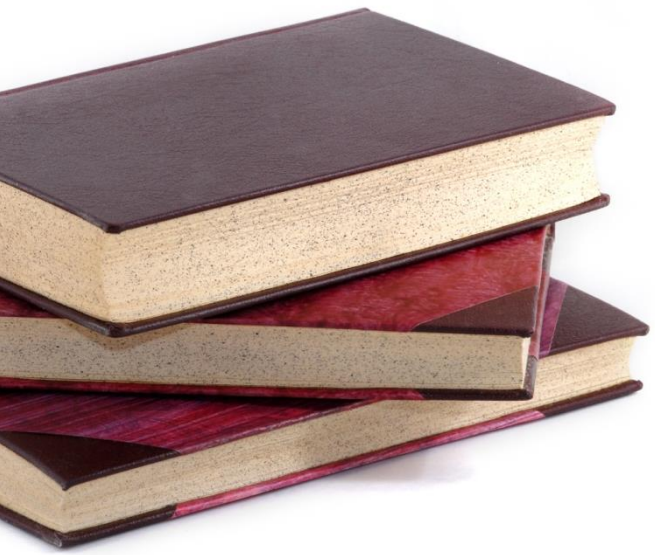
MPSIT

Lecture 4



4

ADT Eclipse Plug-ins

1 noiembrie 2016

Outline

- Environment setup
- Eclipse IDE
- QEMU Emulator
- Debugging
- Profiling and Tracing

Recap: Yocto Project development model

1. System development workflow
2. Application development workflow
3. Modify temporary source code
4. Image development
5. Using the devshell

Recap: Yocto Project getting started

- Get the source code
  - git clone https://git.yoctoproject.org/git/poky
  - cd poky
- Build the demo image
  - source oe-init-build-env
  - vim conf/local.conf
    - MACHINE=qemuarm
  - bitbake core-image-minimal
- Run the demo into an emulator
  - runqemu qemuarm

Application development Toolkit

- Application development = create an application for a target hardware.
- The target hardware runs a kernel image created using the OE build system.

- The Yocto Project provides:
  - Application Development Toolkit (ADT)
  - The possibility to use stand-alone cross-development toolchains
  - Optional Eclipse Yocto Plug-in to develop, deploy and test your application all from within Eclipse.

Remember ADT?

- Provides a standalone cross-compiler, debugger, tool-profilers, emulators and even development board interaction
- Platform independent
- What else do you remember from last course?

Environment setup

1. Using an ADT install script
   - Recommended method. Mostly because it is a completely automated process
2. Using the ADT tarball
   - Involves a tarball selection process and an automate setup process with the help of a script
   - The tarball can also be manually built with the help of Bitbake
   - Can have features limitation
3. Using a toolchain from the build directory
   - Takes advantage of the already available build directory
   - Cross-toolchain setup is really easy
   - Same limitation as the method described above

ADT installer script

- bitbake adt-installer
- tar –xjf adt_installer.tar.bz2
- vim adt_installer.conf
  - YOCTOADT_REPO
  - YOCTOADT_TARGET
  - YOCTOADT_QEMU
  - YOCTOADT_ROOTFS_<arch>
  - YOCTOADT_TARGET_SYSROOT_IMAGE_<arch>
  - YOCTOADT_TARGET_MACHINE_<arch>
  - YOCTOADT_TARGET_SYSROOT_LOC_<arch>
- ./adt_installer

ADT installer .conf example

```
# Your yocto distro repository, this should include IPKG based packages
and root filesystem files where the installation is based on

YOCTOADT_REPO="http://adtrepo.yoctoproject.org//1.7"
YOCTOADT_TARGETS="arm x86"
YOCTOADT_QEMU="Y"
YOCTOADT_NFS_UTIL="Y"

#YOCTOADT_BITBAKE="Y"
#YOCTOADT_METADATA="Y"

YOCTOADT_ROOTFS_arm="minimal sato-sdk"
YOCTOADT_TARGET_SYSROOT_IMAGE_arm="sato-sdk"
YOCTOADT_TARGET_MACHINE_arm="qemuarm"
YOCTOADT_TARGET_SYSROOT_LOC_arm="$HOME/test-yocto/$YOCTOADT_TARGET_
MACHINE_arm"

#Here's a template for setting up target arch of x86
YOCTOADT_ROOTFS_x86="sato-sdk"
YOCTOADT_TARGET_SYSROOT_IMAGE_x86="sato-sdk"
YOCTOADT_TARGET_MACHINE_x86="qemux86"
YOCTOADT_TARGET_SYSROOT_LOC_x86="$HOME/test-yocto/$YOCTOADT_TARGET_
MACHINE_x86"

#Here's some template of other arches, which you need to change the value
in ""
YOCTOADT_ROOTFS_x86_64="sato-sdk"
YOCTOADT_TARGET_SYSROOT_IMAGE_x86_64="sato-sdk"
YOCTOADT_TARGET_MACHINE_x86_64="qemux86-64"
YOCTOADT_TARGET_SYSROOT_LOC_x86_64="$HOME/test-yocto/$YOCTOADT_TARGET_
MACHINE_x86_64"

YOCTOADT_ROOTFS_ppc="sato-sdk"
YOCTOADT_TARGET_SYSROOT_IMAGE_ppc="sato-sdk"
YOCTOADT_TARGET_MACHINE_ppc="qemuppc"
YOCTOADT_TARGET_SYSROOT_LOC_ppc="$HOME/test-yocto/$YOCTOADT_TARGET_
MACHINE_ppc"

YOCTOADT_ROOTFS_mips="sato-sdk"
YOCTOADT_TARGET_SYSROOT_IMAGE_mips="sato-sdk"
YOCTOADT_TARGET_MACHINE_mips="qemumips"
YOCTOADT_TARGET_SYSROOT_LOC_mips="$HOME/test-yocto/$YOCTOADT_TARGET_
MACHINE_mips"
```

Other options

- runqemu-extract-sdk
- wget http://downloads.yoctoproject.org/releases/yocto/yocto-2.1/toolchain/x86_64/poky-glibc-x86_64-core-image-sato-armv7a-neon-toolchain-2.1.sh
- bitbake meta-toolchain
- bitbake –c populate-sdk <image-name>
- ./poky-glibc-x86_64-core-image-sato-armv7a-vfp-neon-toolchain-1.7.sh
- bitbake meta-ide-support

Eclipse IDE

- https://www.youtube.com/watch?v=3ZlOu-gLsh0
- Alternative solution for developers not keen on using vim and command line interaction
- Support for Luna SR2 (4.4.2) and Kepler (4.3.2): http://www.eclipse.org/downloads/
- tar -xzvf ~/Downloads/eclipse-cpp-luna-SR2-linux-gtk-x86_64.tar.gz
- Info also available here: http://www.yoctoproject.org/docs/2.1/mega-manual/mega-manual.html#setting-up-the-eclipse-ide

Qemu emulator

- Used as virtualization machine and emulator
- Useful for tests executions
- One of Yocto Project selling points
- Started in Eclipse using **External tools** option from **Run** menu

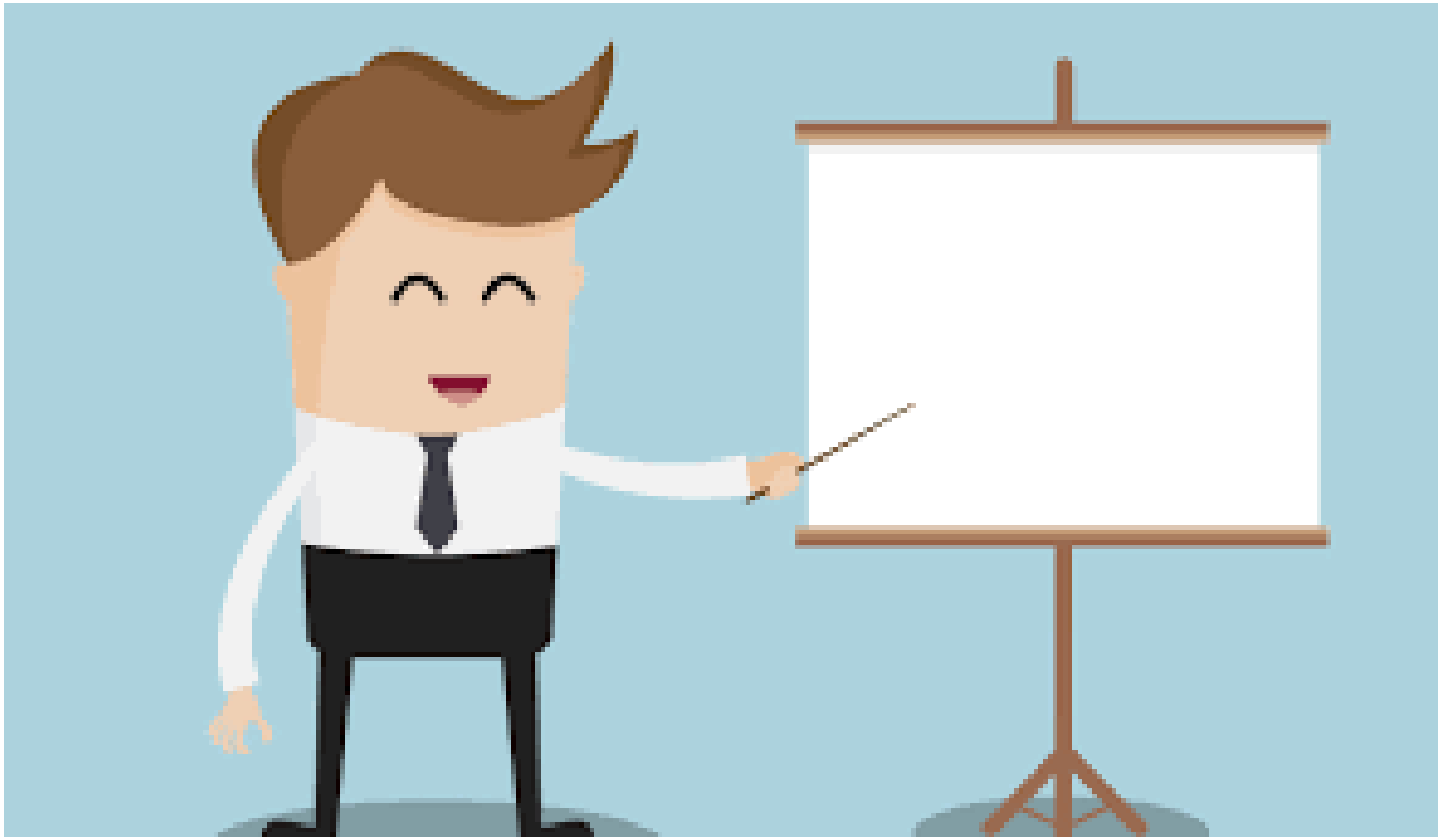
Debugging

- Started in Eclipse using **Remote Application** from **Run** menu
- Name <project-name>_gdb_- <suffix> syntax
- For shared libraries debugging extra steps are required:
  - Select **Add | Path Mapping** option from the **Source** tab to make available a path mapping
  - Select **Load shared libraries symbols automatically** from the **Debug/Shared Library** tab and indicate the path of the shared libraries.
  - In the **Arguments** tab pass libraries arguments if required during execution

Profiling and tracing

- Yocto Tools: oprofile, perf, LTTng, PowerTop, LatencyTop, SystemTap, KGDB
  - LTTng: offers the possibility of tracing a target session and analyzing the results.
  - LatencyTop: identify the latencies available within the kernel and also their root cause.
  - PowerTop: used to measure the consumption of electrical power.
  - SystemTap: enables the use of scripts to get results from a running Linux.
- Can you please help with the rest of them?

Student presentation



Project Documentation

- Define functionalities and features
- Define used technologies
- What are the use cases you have in mind for this project
- Is it Yocto Project integrated or not?
- Document file should be ready by lecture 5.
- Documentation is part of the project score.

Students proposed project

- Optimize boot time & size for a Yocto Project Linux distribution
  - Reduce busybox functionalities support.
  - Reduce resulting rootfs size.
  - Minimize Linux kernel configuration.
  - Optimize bootloader if possible.
  - Resulting output should be able to run a graphical application similar to glxgears.
  - Boot time required under 10 sec.
  - If done in teams of 2, boot time under 7 sec.
  - Use case: boot the target/qemu and check the time at which glxgears appears.

The End