Linux in Mobile Devices

Miko Nieminen
Chief Technology Officer
miko.nieminen@nomovok.com
Some Terms

- **Linux**
  - Linux is just kernel of the operating system

- **Distribution**
  - Bundles of software which are used for distributing Linux-based systems
  - Some examples: Debian, Gentoo, Redhat, SuSE, Ubuntu

- **Open Source Software (OSS)**
  - Licensing model for free/open software

- **Software Development Kit (SDK)**
  - Set of tools and environments for developing software
  - Usually bundled package for some specific device or device family
Using Linux in Mobile Devices

**Product types**
- Internet Tablets
- Mobile phones
- GPS Navigators
- Netbooks
- etc.

**Existing software stacks**
- Maemo
  - For Nokia 770/N800/N810
  - Gnome Mobile based
- ACCESS ALP
  - Designed for mobile phones
  - Gnome Mobile based
- Google ANDROID
  - Basically just Java stack on top of Linux
- Intel's Moblin
  - Currently focus in netbooks, but not limited to
Hardware for Mobiles

- Most often ARM processors
  - ARM has multiple instruction set versions: ARMv4, ARMv5, ARMv6, ARMv7
  - ARM has also multi core versions
- Nokia's N800 has ARMv6 running 400 MHz with 128 MB of RAM
- More recent hardware may have ARMv6 or v7 running 500-700 MHz with 128MB or more RAM
- Later this year we will see products having 1GHz ARM processor with even much as 512MB to 1GB of RAM
Desktop vs. Mobile

- Modern mobile systems share much with desktop systems
- Common components
  - generic frameworks (for example multimedia)
  - Backends and services (for example email or calendar)
- Mobiles have limited input methods
  - no keyboard, no pointer devices (not counting touchscreen here)
  - phone factor keyboard or touchscreen
- Main differences are in display resolutions and UI design
- Mobile devices tend to be always on
  - active polling kills battery
  - memory leaks more problematic
- When writing good software it doesn't matter if software is written for desktop or mobile device
High Level Application Stack

Kernel & Drivers
- Low level components, hardware specific layer

Common Services and Libraries
- Low level libraries (including low level graphical components)
- Mostly components defined by LSB (Linux Standard Base)

User Interface Library
- Toolkits and libraries to ease implementation of GUI applications

Platform Services and Frameworks
- Product or product family specific middleware
Importance of Generic Architectures

- Generic architectures are needed so that mobiles can share components with desktop systems
- Modular structures
  - Plugin architectures
    - Runtime extensions
  - Compile time module selections
- Server-client architectures
  - Separating business logics to backends and daemons
  - Client just has UI logics
  - Separation for example by using IPC, for example D-Bus
- Pipes and filters
  - Building pipelines from smaller components where each component handles specific task, for example Gstreamer
Example Application Architecture

**UI Layer**
- As light as possible
- Defines application UI layout

**Callback Delegate**
- Data model event delegate
- UI interaction event delegate
  - Decision if event just causes UI state change or do we need to discuss with Engine Abstraction Layer

**Engine Abstraction Layer**
- Library and engine initialization and communication
Problematic areas

• Availability of low level drivers
  – Radios (modems, wlan, bluetooth, ...)
  – Hardware accelerators for graphics
  – DSPs (digital signal processors)
  – Used especially to improve multimedia performance
  – Power management components

• UI Design
  – Usability patents
  – Protecting own Look & Feel

• On-line Services
  – Communicating with services which can't open dataformats
  – For example map services
Licenses and legal aspects

- Understanding licenses
  - Rules about using software
  - Mixing licenses
- Incorrect use of licenses
  - May force closed source software to be opened
- License types:
  - Copyright: different EULAs, like MS EULA
  - Copyleft: GPL/LGPL
  - Copycenter: BSD licenses

- SW developer needs to understand more about licensing and legal issues than developer who works completely in proprietary world!
- Larger corporations try to push software under patent laws
Separating OSS and non-OSS parts

- Using dynamic libraries
  - Creating dynamic libraries which are connected through dynamic linking at runtime
  - not enough when mixing GPL and proprietary software
- Using IPC (Inter-process Communication)
  - Separating business critical code and OSS code into separate processes which communicate over IPC
    - Most of the time improves also architectural design
Software Development Kit

- Build tools
  - Compilers
  - Packaging tools
  - Other tools: make, autotools, ...

- IDE
  - Good SDK allows developer to use which they prefer
  - Emacs, Eclipse, Anjuta, Qt Creator, ...

- Documentation
  - Usually documentation is available on-line

- Test and analysis tools
  - Running
  - Debugging
  - Profiling
Development Process Demo

1) Most of the development happens in x86 hardware in virtualized environment

2) After applications is good enough for target testing, target binaries are build
   • Using native build environment which could be replaced with cross-compilation environment like most of the mobile SDK are using.

3) Making final testing and using software at actual target device
Conclusions

- Making software for mobile Linux devices is like any other software development under Linux
  - Requires some additional attention for avoiding active polling and memory leaks, though these are bad for any software
- There are lots of existing applications and libraries which to use so you don't have to write that much by our self
  - Most of the work is about integrating different capabilities
- Strong focus on UI design and customizable systems
  - Basically all devices need custom made UI
- Differs from S60 and other mobile platforms by the fact that desktop and mobile world differences are almost non-existing
  - You can't run S60 on your desktop
Future

- **Dynamic languages and runtimes**
  - Removes build phase of the development process
  - Same software works directly in different hardware architectures
  - UI designers create device UI directly and no engineering part is needed
- **Common distributions**
  - Most of the software is going to come from a common distribution and mobile device manufacturers stop making their own semi-closed systems
  - Common rules for DRM
  - Common APIs for hardware
- **Common development**
  - Distinction between software development for mobiles and desktop will become insignificant and focus will shift even more to UI level