Usually starting with your own layer to customize distribution settings, tweak package recipes, etc.
- Lastly customize the target image

Layer issues

- If you weren’t planning on using anything from poky, it seems reasonable to just use bitbake and oe-core by themselves…
- This can work, but many BSP layers rely on the linux-yocto kernel recipe from poky, making this path more difficult
- It is slight heresy to some in the OE community, but I lean towards using poky.git, as it’s one or two repos less to clone and track
- Branches can be an issue with non-core layers!
  - They may not have branches for different releases of OE (or at all)
  - You may have to mix and match, which can require experimentation and tweaking recipes

Layer issues continued

- There are a wide variety of layers available
- A good resource is the OpenEmbedded Metadata Index , which allows searching for layers and recipes
- Be aware that layer quality varies widely for layers not maintained by OE, the Yocto Project, or a vendor.
- As an example, there are instances of multiple layers existing to support certain SoCs, each supporting different sets of SBCs based on the SoC.

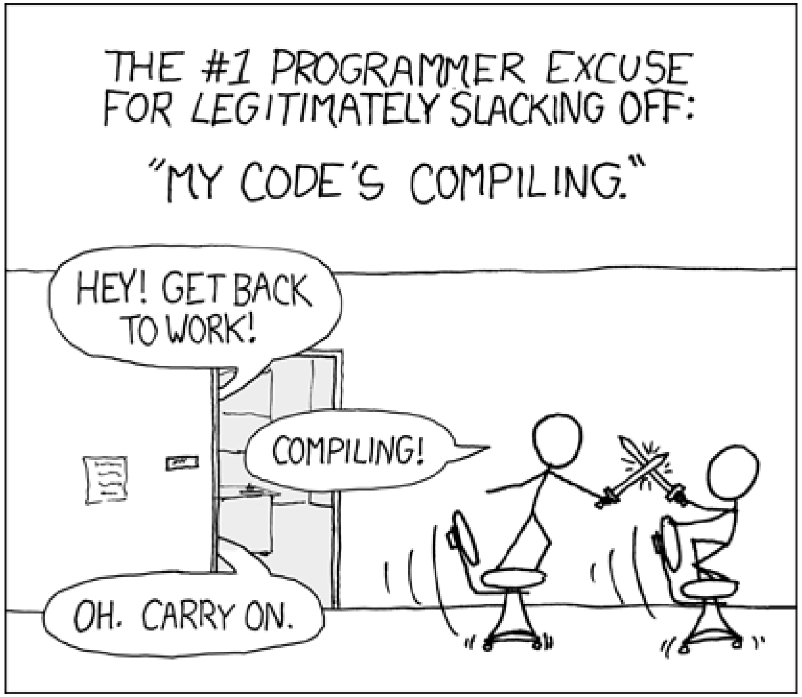
Too much of a good thing

- Layers may also provide more than you want
- Some layers bbappend quite a few recipes with small tweaks
- If you want a layer for one or a small number of recipes, these tweaks may be a nuisance
- This can be handled in a couple of ways
  - By using the BBMASK variable to mask out the undesired recipes
  - By copying the desired recipes to your own layer
  - Neither of these are necessarily ideal

Distribution and Image Customization

- I would recommend looking at the poky distribution configuration files to start
- It is pretty straightforward to copy the poky configuration to your own layer, rename it, and start tweaking it for your own purposes.
- If you are targetting small devices, the poky-tiny configuration trims some things out, and is a good starting point.
- Next, investigate the image configuration files for an image such as “core-image-minimal”
- Create your own image configuration based off of something close to what you want.
- For small devices, you will want to investigate some of the base packagegroups to see if you really need them or not

Build



http://xkcd.com/303/ (CC BY-NC 2.5)

Build Infrastructure

- Since OE bootstraps itself, clean builds are slower
- Throwing more hardware at it helps up to a point
  - See recommendations in latest Yocto Project documentation
  - Anecdotally, I've found that running from SSD helps significantly on machines that do not have a lot of RAM for caching
- Some software applications are painful to build
  - Java, Chromium
- Try to minimize building from scratch

Adding the Secret Sauce

- There may be a few projects where only existing software is required, but you likely have in-house developed software for your product
- For small embedded devices (IoT nodes, etc.), you may have one or two in-house applications
- For larger systems, the bulk of the software on the device may be in-house applications
- How do we combine this software with our OE system images?

scott.murray@konsulko.com

Building In-house Software

- For small targets, or for projects where you’ve started from scratch, you may be thinking of building everything via bitbake recipes
- This can work
- However, in my experience, it doesn’t scale well when:
  - You have a well-established existing build system for a lot of in-house software
  - You have a lot of packages in your image

scott.murray@konsulko.com

Large Project Issues

- If there is an existing build process, potentially already based on another Linux distribution, inertia is likely to work against significant change
- If there is a lot of software:
  - Building it all may not be developer friendly, as it has a good chance of being slower than existing workflow for code-build-test cycles
  - Writing recipes for all the separate components to split up the build could be a substantial amount of work

A Typical Large Project Model

- A common model is to split the OE and in-house software builds
- Usually the artifacts from the OE build serve as input for the in-house software build, which glues everything together.
- This does not have the elegance of a single build, but it has been the case in my experience that development of the two proceeds at different rates anyway
- It also saves in-house application developers from potentially having significant waits if a change on the OE side triggers rebuilding of a lot of packages
- In a continuous integration system, it is straightforward to chain the two builds together

Don’t Panic!

- Intermittently you may get a somewhat cryptic build failure
- Failures you might see include:
  - Changing package contents / splitting a package can confuse RPM, and sysroot population will fail
  - Sometimes changing a variable seems to not be detected
- If it’s a single recipe that’s failing, start off by trying to clean its state with “bitbake -c cleansstate”, then trying to build again
- If sysroot or rootfs population fails and the reason is not obvious, a brute force next step is to remove the “tmp” directory and have it be recreated from the state cache.

SDKs

- If you do build in-house software, it’s likely that you are building a SDK for it using OE.
- If you support a large product, it has been my experience that you will be updating the SDK somewhat regularly for internal users during a development cycle.
- It is common to install the SDK(s) on a NFS share to avoid having developers doing it themselves, and to sometimes to allow control over what tools are used.
- A drawback of this is that installing to NFS can be quite time-consuming. Run the install on the NFS server if possible!

SDKs in motion

- It is important to remember that once a SDK has been installed, it will not work correctly if it is moved afterwards
- If your configuration management process happens to include storing the toolchain(s) in version control, this will likely be a problem
- This can be hacked around by tinkering with the SDK’s relocate scripts, but if at all possible I would recommend changing your process to avoid the issue altogether

scott.murray@konsulko.com

Packaging and Upgrade

- For a lot of small systems the common packaging solution is a single package of some sort containing kernel, root filesystem, etc.
- Pretty well understood, and tools such as swupdate exist to implement this model
- What about using the deb or rpm package management features of OE for piecemeal upgrades?

scott.murray@konsulko.com

Package Management

- There are some issues with implementing a piecemeal upgrade scheme
- Package based upgrades across major OE releases can sometimes be problematic due to package renames or splits
- Packager manager support for pulling package upgrades over the network takes some work
- For rpm, smartpm is not widely used and not well known. There are no recipes for yum, dnf, or zypper.
- For deb, apt is part of oe-core.
- Setting up package repositories is covered in this Intel whitepaper

Workflow

- Your build-update-test cycle for your target
- For example, working on your OE configuration, tweaking a library or application recipe to develop a patch
- devtool is a relatively recent tool to simplify such tasks
  - I must admit I've been a bit of a Luddite so far, and still use a simple workflow iterating with bitbake
- I have heard reports of people feeling more productive when they just temporarily install development packages on the target so they can develop there

Security

- The OE team’s responsiveness on patching new security issues is good
  - However, tracking the application of patches for CVEs does require following the oe-core and oe-devel mailing lists and potentially looking in git
- Note that, since only the last 3 releases receive updates, you will be on your own after 1.5-2 years if you stay with a particular release
- For most products, this will not be sufficient

What are the options?

- Plan to roll out regular updates that track OE releases, to stay within the support window.
- Pay a vendor such as Wind River or Mentor Graphics for longterm support
- Do it yourself
  - Time-consuming to do a good job, especially if your product contains a lot of software packages.
- The recent meta-debian project attempts to solve the problem by fusing OE and Debian
  - Combines OE cross-compilation with Debian package patches
  - Allows tracking 5 years of security fixes from Debian stable
- There are some downsides
  - Requires a new recipe be written for each package
  - Community is currently small

Support

- Most immediate sources of support are the #oe and #yocto IRC channels on the freenode network
  - Need to be patient, especially outside of working hours
  - If you have not used IRC before, read up on etiquette
  - “Don’t ask to ask”
- oe-core, oe-devel, and yocto mailing lists
  - If sending recipe patches, there is a style guide in the OE wiki
- For documentation, bookmarking the "mega-manual" is useful, as it can be easily searched for terms.

Questions?

scott.murray@konsulko.com