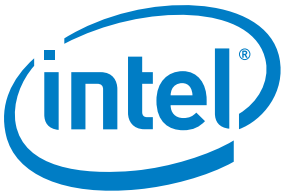


Customizing a Linux OS for Thin Clients using Intel BSPs and the Yocto Project

White Paper

November 2016

Document Number: 335100-002

Contents

Revision History

Date	Revision	Description
November 2016	001	Minor changes
October 2016	001	First public release

1.0 Introduction

Thin clients are:

- Network-based computing devices that lack local storage and are used almost exclusively in business and educational environments[1].
- Becoming a global megatrend with double digit growth predicted as the market moves toward cloud services and green IT.
- Typically reliant on a thin or minute operating system that is purposely built to offer basic computing needs with a good user experience and system interaction from a proprietary, centralized enterprise software that is proprietary.

During a user session, a thin client loads the enterprise application as virtual machine (VM) runtime. The load time depends on the type of network (LAN or Wi-Fi*) and the VM payload. Upon log off, the VM saves a work snapshot and powers down gracefully before committing changes to a central server.

These features and elements are vital in a thin client system to serve in an enterprise and education sectors:

Requirement	Hardware	Purpose
Connectivity	• Ethernet Hub (primary) • Wi-Fi • BLE*	Establish connection to back end systems and load proprietary software into virtual environments or containers.
Graphics and media	• Integrated graphics engine • Integrated audio codec	Deliver an exceptional user experience with graphics and media acceleration and audio codecs.
Internal storage	• Embedded multimedia card (eMMC) • Solid state drive (SSD)	Offer limited non-volatile storage to host thin client OS and paging whenever RAM is fully utilized.
Optional peripheral	• Webcam • Headphone and microphone	Support utilities to enhance productivity at call centers or banking environments.
System memory	• Random access memory (RAM)	High-speed volatile storage to page software, thin client OS, or virtual payload with runtime services.
User input methods	• Keyboard and mouse • Touch panel	Standard user input for most enterprise and education sectors.

Typical thin client feature requirements include:

Feature	Description
Security	• Hardware accelerated encryption and decryption • Secure Security Key, Digital Certificate, and Policy Manifests Storage • Secure Boot or Measure and Verified Boot
Virtualization	• Logical isolation to contain proprietary and secure runtime for enterprise • Dedicating CPU, memory, IO and storage resources on demand

1.1 Terminology and Acronyms

Term	Description
BSP[3]	Board Support Package
eMMC	Embedded Multimedia Card
Grub	Grand Unified Bootloader
ISV	Independent Software Vendor
LAN	Local Area Network (Ethernet)
RAM	Random Access Memory
SI	System Integrator
SoC	System on Chip
SSD	Solid State Drive
VDI	Virtual Device Interface
VM	Virtual Machine

1.2 References

1. IDC : Worldwide Thin Client 2014-2017 Forecast and Analysis
2. Toolchain : http://elinux.org/Toolchains
3. About BSP : http://www.yoctoproject.org/docs/2.1/bsp-guide/bsp-guide.html
4. About Yocto : https://www.yoctoproject.org/about
5. BSP Downloads : https://www.yoctoproject.org/downloads/bsps
6. SPICE : https://www.spice-space.org/spice-user-manual.html
7. FreeRDP : https://github.com/FreeRDP/FreeRDP
8. Remmina : https://github.com/FreeRDP/Remmina

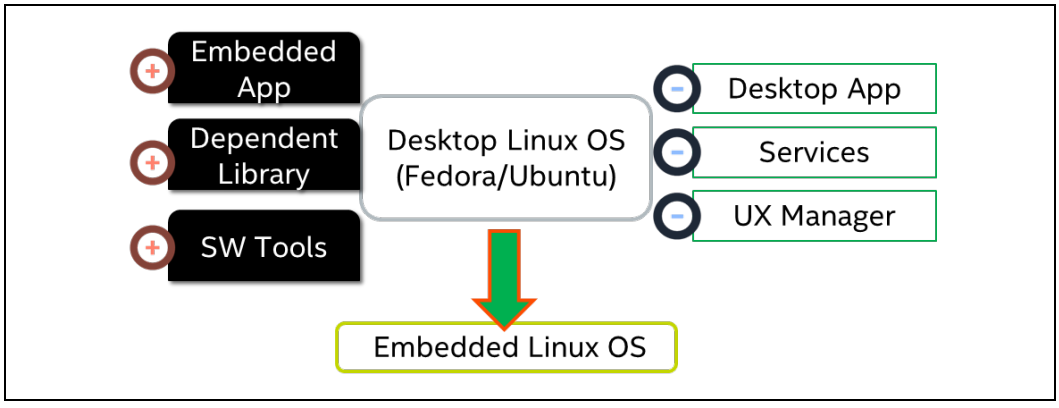
2.0 Business Challenge

Thin clients systems are designed with specific requirements and constrained storage resources. They require efficient runtime, without lag over long hours of operation, and support virtualization, or container technology that is familiar to the ecosystem with security.

These requirements require OS customization, but customizing or creating a thin OS for the masses of thin client system can be time-consuming and can tax engineering resources if the right methodology, tools and ingredients are not applied.

As shown in the diagram below, traditionally customizing a desktop operating system, such as Ubuntu or Fedora, is done by removing unnecessary elements and adding the needed elements. The needed elements might include an application, libraries and software tools that satisfy the hardware needs like internal storage and system memory. Therefore, challenges faces by developers include:

- Searching, testing, and validating system drivers for graphics, I/O devices, and touch panels for the preferred thin client hardware.
- Setting up the toolchain[2]. This means compiling source code with the target toolchain, where the target toolchain is a collection of tools used for development. This task is vital to ensure the customized thin OS and applications can execute on the new architecture or hardware as they did before.
- Architecture Migration. Without a BSP[3], porting legacy and new drivers are added overhead. This overhead also applies to libraries and related runtime dependencies.

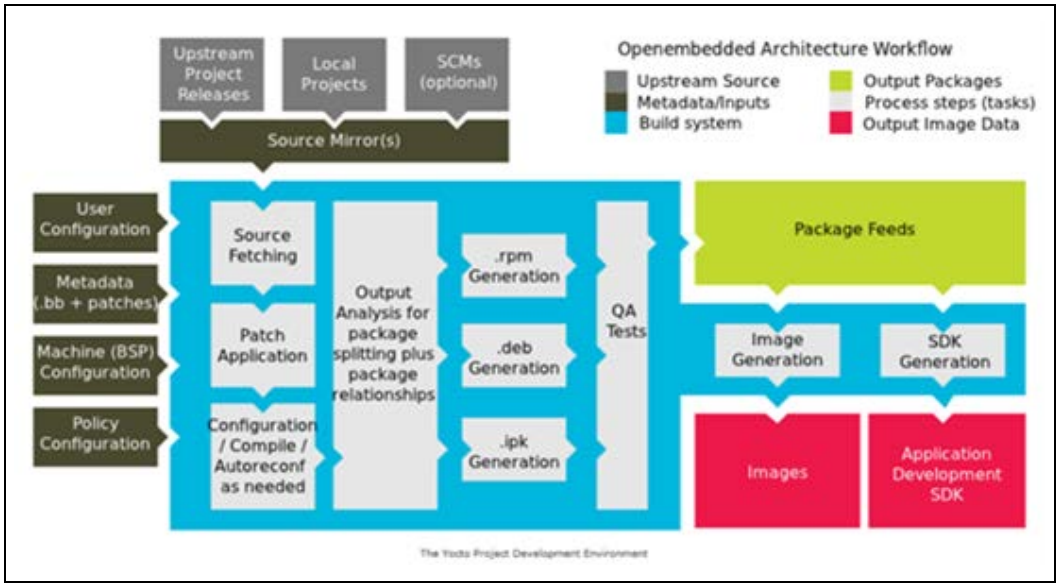


3.0 Customizing a Thin Client OS with the Yocto Project

The remainder of this document provides basic customization information and recommendations to customize an embedded Linux OS with the Yocto Project.

3.1 The Yocto Project

The Yocto Project is an open source collaboration that provides templates, tools, and methods to help you create custom Linux operating systems. The example in this document customizes an operating system for a thin client. Collaboration is necessary between hardware manufacturers, open-source operating systems vendors, and electronics companies to bring order to the chaos of embedded Linux development[4].



A Yocto Project development environment establishes the framework that provides a logical flow to create a Linux OS and software development kits for distribution among working partners.

3.2 Getting Started

An embedded OS developer needs key tools, ingredients, and documentation. See the following list of requirements.

Requirement	Description
Development host	64-bit Linux Development System with a minimum 500 GB storage size. Use this to build the target Linux image. Ubuntu 14.04 LTS is preferred.
Tools	See the toolchain[2]
BSP[3]	Target platform architecture and system definition. Every Linux image developed will exclusively run on the BSP chosen by the developer.
Application recipes	Added components to bring character and purpose to the finished product.
Build script	Controls the build process, sequencing the package to be built intelligently to build a
Output	Customized embedded Linux OS

3.3 Setting up the Environment

See http://www.yoctoproject.org/docs/2.1/ref-manual/ref-manual.html#required-packages-for-the-host-development-system

3.4 Choosing a BSP and Software Recipes

When choosing a BSP, consider the target platform and processor / SoC architecture on which the final operating system will run. A BSP is collection of information that defines the support on a one or more hardware devices or hardware platform[3].

This guide uses the Cherry Hill BSP that can be downloaded from https://www.yoctoproject.org/downloads/bsps/krogoth21/cherry-hill. Since this is targeted for thin client use, these three recipes can be used:

- Virtualization with meta-virtualization https://layers.openembedded.org/layerindex/branch/master/layer/meta-virtualization/
- Chrome browser with meta-browser https://github.com/OSSystems/meta-browser.git
- Remote desktop client with meta-remote desktop (https://github.com/kanapathym/meta-remotedesktop)

3.5 Building the Target Image

3.5.1 Downloading Packages

1. Start a command line terminal and create a working directory.

```
$mkdir yocto_build
$cd yocto_build
```

2. Download the Yocto Development packages in the form of source code and put it into the `yocto_build` directory.

```
git clone git://git.yoctoproject.org/poky.git
  - b <branch-name>.
```

   The <branch-name> might vary. It is based on BSP availability and kernel version. For Cherry Hill and the Intel® Pentium®/Celeron® Processor N3000 family or Intel® Atom™ x5-E8000 processor family, use the krogoth branch.

3. Set the total package both for the Yocto and the Intel BSP layers:

```
$git clone git://git.yoctoproject.org/poky.git -b krogoth &&
  cd poky && git://git.yoctoproject.org/meta-intel -b krogoth
```

4. Download the Cherry Hill BSP into meta-intel layer and extract the compressed package

```
$cd meta-intel
$wget http://downloads.yoctoproject.org/releases/yocto/yocto-
  2.1/machines/intel-corei7-64/intel-corei7-64-5.0-krogoth-
  2.1.tar.bz2
$tar xvjf intel-corei7-64-5.0-krogoth-2.1.tar.bz2
```

3.5.2 Downloading Additional Layers, Recipes, and a Setup Environment for the Build

1. To enable thin client use-cases, download two layers. The first provides a browser console to load web content. The second provides container or virtualization capability.

```
$cd poky
$git clone https://github.com/OSSystems/meta-browser.git
$git clone git://git.yoctoproject.org/meta-virtualization
$git clone https://github.com/kanapathym/meta-remotedesktop
```

2. These two layers require open-ended layers for compilation and runtime. To meet this requirement, download:

```
$git clone git://git.openembedded.org/meta-openembedded
$git clone git://git.openembedded.org/openembedded-core
```

3. Change to the `yocto_build` directory.
4. Setup the toolchain for cross-compilation and build environment:

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

3.5.3 Editing Configuration Files and Building Image

The configuration you add in this section informs the build environment of the BSP and applications to install, and the license binding policy applications to install and the license binding policy.

1. Add these lines to the file `build/conf/bblayers.conf`

```
meta-intel                  (Intel® BSP)
meta-browser                (Contains Firefox* web browser application)
meta-remotedesktop          (Contains spice, remmina for remote desktop and
                            VDI[6] sessions)
meta-virtualization         (Containers and virtualization like docker, KVM and
                            Xen)
meta-openembedded/meta-oe       (Support layers from openembedded)
meta-openembedded/meta-gnome    (Support layers from openembedded)
meta-openembedded/meta-networking (Support layers from openembedded)
meta-openembedded/meta-python       (Support layers from openembedded)
meta-openembedded/meta-filesystems (Support layers from
                                    openembedded)

# POKY_BBLAYERS_CONF_VERSION is increased each time
build/conf/bblayers.conf
# changes incompatibly
POKY_BBLAYERS_CONF_VERSION = "2"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
  /path/poky/meta \
  /path/poky/meta-poky \
  /path/poky/meta-yocto-bsp \
  /path/poky/meta-intel \
  /path/poky/meta-browser \
  /path/poky/meta-openembedded/meta-oe \
  /path/poky/meta-openembedded/meta-gnome \
  /path/poky/meta-virtualization \
  /path/poky/meta-remotedesktop \
  /path/poky/meta-openembedded/meta-networking \
  /path/poky/meta-openembedded/meta-python \
  /path/poky/meta-openembedded/meta-filesystems \
  "
```

2. Run echo to add configuration lines to `build/conf/local.conf`. The `IMAGE_INSTALL_append` configuration allows you to explicitly set your choice of application or service as part of the final image.

```
$ echo 'MACHINE = "intel-corei7-64"
LICENSE_FLAGS_WHITELIST = "commercial"
IMAGE_INSTALL_append =+ ‘‘gstreamer1.0-plugins-ugly
  gstreamer1.0-libav docker containerd docker-registry firefox
  freerdp xterm spice remmina"' >> build/conf/local.conf
```

3. With the configurations has been correctly set, execute `bitbake` to start building the Linux image:

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

`bitbake` takes 2 to 4 hours. Using SSD storage with a large amount of system memory, and a multicore processor may reduce compilation time.

3.5.4 Booting the Final Image

After compiling the Linux image, use these steps to create a bootable USB Linux image.

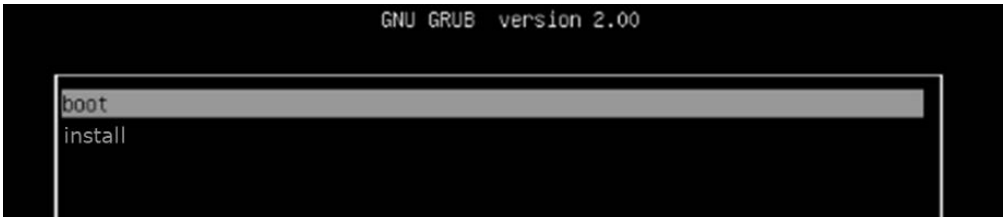
1. Plug in a USB flash drive that has at least 1 GB of storage capacity.
2. Identify the USB flash drive:

   ```
   fdisk -l
   ```

   This command lists each attached storage device and provides its storage capacity. From there, it will be easy to identify which device would become the target. The example in the next steps uses target device `/dev/sdc`.
3. Write the image to a disk. **Use caution**: Writing the image file into a disk is unrecoverable. Make sure you identify the right target disk before running the command:

   ```
   $dd if=tmp/deploy/images/intel-corei7-64/core-image-sato-
     intel-corei7-64.hddimg of=/dev/sdc && sync
   ```

4. Plug the USB flash drive into a Cherry Hill reference platform based on an Intel® Pentium®/Celeron® processor N3000 family or Intel® Atom™ x5-E8000 processor.
5. Power on the reference platform.
6. Update the system BIOS to boot from the USB flash drive. The grub screen displays to allow you to select `boot` or `install` the OS.

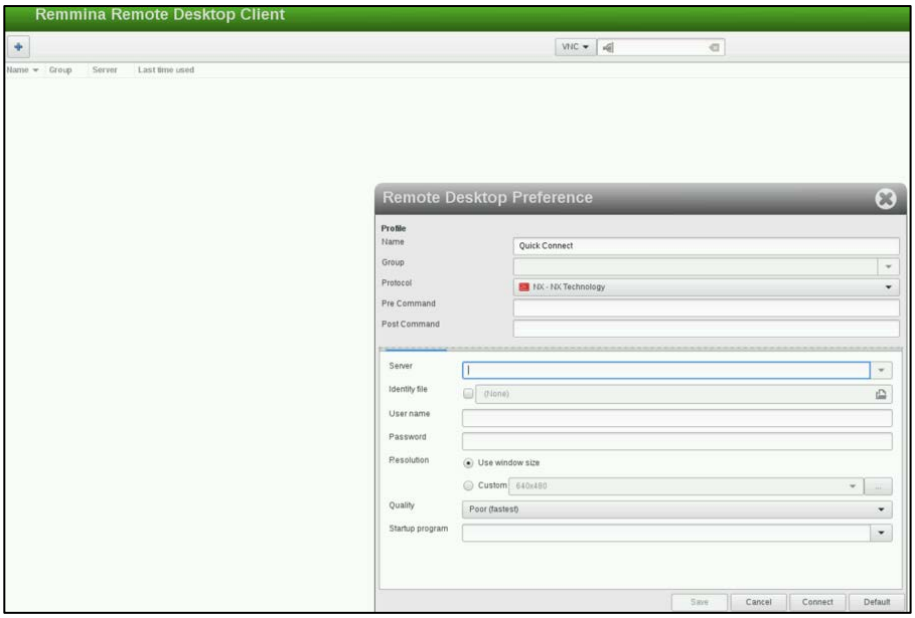


The screen below displays when the Thin Client OS image successfully boots.

3.6 Connecting to Virtual Desktop Infrastructure or Remote System

You can connect to a virtual desktop infrastructure (VDI) or to a remote system with Remmina Remote Desktop Client[8].

Remmina supports multiple network protocols, such as RDP, VNC, SPICE[6], NX, XDMCP and SSH. After completing the build steps, the Remmina icon displays on the matchbox compositor desktop.


Remmina Remote Desktop Client
VNC
Name
Group
Server
Last time used
Remote Desktop Preference
Profile
Name
Quick Connect
Group
Protocol
NX - NX Technology
Pre Command
Post Command
Server
Identity file
(None)
User name
Password
Resolution
Use window size
Custom
640x480
Quality
Poor (fastest)
Startup program
Save
Cancel
Connect
Default

4.0 Conclusion

Customizing a Yocto Project with a thin client OS provides reduced cost, effort, and time-to-market. An OS developer can scale a solution with wide performance and a feature range offered from Intel® Atom™ to Xeon™ microarchitecture with an optimized foot print.

For thin client uses-cases, an OS developer can use third-party or commercial VDI software components.