Advanced Embedded Linux®

Version 1.0

Unable to handle kernel paging request at virtual address 4d1b65e8
Unable to handle kernel paging request at virtual address 4d1b65e8
pgd = c0280000
pgd = c0280000
<1>[4d1b65e8] *pgd=00000000[4d1b65e8] *pgd=
Internal error: Oops: f5 [#1]
Internal error: Oops: f5 [#1]
Modules linked in:Modules linked in: hx4700_udc
CPU: 0
CPU: 0
PC is at set_pxa_fb_info+0x2c/0x44
PC is at set_pxa_fb_info+0x2c/0x44
LR is at hx4700_udc_init+0x1c/0x38 [hx4700_ud
LR is at hx4700_udc_init+0x1c/0x38 [hx4700_ud
pc : [<c00116c8>] lr : [<bf00901c>] Not taint
sp : c076df78 ip : 60000093 fp : c076df84
pc : [<c00116c8>] lr : [<bf00901c>] Not taint

Codefidence
a name you can trust™

מכללת הי-טק

Based on material by:
Free Electrons

Rights to copy

creative commons
COMMONS DEED

Attribution – ShareAlike 2.0

You are free

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

U the following conditions

Attribution. You must give the original author credit.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

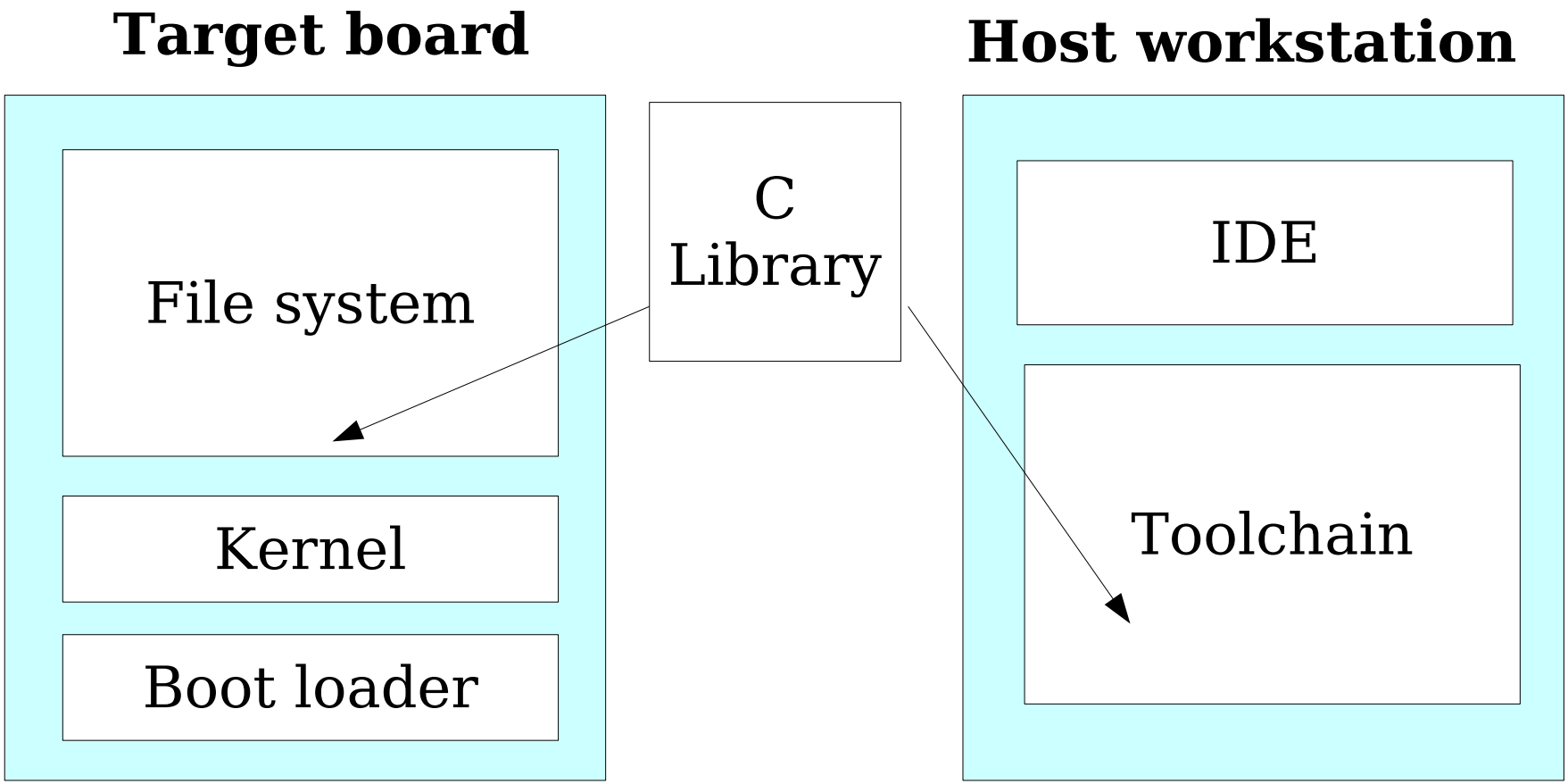
License text: http://creativecommons.org/licenses/by-sa/2.0/legalcode

This kit contains work by the following authors:

© Copyright 2004-2006
Michael Opdenacker
michael@free-electrons.com
http://www.free-electrons.com

© Copyright 2004 – 2007
Codefidence ltd.
info@codefidence.com
http:/www.codefidence.com

What makes an embedded Linux system



C library for the target device

C library options

glibc

http://www.gnu.org/software/libc/

- License: LGPL
- C library from the GNU project
- Designed for performance, standards compliance and portability
- Found on all GNU / Linux host systems
- Quite big for small embedded systems: about 1.7MB on Familiar Linux iPAQs (`libc`: 1.2 MB, `libm`: 500 KB)

uClibc

http://www.uclibc.org/ for CodePoet Consulting

- License: LGPL
- Lightweight C library for small embedded systems, with most features though.
- The whole Debian Woody was ported to it...
  You can assume it satisfied most needs!
- Example size (`arm`): approx. 400KB (`libuClibc`: 300 KB, `libm`: 55KB)

Honey, I shrunk the programs!

C program	*Compiled with shared libraries*		*Compiled statically*	
	glibc	*uClibc*	*glibc*	*uClibc*
Plain “hello world”	4.6 K	4.4 K	475 K	25 K
Busybox	245 K	231 K	843 K	311 K

newlib (1)

http://sources.redhat.com/newlib/

Minimal C library for very small embedded systems

- Lets you remove floating point support wherever you don't need it. Also provides an integer only `iprintf()` function. Much smaller!
- Provides single precision math library functions. Much faster than the standard IEEE compliant ones.

newlib (2)

- Extremely portable:
  - Almost everything implemented in C
  - Already supported on many processors
  - Even supports systems with no operating system! Just requires the implementation of at most 17 system-specific stubs, or less, for example if your programs don't use `fork()` and `system()`.

See http://www.embedded.com/story/OEG20011220S0058 for a very good overview of newlib

C library options - Summary

- GNU C library (glibc)

  Designed for performance compliance with standards.

  Best for servers and desktops.

- uClibc

  Highly compatible, but implemented for small embedded systems with little storage space and RAM

- newlib

  Fully customizable C library for tiny and standalone systems.

GNU / Linux workstation

Cross-compiling toolchains

Standalone toolchain build

Building a cross-compiling toolchain by yourself is a difficult and painful task! Can take days or weeks!

- Lots of details to learn. Several components to build (building gcc twice: once for gcc + once for compilers that need the C library).
- Lots of decisions to make (such as glibc version and configuration for your platform)
- Need kernel headers and glibc sources
- Need to be familiar with current gcc issues and patches on your platform
- Useful to be familiar with building and configuring tools
- http://www.aleph1.co.uk/armlinux/docs/toolchain/toolchHOWTO.pdf can show you how fun it can be!

Get a precompiled toolchain

Can get one from several locations... Just need to know where!

Caution

- Toolchains are not always relocatable! old versions of gcc and related tools are known to have hard coded paths.
- Make sure the toolchain you pick suits your needs:
  CPU, endianism, C library and compiler versions...

uClibc toolchains

Free Electrons uClibc toolchains

http://free-electrons.com/community/tools/uclibc

- Run on i386 GNU/Linux
- Supported platforms

  `arm`, `armeb`, `i386`, `m68k`, `ppc`, `mips`, `mipsel`, `sh`
- Quickly updated at each new uClibc release!

Platform specific toolchains

ARM

- Code Sourcery (glibc only, used by many):
  http://www.codesourcery.com/gnu_toolchains/arm/
  Also available for Solaris and Windows workstations.
- ftp://ftp.handhelds.org/projects/toolchain/ (glibc only)

MIPS

- http://www.linux-mips.org/wiki/Toolchains (useful links)

Toolchain building utilities

Buildroot: http://buildroot.uclibc.org/

- Dedicated Makefile to build uClibc based toolchains and even entire root filesystems.
- Downloads sources and applies patches.

Crosstool: http://www.kegel.com/crosstool/

- Dedicated script to build glibc based toolchains
  Doesn't support uClibc yet.
- Downloads sources and applies patches.

Crosstool

- Crosstool is a set of scripts to build and test several versions of gcc and glibc for most architectures supported by glibc.
- It will even download and patch the original tarballs for you.
- The resulting script and associated patches, and the latest version of this doc, are available at kegel.com/crosstool.

gcc / glibc / binutils / kernel versions

Crosstool build reports: http://kegel.com/crosstool/crosstool-0.38/buildlogs/

gcc-4.1-20050709 cgcc-3.3.6 glibc-2.3.2 binutils-2.15 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050709 cgcc-3.3.6 glibc-2.3.2 binutils-2.15 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.0.1 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-2.95.3 glibc-2.2.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	gcc-4.1-20050716 cgcc-3.3.6 glibc-2.3.2 binutils-2.16.1 linux-2.6.11.3 hdrs-2.6.11.2	
kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	alpha
ICE gdb FAIL	ICE gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	arm
ICE gdb FAIL	ICE gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	arm9tdmi
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	arm-iwmmxt
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	arm-softfloat
ICE gdb FAIL	ICE gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	arm-xscale
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	armeb
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	armv5b-softfloat
kernel ICE gdb ok	kernel ICE gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	i686
FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ia64
kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	m68k
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	mips
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	mipsel
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	powerpc-405
kernel ICE gdb ok	kernel ICE gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	ok gdb ok	powerpc-750
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	ok gdb ok	ok gdb ok	ok gdb ok	powerpc-860
kernel ICE gdb ok	kernel ICE gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	powerpc-970
kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	FAIL gdb FAIL	FAIL gdb FAIL	FAIL gdb FAIL	kernel fail gdb ok	kernel fail gdb ok	kernel fail gdb ok	s390

Can help you to find a working combination of `gcc`, `glibc`, `binutils` and kernel headers versions!

Cross-compiling toolchains - Summary

- Building a toolchain by yourself
  Tough and very long to master.
- Ready-to-use toolchains
  Available from several locations for most platforms.
- Tools to build toolchains: Buildroot and Crosstool.
  Make it easy to create a toolchain for your exact needs.

GNU / Linux workstation

Emulators

Usefulness of emulators

- Of course, can be used to run another system on top of another.
- Can be used to test kernel and applications ahead of time on a workstation without the cost of a development board.
- Make the operating system easier to debug than with actual hardware. If the OS freezes, just restart the emulator.
- Can provide debugger and tracing capabilities
- However, often emulate only the processor and a few devices at best. Don't replace the real hardware or a development board.

qemu

http://qemu.org

Fast processor emulator
using a portable dynamic translator.

2 operating modes

- Full system emulation: processor and various peripherals
  Supported: `x86`, `x86_64`, `ppc`, `arm`, `sparc`, `mips`
- User mode emulation (Linux host only): can run applications compiled for another CPU.
  Supported: `x86`, `ppc`, `arm`, `sparc`, `mips`

qemu example

System emulation

- Even easier to run: (`x86` example)

```
qemu –kernel bzImage –append “root=/bin/sh”
-initrd root.cpio.gz -m 64
```

- A lot of additional options exists. See the Qemu manual for the details.

ARM emulators

Only Free Software, of course!

- SkyEye: http://skyeye.sourceforge.net
  Emulates several ARM platforms (AT91, Xscale...) and can boot several operating systems (Linux, uClinux, and others)
- Softgun: http://softgun.sourceforge.net
  Virtual ARM system with many virtual on-board peripherals. Boots Linux.
- SWARM - Software ARM - arm7 emulator
  http://www.cl.cam.ac.uk/~mwd24/phd/swarm.html
  Can run uClinux

Other emulators

- ColdFire emulator

  http://www.slicer.ca/coldfire/

  Can boot uClinux

Emulators - Summary

- System emulators

  Useful to experiment with a full system, including the kernel

  qemu: `x86`, `x86_64`, `arm`, `sparc`, `ppc`, `mips`

  SkyEye: several `arm` architectures

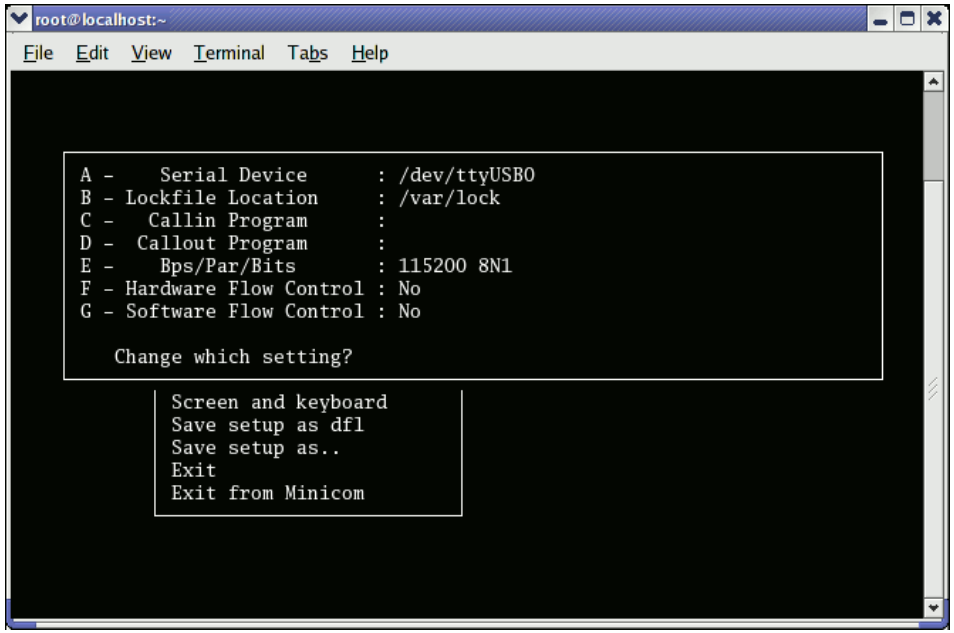
GNU / Linux workstation

Various tools

Minicom (1)

- Definition: serial communication program
- Available in all GNU / Linux distributions
- Capabilities (all through a serial link):
  - Serial console to a remote Unix system
  - File transfer
  - Modem control and dial-up
  - Serial port configuration

Minicom (2)



- Start by running `minicom -s` to setup Minicom
- A bit austere at first glance, but quickly gets friendly (see the labs for details)

Kernel overview

Linux versions and development process

Linux stable releases

Major versions

- 1 major version every 2 or 3 years

  Examples: `1.0`, `2.0`, `2.4`, `2.6`

Stable releases

- 1 stable release every 1 or 2 months

  Examples: `2.0.40`, `2.2.26`, `2.4.27`, `2.6.7` ...

Stable release updates (since March 2005)

- Updates to stable releases up to several times a week

  Address only critical issues in the latest stable release

  Examples: `2.6.11.1` to `2.6.11.7`

Linux development and testing releases

Testing releases

- Several testing releases per month, before the next stable one. You can contribute to making kernel releases more stable by testing them!
  Example: `2.6.12-rc1`

Development versions

- Unstable versions used by kernel developers before making a new stable major release
  Examples: `2.3.42`, `2.5.74`

Continued development in Linux 2.6

- Since `2.6.0`, kernel developers have been able to introduce lots of new features one by one on a steady pace, without having to make major changes in existing subsystems.
- Opening a new Linux `2.7` (or `2.9`) development branch will be required only when Linux `2.6` is no longer able to accommodate key features without undergoing traumatic changes.
- Thanks to this, more features are released to users at a faster pace.
- However, the internal kernel API can undergo changes between two `2.6.x` releases. A module compiled for a given version may no longer compile or work on a more recent one.

What's new in each Linux release? (1)

```
commit 3c92c2ba33cd7d666c5f83cc32aa590e794e91b0
Author: Andi Kleen <ak@suse.de>
Date:   Tue Oct 11 01:28:33 2005 +0200

    [PATCH] i386: Don't discard upper 32bits of HWCR on K8

    Need to use long long, not long when RMWing a MSR. I think
    it's harmless right now, but still should be better fixed
    if AMD adds any bits in the upper 32bit of HWCR.

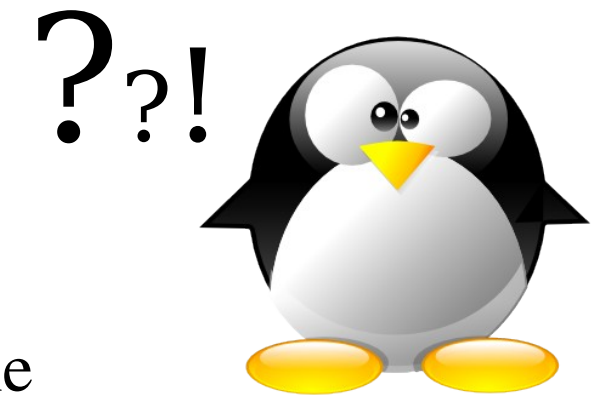
    Bug was introduced with the TLB flush filter fix for i386

    Signed-off-by: Andi Kleen <ak@suse.de>
    Signed-off-by: Linus Torvalds <torvalds@osdl.org>
...
```

- The official list of changes for each Linux release is just a huge list of individual patches!
- Very difficult to find out the key changes and to get the global picture out of individual changes.

What's new in each Linux release? (2)

- Fortunately, a summary of key changes with enough details is available on http://wiki.kernelnewbies.org/LinuxChanges



- For each new kernel release, you can also get the changes in the kernel internal API: http://lwn.net/Articles/2.6-kernel-api/

- What's next? Documentation/feature-removal-schedule.txt lists the features, subsystems and APIs that are planned for removal (announced 1 year in advance).

Linux kernel licensing constraints

- The whole Linux sources are Free Software released under the GNU General Public License (GPL)
- For any device embedding Linux and Free Software, you have to release sources to the end user. You have no obligation to release them to anybody else!
- According to the GPL, only Linux drivers with a GPL compatible license are allowed.
- Proprietary modules are tolerated (but not recommended) as long as they cannot be considered as derived work of GPLed code.
- Proprietary drivers cannot be statically compiled in the kernel.

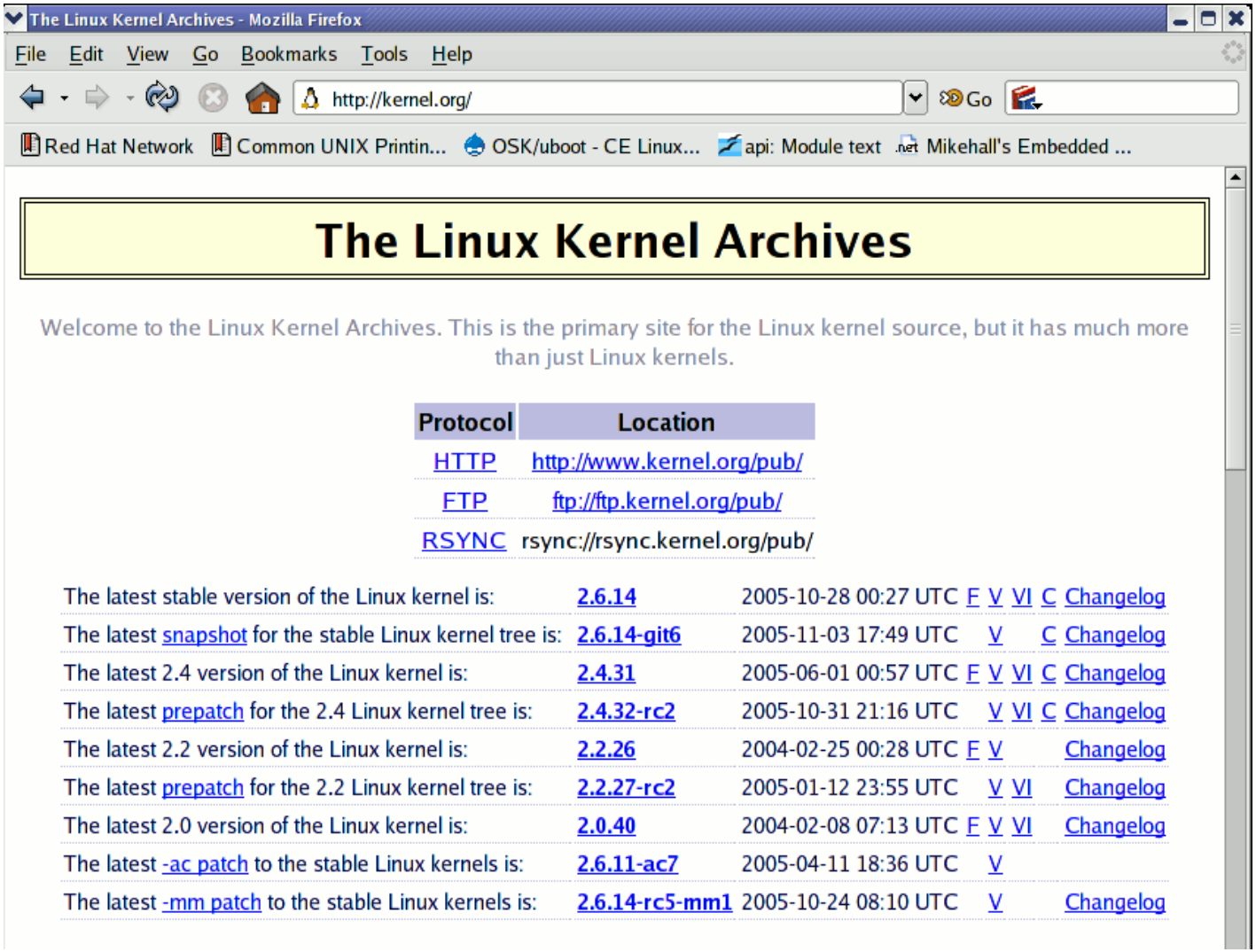
Compiling and booting Linux

Getting the sources

Linux kernel size

- Linux 2.6.16 sources:
  Raw size: 260 MB (20400 files, approx 7 million lines of code)
  `bzip2` compressed tar archive: 39 MB (best choice)
  `gzip` compressed tar archive: 49 MB
- Minimum compiled Linux kernel size (with Linux-Tiny patches)
  approx 300 KB (compressed), 800 KB (raw)
- Why are these sources so big?
  Because they include thousands of device drivers, many network protocols, support many architectures and filesystems...
- The Linux core (scheduler, memory management...) is pretty small!

kernel.org



Getting Linux sources: 2 possibilities

Full sources

- The easiest way, but longer to download.
- Example:
  http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.14.1.tar.bz2

Or patch against the previous version

- Assuming you already have the full sources of the previous version
- Example:
  http://kernel.org/pub/linux/kernel/v2.6/patch-2.6.14.bz2 (2.6.13 to 2.6.14)
  http://kernel.org/pub/linux/kernel/v2.6/patch-2.6.14.7.bz2 (2.6.14 to 2.6.14.7)

Downloading full kernel sources

Downloading from the command line

- With a web browser, identify the version you need on http://kernel.org
- In the right directory, download the source archive and its signature (copying the download address from the browser):

```
wget http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.11.12.tar.bz2
wget http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.11.12.tar.bz2.sign
```

- Check the electronic signature of the archive:

```
gpg --verify linux-2.6.11.12.tar.bz2.sign
```

- Extract the contents of the source archive:

```
tar jxvf linux-2.6.11.12.tar.bz2
```

`~/.wgetrc` config file for proxies:

```
http_proxy = <proxy>:<port>
ftp_proxy = <proxy>:<port>
proxy_user = <user> (if any)
proxy_password = <passwd> (if any)
```

Downloading kernel source patches (1)

Assuming you already have the `linux-x.y.<n-1>` version

- Identify the patches you need on http://kernel.org with a web browser
- Download the patch files and their signature:

Patch from `2.6.10` to `2.6.11`

```
wget ftp://ftp.kernel.org/pub/linux/kernel/v2.6/patch-2.6.11.bz2
wget ftp://ftp.kernel.org/pub/linux/kernel/v2.6/patch-2.6.11.bz2.sign
```

Patch from `2.6.11` to `2.6.11.12` (latest stable fixes)

```
wget http://www.kernel.org/pub/linux/kernel/v2.6/patch-2.6.11.12.bz2
wget http://www.kernel.org/pub/linux/kernel/v2.6/patch-2.6.11.12.bz2.sign
```

Downloading kernel source patches (2)

- Check the signature of patch files:

```
gpg --verify patch-2.6.11.bz2.sign
gpg --verify patch-2.6.11.12.bz2.sign
```

- Apply the patches in the right order:

```
cd linux-2.6.10/
bzcat ../patch-2.6.11.bz2 | patch -p1
bzcat ../patch-2.6.11.12.bz2 | patch -p1
cd ..
mv linux-2.6.10 linux-2.6.11.12
```

Checking the integrity of sources

Kernel source integrity can be checked through OpenPGP digital signatures.
Full details on http://www.kernel.org/signature.html

- If needed, read http://www.gnupg.org/gph/en/manual.html and create a new private and public keypair for yourself.
- Import the public GnuPG key of kernel developers:
  - `gpg --keyserver pgp.mit.edu --recv-keys 0x517D0F0E`
  - If blocked by your firewall, look for `0x517D0F0E` on http://pgp.mit.edu/ , copy and paste the key to a `linuxkey.txt` file:
    `gpg --import linuxkey.txt`
- Check the signature of files:
  `gpg --verify linux-2.6.11.12.tar.bz2.sign`

Anatomy of a patch file

A patch file is the output of the `diff` command

```
diff -Nru a/Makefile b/Makefile
--- a/Makefile  2005-03-04 09:27:15 -08:00
+++ b/Makefile  2005-03-04 09:27:15 -08:00
@@ -1,7 +1,7 @@
 VERSION = 2
 PATCHLEVEL = 6
 SUBLEVEL = 11
-EXTRAVERSION =
+EXTRAVERSION = .1
 NAME=Woozy Numbat

 # *DOCUMENTATION*
```

`diff` command line

File date info

Line numbers in files

Context info: 3 lines before the change
Useful to apply a patch when line numbers changed

Removed line(s) if any

Added line(s) if any

Context info: 3 lines after the change

Using the patch command

The `patch` command applies changes to files in the current directory:

- Making changes to existing files
- Creating or deleting files and directories

`patch` usage examples:

- `patch -p<n> < diff_file`
- `cat diff_file | patch -p<n>`
- `bzcat diff_file.bz2 | patch -p<n>`
- `zcat diff_file.gz | patch -p<n>`

`n`: number of directory levels to skip in the file paths

You can reverse a patch with the `-R` option

Applying a Linux patch

Linux patches...

- Always to apply to the `x.y.<z-1>` version
- Always produced for `n=1` (that's what everybody does... do it too!)
- Downloadable in `gzip` and `bzip2` (much smaller) compressed files.
- Linux patch command line example:

```
cd linux-2.6.10
bzcat ../patch-2.6.11.bz2 | patch -p1
cd ..; mv linux-2.6.10 linux-2.6.11
```

- Keep patch files compressed: useful to check their signature later.
  You can still view (or even edit) the uncompressed data with `vi`:
  `vi patch-2.6.11.bz2` (on the fly (un)compression)

Accessing development sources (1)

- Kernel development sources are now managed with git
- You can browse Linus' git tree (if you just need to check a few files): http://www.kernel.org/git/?p=linux/kernel/git/torvalds/linux-2.6.git;a=tree
- Get and compile git from http://kernel.org/pub/software/scm/git/
- Get and compile the cogito front-end from http://kernel.org/pub/software/scm/cogito/
- If you are behind a proxy, set Unix environment variables defining proxy settings. Example:

```
export http_proxy="proxy.server.com:8080"
export ftp_proxy="proxy.server.com:8080"
```

Accessing development sources (2)

- Pick up a git development tree on http://kernel.org/git/
- Get a local copy ("clone") of this tree.
  Example (Linus tree, the one used for Linux stable releases):

```
cg-clone http://kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git
```
or `cg-clone rsync://rsync.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git`

- Update your copy whenever needed (Linus tree example):

```
cd linux-2.6
cg-update origin
```

More details available
on http://git.or.cz/ or http://linux.yyz.us/git-howto.html

Compiling and booting Linux

Structure of source files

Linux sources structure (1)

`arch/<arch>`	Architecture specific code
`arch/<arch>/mach-<mach>`	Machine / board specific code
`COPYING`	Linux copying conditions (GNU GPL)
`CREDITS`	Linux main contributors
`crypto/`	Cryptographic libraries
`Documentation/`	Kernel documentation. Don't miss it!
`drivers/`	All device drivers (`drivers/usb/`, etc.)
`fs/`	Filesystems (`fs/ext3/`, etc.)
`include/`	Kernel headers
`include/asm-<arch>`	Architecture and machine dependent headers
`include/linux`	Linux kernel core headers
`init/`	Linux initialization (including `main.c`)
`ipc/`	Code used for process communication

Linux sources structure (2)

`kernel/`	Linux kernel core (very small!)
`lib/`	Misc library routines (`zlib`, `crc32`...)
`MAINTAINERS`	Maintainers of each kernel part. Very useful!
`Makefile`	Top Linux makefile (sets arch and version)
`mm/`	Memory management code (small too!)
`net/`	Network support code (not drivers)
`README`	Overview and building instructions
`REPORTING-BUGS`	Bug report instructions
`scripts/`	Scripts for internal or external use
`security/`	Security model implementations (SELinux...)
`sound/`	Sound support code and drivers
`usr/`	Early user-space code (`initramfs`)

On-line kernel documentation

http://free-electrons.com/kerneldoc/

- Provided for all recent kernel releases
- Easier than downloading kernel sources to access documentation
- Indexed by Internet search engines
  Makes kernel pieces of documentation easier to find!
- Unlike most other sites offering this service too, also includes an HTML translation of kernel documents in the DocBook format.

Never forget documentation in the kernel sources! It's a very valuable way of getting information about the kernel.

Compiling and booting Linux

Kernel source management tools

LXR: Linux Cross Reference

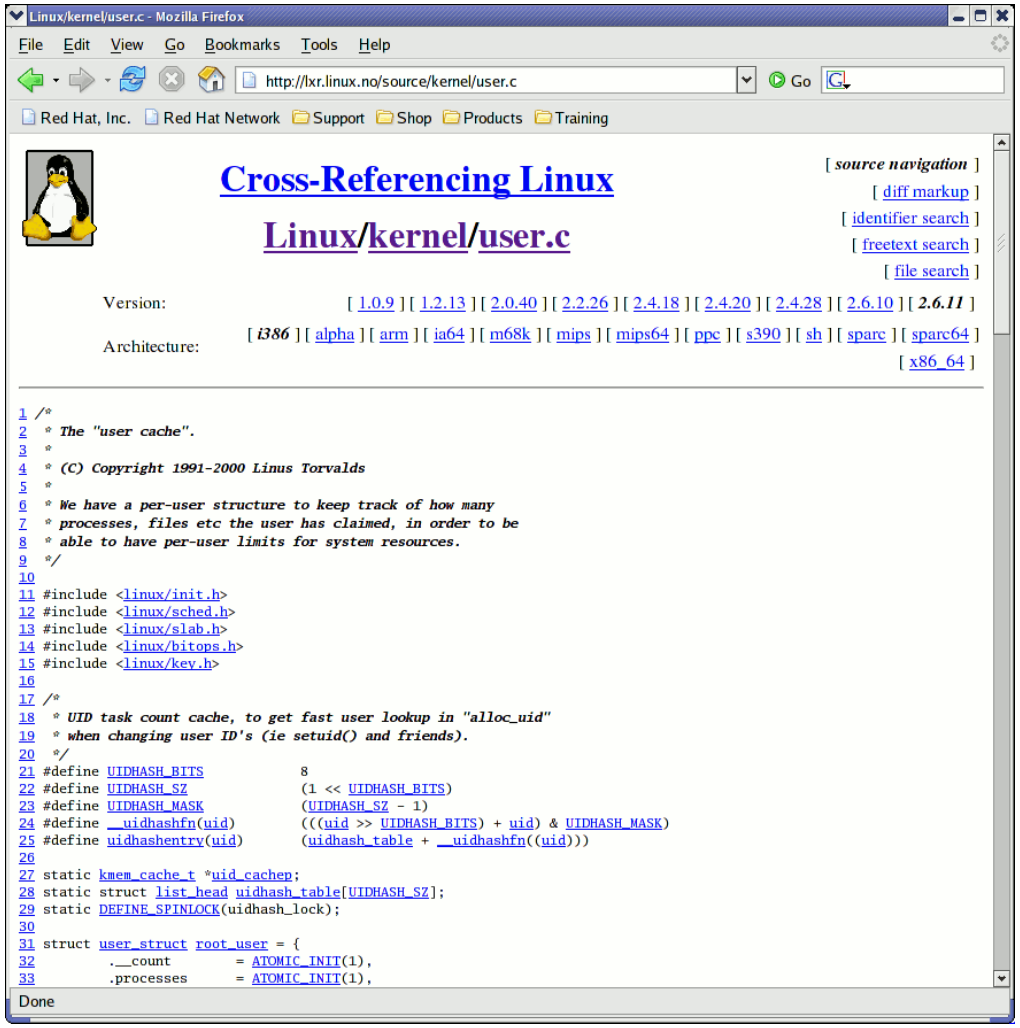
http://sourceforge.net/projects/lxr

Generic source indexing tool and code browser

- Web server based
  Very easy and fast to use
- Identifier or text search available
- Very easy to find the declaration, implementation or usages of symbols
- Supports C and C++
- Supports huge code projects such as the Linux kernel (260 M in Apr. 2006)

- Takes a little bit of time and patience to setup (configuration, indexing, server configuration).
- Initial indexing quite slow:
  Linux 2.6.11: 1h 40min on P4 M
  1.6 GHz, 2 MB cache
- You don't need to set up LXR by yourself.
  Use our http://lxr.free-electrons.com server!
  Other servers available on the Internet:
  http://free-electrons.com/community/kernel/lxr/

LXR screenshot



Ketchup - Easy access to kernel source trees

http://www.selenic.com/ketchup/wiki/

- Makes it easy to get the latest version of a given kernel source tree (`2.4`, `2.6`, `2.6-rc`, `2.6-git`, `2.6-mm`, `2.6-rt`...)
- Only downloads the needed patches.
  Reverts patches when needed to apply a more recent patch.
- Also checks the signature of sources and patches.

Ketchup examples

- Get the version in the current directory:

```
> ketchup -m
2.6.10
```

- Upgrade to the latest stable version:

```
> ketchup 2.6-tip
2.6.10 -> 2.6.12.5
Applying patch-2.6.11.bz2
Applying patch-2.6.12.bz2
Applying patch-2.6.12.5.bz2
```

More on http://selenic.com/ketchup/wiki/index.cgi/ExampleUsage

Compiling and booting Linux

Kernel configuration

Kernel configuration overview

- `Makefile` edition

  Setting the version and target architecture if needed

- Kernel configuration: defining what features to include in the kernel:

```
make [config|xconfig|gconfig|menuconfig|oldconfig]
```

  - Kernel configuration file (Makefile syntax) stored in the `.config` file at the root of the kernel sources

Makefile changes

- To identify your kernel image with others build from the same sources, use the `EXTRAVERSION` variable:

```
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 15
EXTRAVERSION = -acme1
```

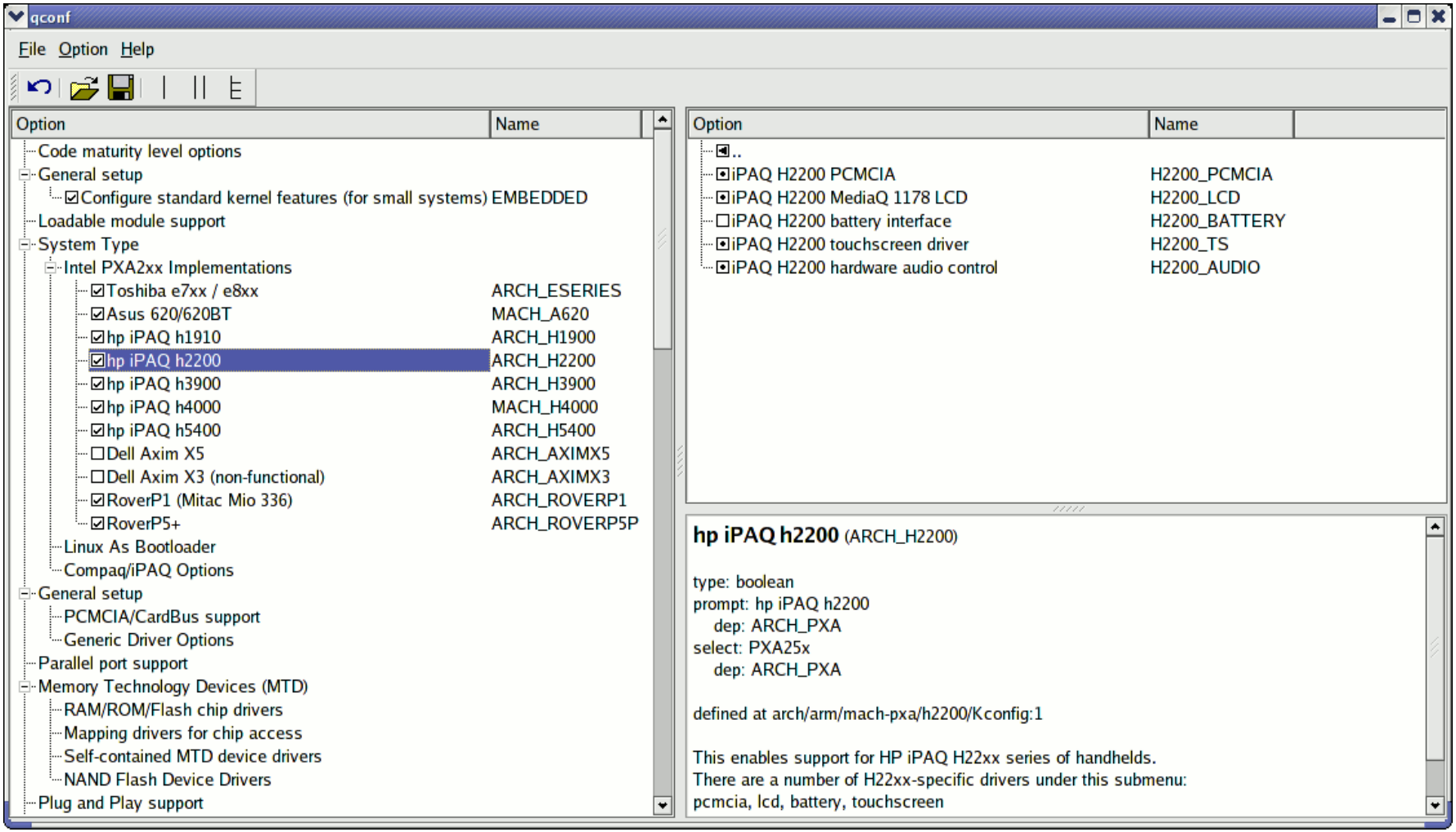
- `uname -r` will return:
  `2.6.15-acme1`

make xconfig

```
make xconfig
```

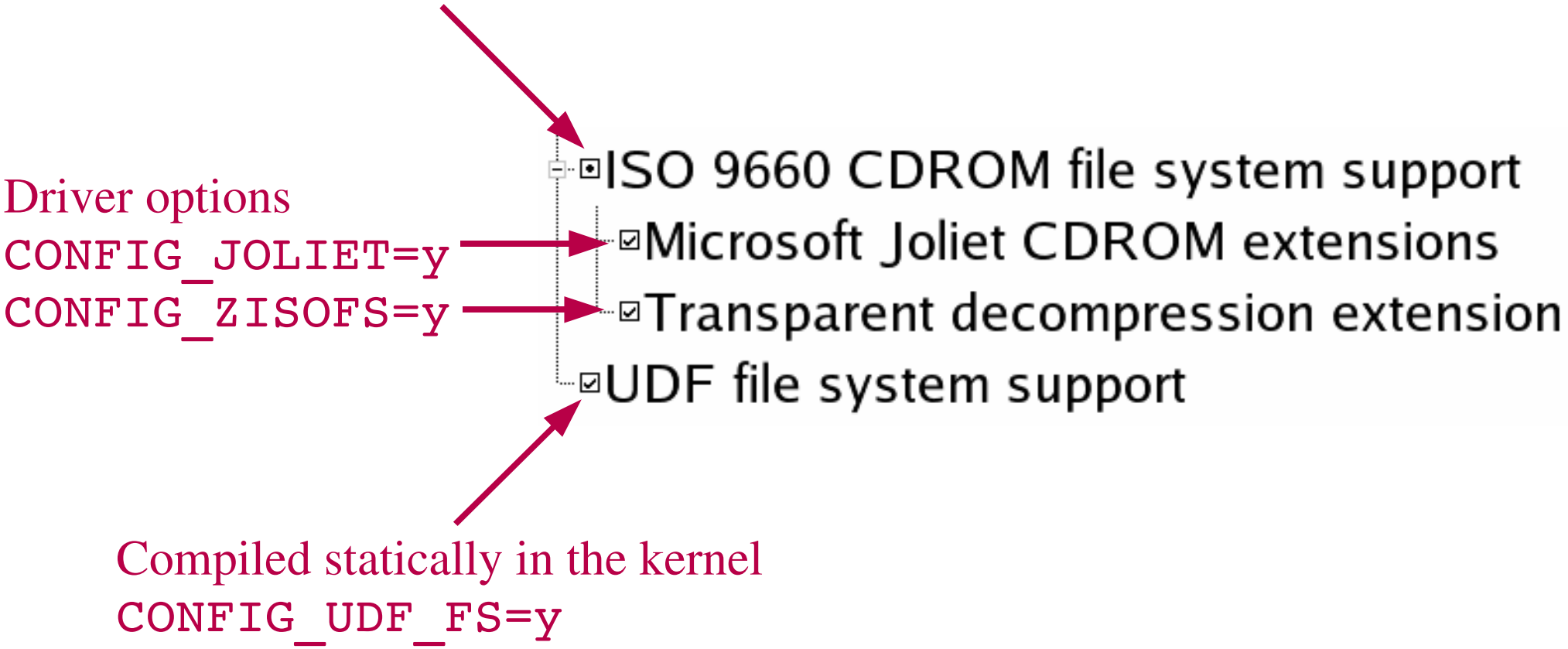
- Graphical interface to configure the Linux kernel.
- Make sure you read
  `help -> introduction: useful options!`

make xconfig screenshot



Compiling statically or as a module

Compiled as a module (separate file)
`CONFIG_ISO9660_FS=m`



Driver options
`CONFIG_JOLIET=y`
`CONFIG_ZISOFS=y`

Compiled statically in the kernel
`CONFIG_UDF_FS=y`

make config / menuconfig / gconfig

`make config`

- Asks you the questions 1 by 1. Extremely long!

`make menuconfig`

- Same old text interface as in Linux 2.4.
  Useful when no graphics are available.
  Pretty convenient too!

`make gconfig`

- New GTK based graphical configuration interface.
  Functionality similar to that of `make xconfig`.

make oldconfig

`make oldconfig`

- Needed very often!
- Useful to upgrade a `.config` file from an earlier kernel release
- Issues warnings for obsolete symbols
- Asks for values for new symbols

If you edit a `.config` file by hand, it's strongly recommended to run `make oldconfig` afterwards!

make allnoconfig

`make allnoconfig`

- Only sets strongly recommended settings to `y`.
- Sets all other settings to `n`.
- Very useful in embedded systems to select only the minimum required set of features and drivers.
- Much more convenient than unselecting hundreds of features one by one!

make help

`make help`

- Lists all available `make` targets
- Useful to get a reminder, or to look for new or advanced options!

Important Configuration Options

- EXPERIMENTAL
  - Show alpha stage or obsolete drivers and other code.
- LOCALVERSION
  - Gets added to the kernel version.
- EMBEDDED
  - Enables the advanced option relevant for embedded systems. Required to see some of the options ahead.
- MODULES
  - Include modules in the kernel

Performance and Size Configuration Options

- IOSCHED_NOOP
  - Chooses the minimal IO scheduler, best suited for flash and local storage less based systems.
- KALLSYMS
  - Include symbolic crash information required to show function names in crash dump. Symbol names take size.
- PRINTK
  - Turn off all log messages. Makes kernel smaller by removing all log strings.

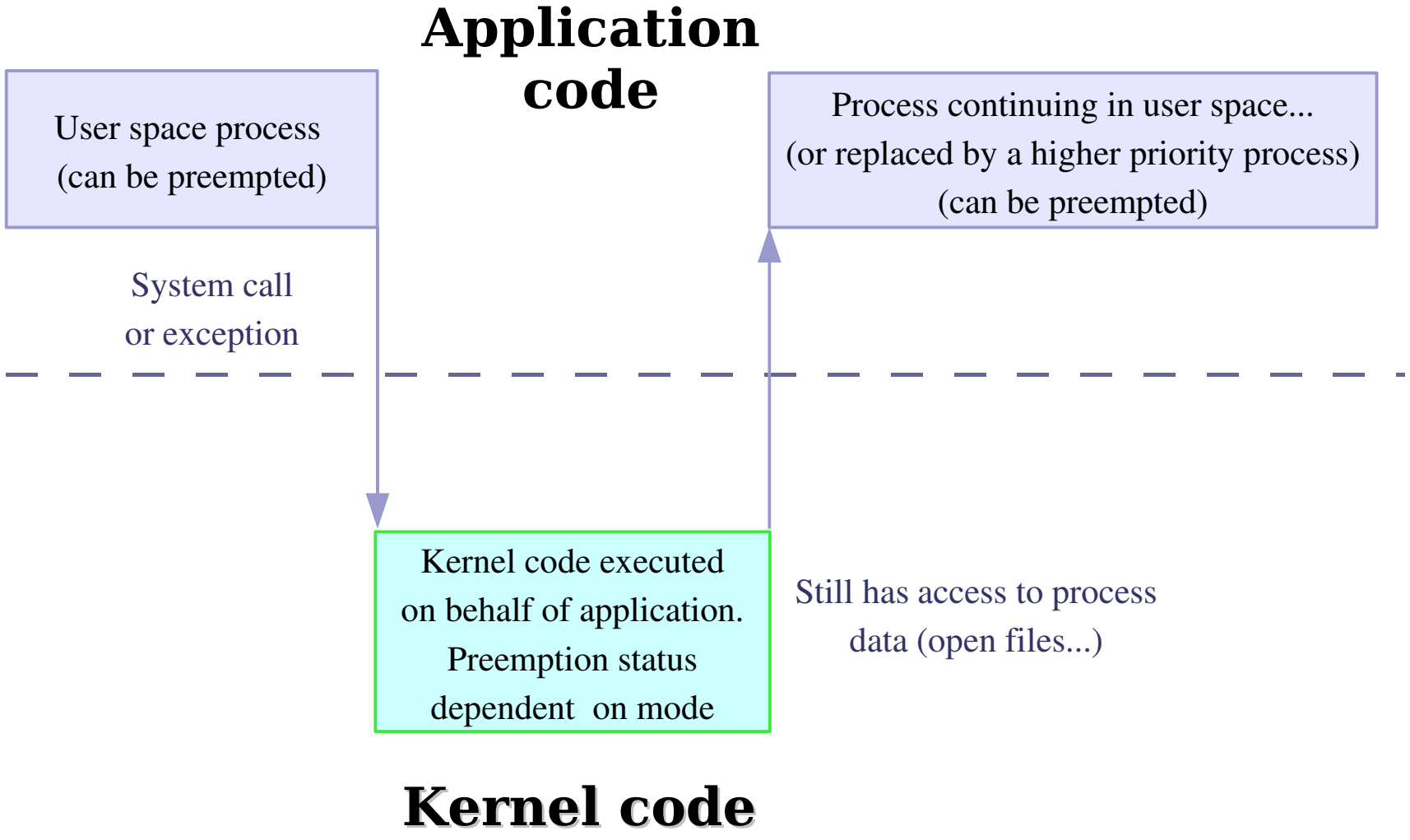
Performance and Size Configuration Options

- BUG
  - Turn off assert checking.
- BASE_FULL
  - Use smaller versions for some kernel data structures. Smaller in size but slower.
- CC_OPTIMIZE_FOR_SIZE
  - Optimize kernel for size, not speed. May produce bad code with some GCC versions.

Scheduling Latency options

- Preemption is a high priority task taking the place of a low prority one.
- User space code is always preemptive.
- PREEMPT_NONE
  - None preemptive kernel. Best throughput, poor latency.
- PREEMPT_VOLUNTARY
  - Voluntary preemption points. Still problem with jitter.
- PREEMPT
  - Non critical section preemptive kernel. All kernel code is preemptive, unless locked.

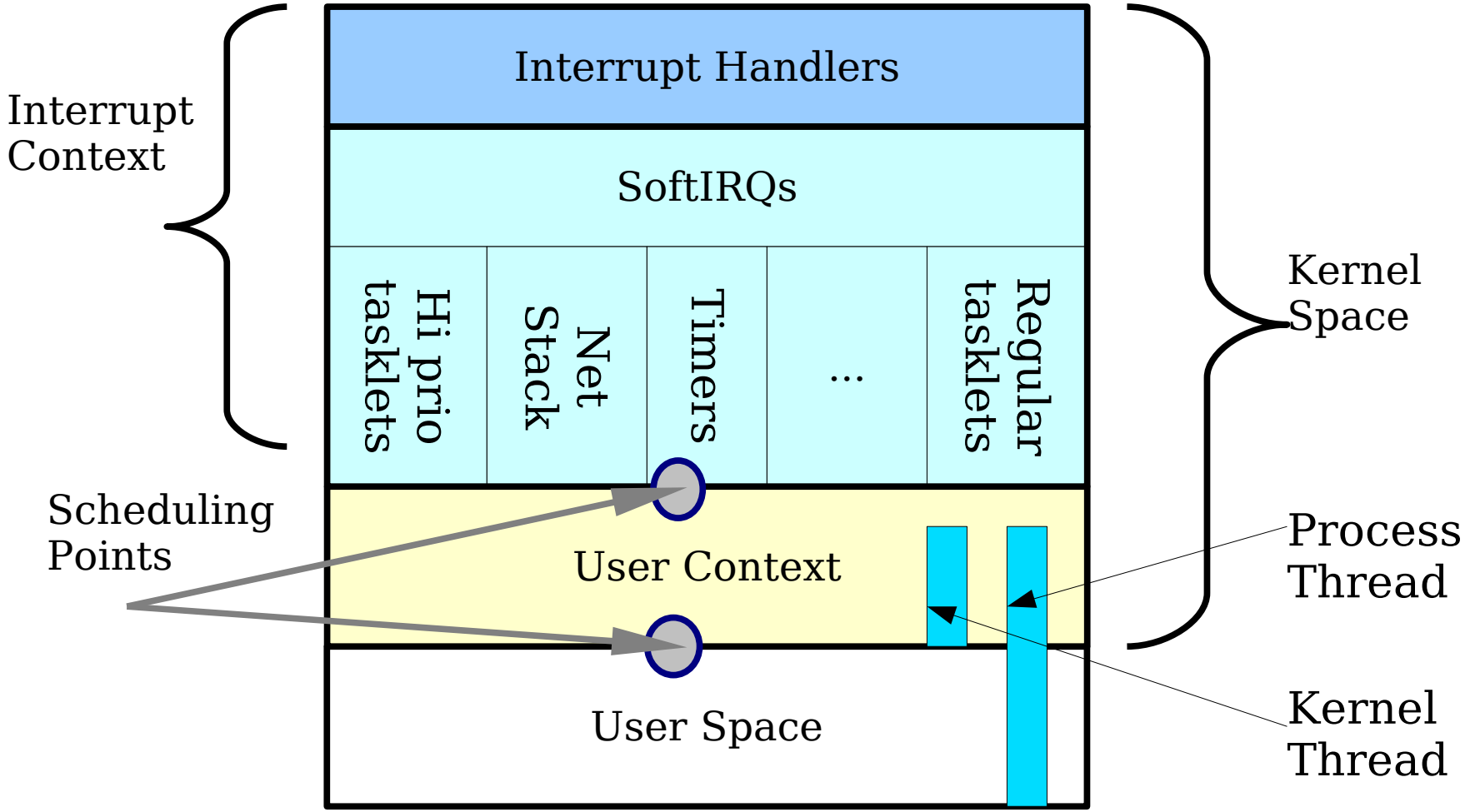
Preemption in the kernel



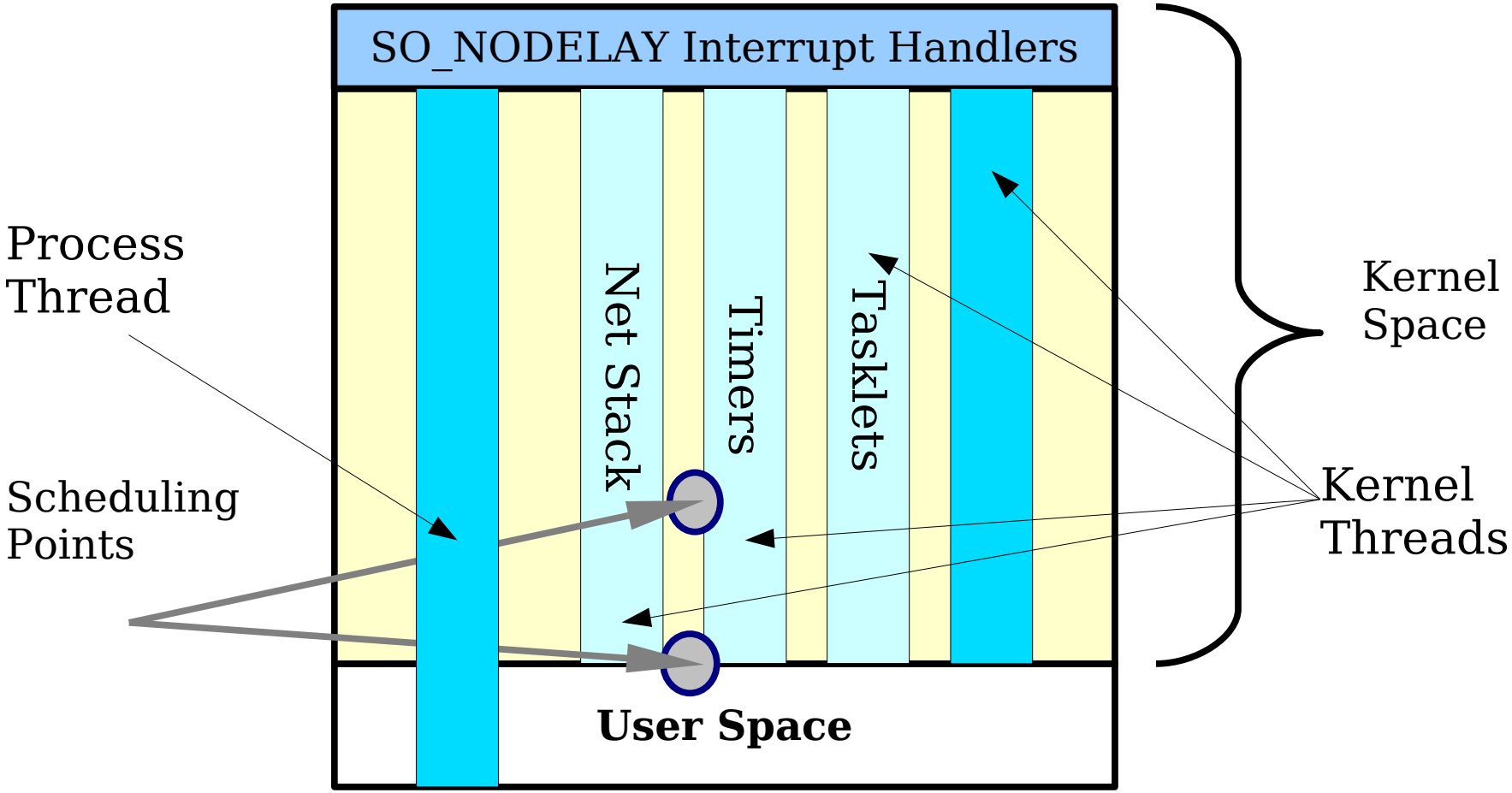
PREEMPT-RT

- Fully preemptive kernel
  - Kernel config option PREEMPT_RT
  - Patched 2.6 / 2.6.18 +
  - Preemptible critical sections
  - Preemptible interrupt handlers
  - Preemptible "interrupt disable" sequences
  - Kernel locks priority inheritance
  - Deferred operations
  - Latency-reduction measures

Linux Contexts



PREEMPT-RT Linux Contexts



Interface Changes

- Spinlocks and local_irq_save() no longer disable hardware interrupts.
- Spinlocks no longer disable preemption.
- Raw_ variants for spinlocks and local_irq_save() preserve original meaning for SO_NODELAY interrupts.
- Semaphores and spinlocks employ priority inheritance

Deferred Operations

- No longer legal to acquire spinlock while preemption or interrupts disabled.
- So operation requiring a spinlock must sometime be delayed:
  - put_task_struct_delayed()
  - mmdrop_delayed()
  - TIF_NEED_RESCHED_DELAYED
  - wake_up_proccess_sync()

Debug Kernel Configuration Options

- PROFILING
  - Enable the profiling support menu.
- OPROFILE
  - A profiling system for Linux kernel and applications.
- DEBUG_KERNEL
  - Open the kernel debug options.
- PRINTK_TIMES
  - Add time stamp to log messages.
- MAGIC_SYSRQ
  - Online serial/keyboard debug hook (Documentaion/sysrq.txt)

Compiling and booting Linux

Compiling the kernel

Compiling and installing the kernel

Compiling step

- `make`

Install steps (logged as `root`!)

- `make install`
- `make modules_install`

Dependency management

- When you modify a regular kernel source file, `make` only rebuilds what needs recompiling. That's what it is used for.
- However, the `Makefile` is quite pessimistic about dependencies. When you make significant changes to the `.config` file, `make` often redoes much of the compile job!

Compiling faster on multiprocessor hosts

- If you are using a workstation with `n` processors, you may roughly divide your compile time by `n` by compiling several files in parallel
- `make -j <n>`
  Runs several targets in parallel, whenever possible
- Using `make -j 2` or `make -j 3` on single processor workstations. This doesn't help much. In theory, several parallel compile jobs keep the processor busy while other processes are waiting for files to be read of written. In practice, you don't get any significant speedup (not more than 10%), unless your I/Os are very slow.

Compiling faster with ccache

http://ccache.samba.org/

Compiler cache for C and C++, already shipped by some distributions

Much faster when compiling the same file a second time!

- Very useful when `.config` file change are frequent.
- Use it by adding a `ccache` prefix

  to the `CC` and `HOSTCC` definitions in `Makefile`:

```
CC                  = ccache $(CROSS_COMPILE)gcc
HOSTCC              = ccache gcc
```

- Performance benchmarks:

  -63%: with a Fedora Core 3 config file (many modules!)

  -82%: with an embedded Linux config file (much fewer modules!)

Kernel compiling tips

- View the full (`gcc`, `ld`...) command line:
  `make V=1`
- Clean-up generated files
  (to force re-compiling drivers):
  `make clean`
- Remove all generated files
  (mainly to create patches)
  Caution: also removes your `.config` file!
  `make mrproper`

Generated files

Created when you run the `make` command

- `vmlinux`

  Raw Linux kernel image, non compressed.

- `arch/<arch>/boot/zImage` (default image on `arm`)

  `zlib` compressed kernel image

- `arch/<arch>/boot/bzImage` (default image on `i386`)

  Also a `zlib` compressed kernel image.

  Caution: `bz` means "big zipped" but not "`bzip2` compressed"!

  (`bzip2` compression support only available on `i386` as a tactical patch. Not very attractive for small embedded systems though: consumes 1 MB of RAM for decompression).

Files created by make install

- `/boot/vmlinuz-<version>`
  Compressed kernel image. Same as the one in `arch/<arch>/boot`
- `/boot/System.map-<version>`
  Stores kernel symbol addresses
- `/boot/initrd-<version>.img` (when used by your distribution)
  Initial RAM disk, storing the modules you need to mount your root filesystem. `make install` runs `mkinitrd` for you!
- `/etc/grub.conf` or `/etc/lilo.conf`
  `make install` updates your bootloader configuration files to support your new kernel! It reruns `/sbin/lilo` if LILO is your bootloader.

Files created by make modules_install (1)

`/lib/modules/<version>/`: Kernel modules + extras

- `build/`

  Everything needed to build more modules for this kernel: `Makefile`, `.config` file, module symbol information (`module.symVers`), kernel headers (`include/` and `include/asm/`)

- `kernel/`

  Module `.ko` (Kernel Object) files, in the same directory structure as in the sources.

Files created by make modules_install (2)

`/lib/modules/<version>/` (continued)

- `modules.alias`

  Module aliases for module loading utilities. Example line:
  `alias sound-service-?-0 snd_mixer_oss`

- `modules.dep`

  Module dependencies (see the Loadable kernel modules section)

- `modules.symbols`

  Tells which module a given symbol belongs to.

All the files in this directory are text files.

Don't hesitate to have a look by yourself!

Compiling the kernel in a nutshell

- Edit version information in the `Makefile` file
- ```
  make xconfig
  make
  make install
  make modules_install
  ```
  ```

Compiling and booting Linux

Cross-compiling the kernel

Makefile changes

- Update the version as usual
- You should change the default target platform.
  Example: ARM platform, cross-compiler command: `arm-linux-gcc`

```
ARCH    ?= arm
CROSS_COMPILE  ?= arm-linux-
```

  (The Makefile defines later `CC = $(CROSS_COMPILE)gcc`)

See comments in `Makefile` for details

Configuring the kernel

```
make xconfig
```

- Same as in native compiling.
- Don't forget to set the right board / machine type!

Ready-made config files

```
assabet_defconfig       integrator_defconfig  mainstone_defconfig
badge4_defconfig        iq31244_defconfig     mx1ads_defconfig
bast_defconfig          iq80321_defconfig     neponset_defconfig
cerfcube_defconfig      iq80331_defconfig     netwinder_defconfig
clps7500_defconfig      iq80332_defconfig     omap_h2_1610_defconfig
ebsa110_defconfig       ixdp2400_defconfig    omnimeter_defconfig
edb7211_defconfig       ixdp2401_defconfig    pleb_defconfig
enp2611_defconfig       ixdp2800_defconfig    pxa255-idp_defconfig
ep80219_defconfig       ixdp2801_defconfig    rpc_defconfig
epxa10db_defconfig      ixp4xx_defconfig      s3c2410_defconfig
footbridge_defconfig    jornada720_defconfig  shannon_defconfig
fortunet_defconfig      lart_defconfig        shark_defconfig
h3600_defconfig         lpd7a400_defconfig    simpad_defconfig
h7201_defconfig         lpd7a404_defconfig    smdk2410_defconfig
h7202_defconfig         lubbock_defconfig     versatile_defconfig
hackkit_defconfig       lusl7200_defconfig
```

`arch/arm/configs` example

Using ready-made config files

- Default configuration files available for many boards / machines! Check if one exists in `arch/<arch>/configs/` for your target.
- Example: if you found an `acme_defconfig` file, you can run: `make acme_defconfig`
- Using `arch/<arch>/configs/` is a very good good way of releasing a default configuration file for a group of users or developers.

Cross-compiling setup

Example

- If you have an ARM cross-compiling toolchain in `/usr/local/arm/3.3.2/`
- You just have to add it to your Unix search path:
  ```
  export PATH=/usr/local/arm/3.3.2/bin:$PATH
  ```

Choosing a toolchain

- See the `Documentation/Changes` file in the sources for details about minimum tool versions requirements.
- More about toolchains: Free Software tools for embedded systems training: http://free-electrons.com/training/devtools/

Building the kernel

- Run
  `make`
- Copy
  `arch/<platform>/boot/zImage`
  to the target storage
- You can customize `arch/<arch>/boot/install.sh` so that `make install` does this automatically for you.
- `make INSTALL_MOD_PATH=<dir>/ modules_install`
  and copy `<dir>/` to `/lib/modules/` on the target storage

Cross-compiling summary

- Edit `Makefile`: set `ARCH`, `CROSS_COMPILE` and `EXTRA_VERSION`
- Get the default configuration for your machine:
  `make <machine>_defconfig` (if existing in `arch/<arch>/configs`)
- Refine the configuration settings according to your requirements:
  `make xconfig`
- Add the crosscompiler path to your `PATH` environment variable
- Compile the kernel: `make`
- Copy the kernel image from `arch/<arch>/boot/` to the target
- Copy modules to a directory which you replicate on the target:
  `make INSTALL_MOD_PATH=<dir> modules_install`

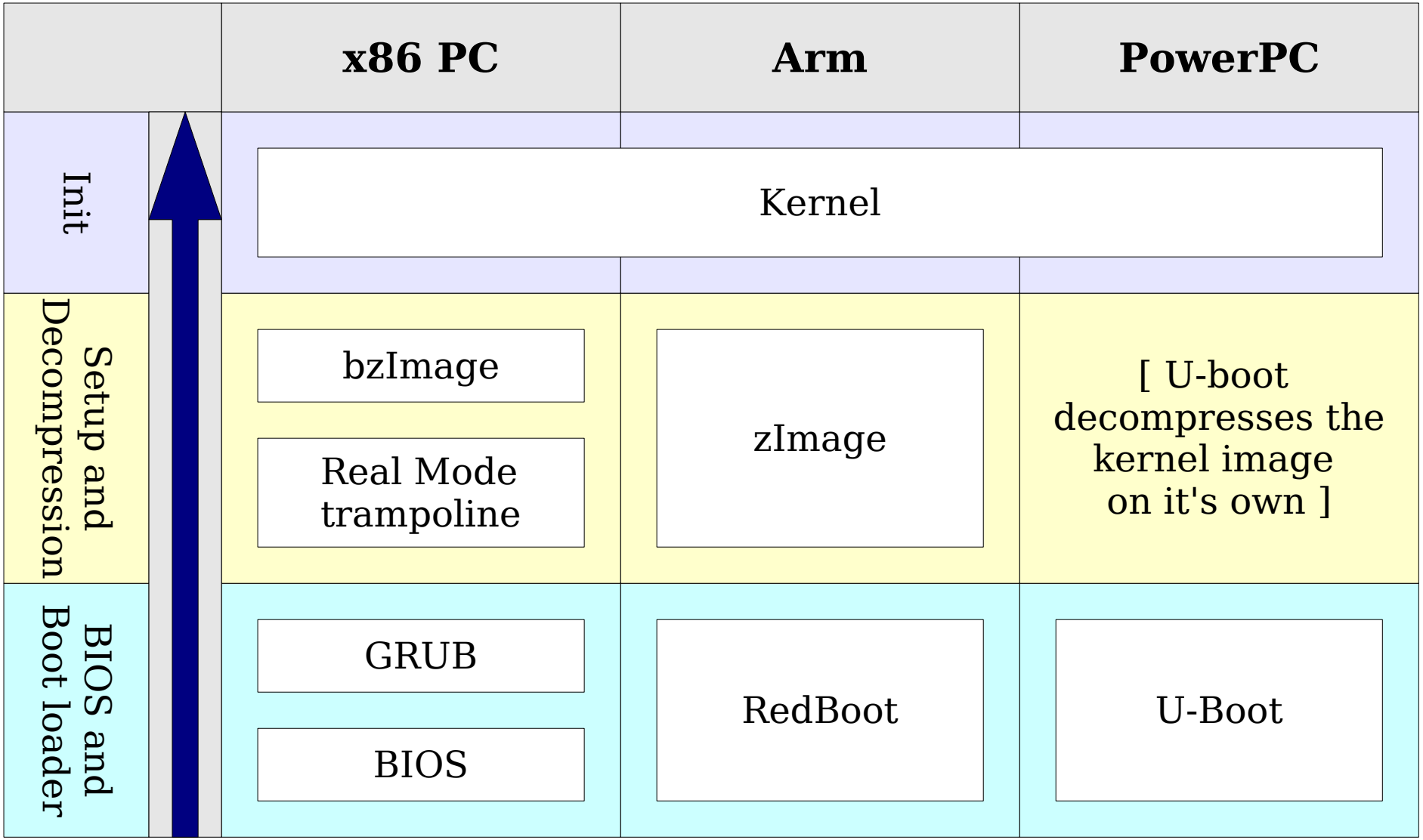
Kernel overview

Boot Sequence

Linux Boot Process

- BIOS and/or boot loader initializes hardware
- Boot loader loads kernel image into memory
  - Boot loader can get kernel image from flash, HD, network
  - Possibly also loads a file system to RAM in the form of initrd or initramfs archives
- Boot loader or kernel decompress compressed kernel
- Kernel performs internal (has table, lists etc.) and hardware (device driver) setup.
- Kernel finds and mount root file system
- Kernel executes the “/sbin/init” application.

Boot Sequences



Compiling and booting Linux

Bootloaders

2-stage bootloaders

- At startup, the hardware automatically executes the bootloader from a given location, usually with very little space (such as the boot sector on a PC hard disk)
- Because of this lack of space, 2 stages are implemented:
  - 1st stage: minimum functionality. Just accesses the second stage on a bigger location and executes it.
  - 2nd stage: offers the full bootloader functionality. No limit in what can be implemented. Can even be an operating system itself!

x86 bootloaders

- LILO: LInux LOad. Original Linux bootloader. Still in use!
  http://freshmeat.net/projects/lilo/
  Supports: `x86`
- GRUB: GRand Unified Bootloader from GNU. More powerful.
  http://www.gnu.org/software/grub/
  Supports: `x86`
- SYSLINUX: Utilities for network and removable media booting
  http://syslinux.zytor.com
  Supports: `x86`

Generic bootloaders

- Das U-Boot: Universal Bootloader from Denk Software
  The most used on `arm`.
  http://u-boot.sourceforge.net/
  Supports: `arm`, `ppc`, `mips`, `x86`
- RedBoot: eCos based bootloader from Red-Hat
  http://sources.redhat.com/redboot/
  Supports: `x86`, `arm`, `ppc`, `mips`, `sh`, `m68k`...
- uMon: MicroMonitor general purpose, multi-OS bootloader
  http://microcross.com/html/micromonitor.html
  Supports: ARM, ColdFire, SH2, 68K, MIPS, PowerPC, Xscale...

Other bootloaders

- LAB: Linux As Bootloader, from Handhelds.org
  http://handhelds.org/cgi-bin/cvsweb.cgi/linux/kernel26/lab/
  Idea: use a trimmed Linux kernel with only features needed in a bootloader (no scheduling, etc.). Reuses flash and filesystem access, LCD interface, without having to implement bootloader specific drivers.
  Supports: `arm` (still experimental)
- And many more: lots of platforms have their own!

Postprocessing kernel image for U-boot

The U-boot bootloader needs extra information to be added to the kernel and initrd image files.

- `mkimage` postprocessing utility provided in U-boot sources
- Kernel image postprocessing:
  `make uImage`

Postprocessing initrd image for U-boot

```
mkimage
  -n initrd \
  -A arm \
  -O linux \
  -T ramdisk \
  -C gzip \
  -d rd-ext2.gz \
  uInitrd
```

Name
Architecture
Operating System
Type
Compression
Input file
Output file

Compiling U-boot mkimage

If you don't have `mkimage` yet

- Get the U-boot sources from
  http://u-boot.sourceforge.net/
- In the U-boot source directory:
  Find the name of the config file for your board in
  `include/configs` (for example: `omap1710h3.h`)
  `make omap1710h3_config` (`.h` replaced by `_config`)
  `make` (or `make -k` if you have minor failures)
  `cp tools/mkimage /usr/local/bin/`

Configuring tftp (1)

Often in development: downloading a kernel image from the network.
Instructions for xinetd based systems (Fedora Core, Red Hat...)

- Install the `tftp-server` package if needed
- Remove `disable = yes` in `/etc/xinetd.d/tftp`
- Copy your image files to the `/tftpboot/` directory (or to the location specified in `/etc/xinetd.d/tftp`)
- You may have to disable SELinux in `/etc/selinux/config`
- Restart xinetd:
  `/etc/init.d/xinetd restart`

Configuring tftp (2)

On systems like Debian (or Knoppix) GNU/Linux

- Set `RUN_DAEMON="yes"`
  in `/etc/default/tftpd-hpa`
- Copy your images to `/var/lib/tftpboot`
- `/etc/hosts.allow`:
  Replace `ALL : ALL@ALL : DENY` by `ALL : ALL@ALL : ALLOW`
- `/etc/hosts.deny`:
  Comment out `ALL: PARANOID`
- Restart the server:
  `/etc/init.d/tftpd-hpa restart`

Compiling and booting Linux

Kernel booting

Kernel command line parameters

As most C programs, the Linux kernel accepts command line arguments

- Kernel command line arguments are part of the bootloader configuration settings.
- Useful to configure the kernel at boot time, without having to recompile it.
- Useful to perform advanced kernel and driver initialization, without having to use complex user-space scripts.

Kernel command line example

HP iPAQ h2200 PDA booting example:

```
root=/dev/ram0 \
rw \
init=/linuxrc \
console=ttyS0,115200n8 \
console=tty0 \
ramdisk_size=8192 \
cachepolicy=writethrough
```

Root filesystem (first ramdisk)

Root filesystem mounting mode

First userspace program

Console (serial)

Other console (framebuffer)

Misc parameters...

Hundreds of command line parameters described on `Documentation/kernel-parameters.txt`

Booting variants

XIP (Execute In Place)

- The kernel image is directly executed from the storage
- Can be faster and save RAM
  However, the kernel image can't be compressed

No initramfs / initrd

- Directly mounting the final root filesystem
  (`root` kernel command line option)

No new root filesystem

- Running the whole system from the initramfs

Usefulness of rootfs on NFS

Once networking works, your root filesystem could be a directory on your GNU/Linux development host, exported by NFS (Network File System). This is very convenient for system development:

- Makes it very easy to update files (driver modules in particular) on the root filesystem, without rebooting. Much faster than through the serial port.
- Can have a big root filesystem even if you don't have support for internal or external storage yet.
- The root filesystem can be huge. You can even build native compiler tools and build all the tools you need on the target itself (better to cross-compile though).

NFS boot setup (1)

On the PC (NFS server)

- Add the below line to your `/etc/exports` file:
  `/home/rootfs 192.168.0.202(rw,insecure,sync,no_wdelay,no_root_squash)`
  client address — NFS server options
- If not running yet, you may need to start `portmap`
  `/etc/init.d/portmap start`
- Start or restart your NFS server:

  Fedora Core: `/etc/init.d/nfs restart`

  Debian (Knoppix, KernelKit): `/etc/init.d/nfs-user-server restart`

NFS boot setup (2)

On the target (NFS client)

- Compile your kernel with `CONFIG_NFS_FS=y` and `CONFIG_ROOT_NFS=y`
- Boot the kernel with the below command line options:

```
root=/dev/nfs
```

virtual device

```
ip=192.168.1.111:192.168.1.110:192.168.1.100:255.255.255.0:at91:eth0
```

local IP address | server IP address | gateway | netmask | hostname | device

```
nfsroot=192.168.1.110:/home/nfsroot
```

NFS server IP address | Directory on the NFS server

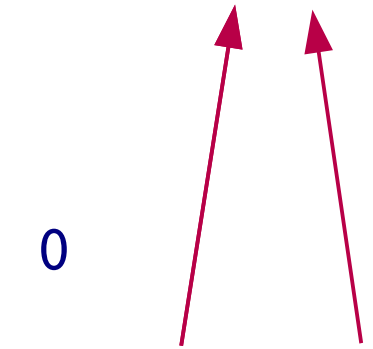
Root File System Options

	External FS	Initrd Style	Initramfs style
Kernel Version	2.4 and 2.6	2.4 and 2.6	2.6 only
Storage format	File system on storage device or network	File system image	CPIO archive
Provided by	Passed to kernel as parameter (“root=”)	Loaded and location provided by boot loader	Like initrd or statically linked into kernel image
Stored in	On device	Fixed allocation RAM disk	Dynamically allocated page cache
Run time Location	Copied to page cache	Copied to page cache	Already in page cache

Initramfs specification file example

```
dir /dev 755 0 0
nod /dev/console 644 0 0 c 5 1
nod /dev/loop0 644 0 0 b 7 0
dir /bin 755 1000 1000
slink /bin/sh busybox 777 0 0
file /bin/busybox initramfs/busybox 755 0 0
dir /proc 755 0 0
dir /sys 755 0 0
dir /mnt 755 0 0
file /init initramfs/init.sh 755 0 0
```

No need for root user access!



How to handle compressed cpio archives

Useful when you want to build the kernel with a ready-made cpio archive. Better let the kernel do this for you!

- Extracting:
  ```
  gzip -dc initramfs.img | cpio -id
  ```
- Creating:
  ```
  find <dir> -print -depth | cpio -ov | gzip -c > initramfs.img
  ```

/linuxrc

- 1 of the 2 default init programs
  (if no `init` parameter is given to the kernel)
- Traditionally used in initrds or in simple systems not using `/sbin/init`.
- Is most of the time a shell script, based on a very lightweight shell: `nash` or `busybox sh`
- This script can implement complex tasks: detecting drivers to load, setting up networking, mounting partitions, switching to a new root filesystem...

How to create an initrd

In case you really need an initrd (why?).

```
mkdir /mnt/initrd
dd if=/dev/zero of=initrd.img bs=1k count=2048
mkfs.ext2 -F initrd.img
mount -o loop initrd.img /mnt/initrd
```

Fill the ramdisk contents: busybox, modules, `/linuxrc` script

More details in the Free Software tools for embedded systems training!

```
umount /mnt/initrd
gzip --best -c initrd.img > initrd
```

More details on `Documentation/initrd.txt` in the kernel sources! Also explains pivot rooting.

First user-space program

- Specified by the `init` kernel command line parameter
- Executed at the end of booting by the kernel
- Takes care of starting all other user-space programs (system services and user programs).
- Gets the `1` process number (pid)
  Parent or ancestor of all user-space programs
  The system won't let you kill it.
- Only other user-space program called by the kernel: `/sbin/hotplug`

init – the first process

- The first process the kernel will start is init
  - By default it is searched in /sbin/init
  - Can be overridden by the kernel parameter “init=”.
- Init has three roles:
  - To setup the system configuration and start applications.
  - To shut down the system applications.
  - The serve as the parent process of all child processes whose parent has exited.
- The default init implementation reads it's instructions from the file /etc/inittab

inittab

This is an Busybox style inittab (for an example for Sys V inittab see the Appendix.)

Format:

id: runlevel : action : command

id: To which device should std input/output go (empty means the console)

runlevel: ignored. For compatibility with Sys V init.

action: one of sysinit, respawn, askfirst, wait and once

command: shell command to execute

```
# Startup the system
::sysinit:/bin/mount -o remount,rw /
::sysinit:/bin/mount -t proc proc /proc
::sysinit:/bin/mount -a
::sysinit:/sbin/ifconfig lo 127.0.0.1 up

# Put a getty on the serial port
::respawn:/sbin/getty -L ttyS1 115200 vt100

# Start system loggers
null::respawn:/sbin/syslogd -n -m 0
null::respawn:/sbin/klogd -n

# Stuff to do before rebooting
null::shutdown:/usr/bin/killall klogd
null::shutdown:/usr/bin/killall syslogd
null::shutdown:/bin/umount -a -r
```

Compiling and booting Linux

Linux device files

Character device files

- Accessed through a sequential flow of individual characters
- Character devices can be identified by their `c` type (`ls -l`):

```
crw-rw---- 1 root uucp   4,  64 Feb 23 2004 /dev/ttyS0
crw--w---- 1 jdoe tty  136,   1 Feb 23 2004 /dev/pts/1
crw------- 1 root root  13,  32 Feb 23 2004 /dev/input/mouse0
crw-rw-rw- 1 root root   1,   3 Feb 23 2004 /dev/null
```

- Example devices: keyboards, mice, parallel port, IrDA, Bluetooth port, consoles, terminals, sound, video...

Block device files

- Accessed through data blocks of a given size. Blocks can be accessed in any order.
- Block devices can be identified by their `b` type (`ls -l`):

```
brw-rw----   1 root disk     3,   1 Feb 23  2004 hda1
brw-rw----   1 jdoe floppy   2,   0 Feb 23  2004 fd0
brw-rw----   1 root disk     7,   0 Feb 23  2004 loop0
brw-rw----   1 root disk     1,   1 Feb 23  2004 ram1
brw-------   1 root root     8,   1 Feb 23  2004 sda1
```

- Example devices: hard or floppy disks, ram disks, loop devices...

Device major and minor numbers

As you could see in the previous examples,
device files have 2 numbers associated to them:

- First number: *major* number
- Second number: *minor* number
- Major and minor numbers are used by the kernel to bind a driver to the device file. Device file names don't matter to the kernel!
- To find out which driver a device file corresponds to,
  or when the device name is too cryptic,
  see `Documentation/devices.txt`.

Device file creation

- Device files are not created when a driver is loaded.
- They have to be created in advance:
  `mknod /dev/<device> [c|b] <major> <minor>`
- Examples:
  `mknod /dev/ttyS0 c 4 64`
  `mknod /dev/hda1 b 3 1`

Drivers without device files

They don't have any corresponding `/dev` entry you could read or write through a regular Unix command.

- Network drivers

  They are represented by a network device such as `ppp0`, `eth1`, `usbnet`, `irda0` (listed by `ifconfig -a`)

- Other drivers

  Often intermediate or lowlevel drivers just interfacing with other ones. Example: `usbcore`.

GNU / Linux workstation

How to find existing Free Software

Freshmeat.net

http://freshmeat.net/

- The number one site for Free Software searches
- Most free software tools advertised there
- Easy keyword search
- Search by relevance, vitality, popularity, last update...
- Useful details: license, home page, downloads, screenshot
- Possible to subscribe to new releases

Other ways to find existing free software

- SourceForge: http://sourceforge.net/
  Lots of projects hosted there
- FSF / UNESCO Free Software Directory:
  http://directory.fsf.org/
  More than 4300 projects indexed (Nov 2005)
- Internet search engines (of course!)

Tools for the target device

Busybox

General purpose toolbox: busybox

http://www.busybox.net/ from Codepoet Consulting

- Most Unix command line utilities within a single executable! Even includes a web server!
- Sizes less than < 500 KB (statically compiled with uClibc) or less than 1 MB (statically compiled with glibc)
- Easy to configure which features to include
- The best choice for
  - Initrds with complex scripts
  - Small embedded systems with little RAM or ROM.

Busybox commands!

addgroup, adduser, adjtimex, ar, arping, ash, awk, basename, bunzip2, bzcat, cal, cat, chgrp, chmod, chown, chroot, chvt, clear, cmp, cp, cpio, crond, crontab, cut, date, dc, dd, deallocvt, delgroup, deluser, devfsd, df, dirname, dmesg, dos2unix, dpkg, dpkg-deb, du, dumpkmap, dumpleases, echo, egrep, env, expr, false, fbset, fdflush, fdformat, fdisk, fgrep, find, fold, free, freeramdisk, fsck.minix, ftpget, ftpput, getopt, getty, grep, gunzip, gzip, halt, hdparm, head, hexdump, hostid, hostname, **httpd**, hush, hwclock, id, ifconfig, ifdown, ifup, inetd, init, insmod, install, ip, ipaddr, ipcalc, iplink, iproute, iptunnel, kill, killall, klogd, lash, last, length, linuxrc, ln, loadfont, loadkmap, logger, login, logname, logread, losetup, ls, lsmod, makedevs, md5sum, mesg, mkdir, mkfifo, mkfs.minix, mknod, mkswap, mktemp, modprobe, more, mount, msh, mt, mv, nameif, nc, netstat, nslookup, od, openvt, passwd, patch, pidof, ping, ping6, pipe_progress, pivot_root, poweroff, printf, ps, pwd, rdate, readlink, realpath, reboot, renice, reset, rm, rmdir, rmmod, route, rpm, rpm2cpio, run-parts, rx, sed, seq, setkeycodes, sha1sum, sleep, sort, start-stop-daemon, strings, stty, su, sulogin, swapoff, swapon, sync, sysctl, syslogd, tail, tar, tee, telnet, telnetd, test, tftp, time, top, touch, tr, traceroute, true, tty, udhcpc, udhcpd, umount, uname, uncompress, uniq, unix2dos, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, vlock, watch, watchdog, wc, wget, which, who, whoami, xargs, yes, zcat

Compiling busybox

- Get the latest stable sources from http://busybox.net
- Configure busybox (creates a `.config` file):
  ```
  make menuconfig
  ```
- Compile it:
  ```
  make
  ```
- Install it:
  ```
  make install
  ```
  Logged as root, mount the root fs image (e.g. `/mnt/rootfs`)
  ```
  rsync -a _install/ /mnt/rootfs/
  ```
  (the `/` characters are needed!)

Tools for the target device

http and ssh servers

ssh server and client: dropbear

http://matt.ucc.asn.au/dropbear/dropbear.html

- Very small memory footprint ssh server for embedded systems, compared to OpenSSH.
- Satisfies most needs. Both client and server!
- Useful to:
  - Get a remote console on the target device
  - Copy files to and from the target device
  - `rsync` files with the target device

Benefits of a web server interface

Many network enabled devices can just have a network interface

- Examples: modems / routers, IP cameras, printers...
- No need to develop drivers and applications for computers connected to the device. No need to support multiple operating systems!
- Just need to develop static or dynamic HTML pages
  (possibly with powerful client-side JavaScript).
  Easy way of providing access to device information and parameters.
- Reduced hardware costs (no LCD, very little storage space needed)

thttpd

Tiny/Turbo/Throttling HTTP server
http://acme.com/software/thttpd/

- Simple
  Implements the HTTP/1.1 minimum (or just a little more)
  Simple to configure and run.
- Small
  Executable size: 88K (version 2.25b), Apache 2.0.52: 264K
  Very low memory consumption: does not fork and very careful about memory consumption.
- Portable
  Compiles cleanly on most Unix-like operating systems
- Fast
  About as fast as full-featured servers. Much faster on very high loads (because reduces the server load for the same amount of work)
- Secure
  Designed to protect the webserver machine from attacks.

Other web servers (1)

- Busybox http server: http://busybox.net
  Tiny: only adds 12 K to Busybox 1.01 (compiled statically with uClibc)!
  Sufficient features for many devices with a web interface,
  including CGI and http authentication support.
  License: GPL
- KLone: http://koanlogic.com/kl/cont/gb/html/klone.html
  Lightweight but full featured web server for embedded systems.
  Can enclose dynamic (written in C/C++ <% code %>) and compressed content all in an executable of an approximate size of 150 KB.
  License: Dual GPL / Commercial.
  See also http://linuxdevices.com/news/NS8234701895.html

Other web servers (2)

- Boa: http://www.boa.org/

  Designed to be simple, fast and secure.

  Unlike thttpd, no particular care for memory or disk footprint though.

  Embedded systems: pretty popular, though not targeted by developers.

- lighthttpd: http://lighttpd.net

  Low footprint server good at managing high loads.

  May be useful in embedded systems too.

getty or hooking to the serial line

- The getty program opens a serial device, configures it appropriately, optionally configures a modem, and waits for a connection to be made.
  - An active connection on a serial device is usually indicated by the Data Carrier Detect (DCD) pin on the serial device being raised, but getty can also work on NULL modem cable lines.
- When a connection is detected, the getty program defaults to issuing a login: prompt, and invoking the login program to handle the actual system login.

getty continued

- We can also use getty to attach any program that reads/writes the standard input/output to the serial port:

```
getty -i -L -n -l /bin/my_program
```

- This way our program does not need to do anything to handle the serial line correctly, and the getty program does it for us.
- There are many getty implementations, the above on assumes that Busbox getty is used.

telnetd or hooking to a network port

- telnetd is a telnet daemon program implementing the telnet protocol.
- By default, telnet responds to connection by running the normal login program.
- Just like getty, we can use it to hook any program that reads/writes standard input and output to the telnet port:

```
telnetd -l /bin/my_program
```

- In fact, we could use the very same program for both telnet and serial!

Free Software tools for embedded systems

Tools for the target device

Precompiled packages, distributions

Distributions

- Distribute ready to use root filesystems.
  Can be used by any binary compatible platform.
  Example: running Familiar Linux on a mobile phone!
- Can be reused even on an early development phase.
  No need to create your root filesystem from scratch!
- Easy to upgrade, remove or add applications through binary packages.

Embedded distributions (1)

`arm` little endian:

- Familiar http://familiar.handhelds.org/
  Targets PDAs and webpads (Siemens Simpad...)
- IpkgFind: http://ipkgfind.handhelds.org/
  References lots of arm binary packages for `ipkg` based distributions (such as Familiar / OpenZaurus)
  Search by category, keyword or name.

Embedded distributions (2)

Emdebian: http://emdebian.sourceforge.net

- Supports all embedded platforms
- Leverages standard Debian package descriptions and sources, but adapts them for embedded systems: uClibc usage, reduced package size (no documentation), removed dependencies, configured with less options...
- Packages not available yet, but makes it possible to build packages using Debian utilities (such as `dpkg-cross`)
- Unfortunately, lacking momentum in recent months

Ready-made root filesystems

Very useful to start porting work. Contain C library, and minimum apps.

uClibc

- http://www.uclibc.org/toolchains.html (`i386`, `arm`, `mipsel`)
  Images include a native `gcc` toolchain

ARM

- http://www.arm.com/linux/linux_download.html
  Include kernel image, bootloader, and rootfs (`cramfs` or packages)

Other architectures

- If some readers know more useful links, they are welcome!

Buildroot

http://buildroot.uclibc.org/

- Tool to automatically build a ready-made uClibc rootfs with basic applications and a development toolchain. Useful in early development.
- Can be used to build only uClibc toolchains.
- Supports lots of architectures
- Automatically downloads source packages
- Very easy to use: just run:

```
make
```

About Buildroot

- Buildroot is a tool that allow to easily generate both a cross-compilation toolchain and a root file system for your target.
- The cross-compilation toolchain uses uClibc.
- Buildroot automates the process through the use of Makefiles, and has a collection of patches for each gcc and binutils version to make them work on most architectures.

Obtaining Buildroot

- Buildroot is available as daily SVN snapshots or directly using SVN.
- The latest snapshot is always available at http://buildroot.uclibc.org/downloads/snapshots/buildroot-snapshot.tar.bz2
- To download Buildroot using SVN use:
  - `$ svn co svn://uclibc.org/trunk/buildroot`

Using Buildroot

- Buildroot has a nice configuration tool similar to the one you can find in the Linux Kernel.
- Note that you can build everything as a normal user. There is no need to be root to configure and use Buildroot.
- The first step is to run the configuration assistant:
  - $ make menuconfig
- For each entry of the configuration tool, you can find associated help that describes the purpose of the entry.

Using Buildroot

- Once everything is configured, the configuration tool has generated a `.config` file that contains the description of your configuration.
- It will be used by the Makefiles to do what's needed.
- Let's go:
  - $ make
- This command will download, configure and compile all the selected tools, and finally generate a target file system.

Using Buildroot

- The target file system will be named root_fs_ARCH.EXT where ARCH is your architecture and EXT depends on the type of target file system selected in the Target options section of the configuration tool.
- If you intend to do an offline-build and just want to download all sources that you previously selected in "make menuconfig" then issue:
- `$ make source`
- You can now disconnect or copy the content of your dl directory to the build-host.

Environment variables

- Buildroot optionally honors some environment variables that are passed to make :
  - `UCLIBC_CONFIG_FILE=<path/to/.config>`
  - `BUSYBOX_CONFIG_FILE=<path/to/.config>`
- An example that uses config files located in the toplevel directory and in your $HOME:
  - `$ make UCLIBC_CONFIG_FILE=uClibc.config BUSYBOX_CONFIG_FILE=$HOME/bb.config`

Customizing the target file system

- There are two ways to customize the resulting target file system:
  - Customize the target filesystem directly, and rebuild the image.
  - The target file system is available under build_ARCH/root/ where ARCH is the chosen target architecture.
  - You can simply make your changes here, and run make afterwards, which will rebuild the target file system image.
  - Note that if you decide to completely rebuild your tool chain and tools, these changes will be lost.

Customizing the target file system

- The second method is:
  - Customize the target file system skeleton, available under target/generic/target_skeleton/.
  - You can customize configuration files or other stuff here.
  - Note that the full file hierarchy is not yet present, because it's created during the compilation process. So you can't do everything on this target file system skeleton.
  - However, changes to it remain even if you completely rebuild the cross-compilation toolchain and the tools

Customizing the target file system

- You can also customize the target/generic/device_table.txt file which is used by the tools that generate the target file system image to properly set permissions and create device nodes.
- The target/generic/skel.tar.gz file contains the main directories of a root file system and there is no obvious reason for which it should be changed.
- These customizations are deployed into build_ARCH/root/ just before the actual image is made. So simply rebuilding the image by running make should propogate any new changes to the image.

Customizing the Busybox config

- Busybox is very configurable, and you may want to customize it. You can follow these simple steps to do it. It's not an optimal way, but it's simple and it works.
  - Make a first compilation of buildroot with busybox without trying to customize it.
  - Invoke make busybox-menuconfig.
  - The nice configuration tool appears and you can customize everything.
  - Run the compilation of buildroot again.

Customizing the Busybox config

- Otherwise, you can simply change the package/busybox/busybox-<version>.config file if you know the options you want to change without using the configuration tool.
- If you want to use an existing config file for busybox, then see section environment variables.

Customizing the uClibc configuration

- The easiest way to modify the configuration of uClibc is to follow these steps :
  - Make a first compilation of buildroot without trying to customize uClibc.
  - Invoke make uclibc-menuconfig. The configuration assistant appears. Make your configuration as appropriate.
  - Copy the .config file to toolchain/uClibc/uClibc.config or toolchain/uClibc/uClibc.config-locale.
    - The former is used if you haven't selected locale support in Buildroot configuration, and the latter is used if you have selected locale support.
  - Run the compilation of Buildroot again

Customizing the uClibc configuration

- Otherwise, you can simply change toolchain/uClibc/uClibc.config or toolchain/uClibc/uClibc.config-locale without running the configuration assistant.
- If you want to use an existing config file for uclibc, then see section environment variables.

How Buildroot works

- Buildroot is a set of Makefiles that download, configure and compiles software with the correct options.
- It also includes some patches for various software, mainly the ones involved in the cross-compilation tool chain (gcc, binutils and uClibc).
- There is one Makefile per software, and they are named with the .mk extension.

How Buildroot works

- Makefiles are located into three sections:
  - **package** contains the files for all programs to add to the target root filesystem. There is one sub-directory per tool.
  - **toolchain** contains the files for the cross-compilation toolchain : binutils, ccache, gcc, gdb, kernel-headers and uClibc.
  - **target** contains the files for software related to the generation of the target root file system image.
    - Four types of file systems are supported : ext2, jffs2, cramfs and squashfs.
    - For each of them, there's a sub-directory with the required files.
    - There is also a default/ directory that contains the skeleton.

How Buildroot works

- Each directory contains at least 2 files :
  - something.mk is the Makefile that downloads, configures, compiles and installs the software something.
  - Config.in is a part of the configuration tool description file. It describes the option related to the current software.

How Buildroot works

- The main Makefile do the job through the following steps (once the configuration is done):
  - Create the download directory (dl/ by default).
    - This is where the tarballs will be downloaded.
  - Create the build directory (build_ARCH/ by default, where ARCH is your architecture).
    - This is where all user-space tools while be compiled.
  - Create the toolchain build directory (toolchain_build_ARCH/ by default, where ARCH is your architecture).
    - This is where the cross compilation toolchain will be compiled.

How Buildroot works

- Setup the staging directory (build_ARCH/staging_dir/ by default).
  - This is where the cross-compilation toolchain will be installed.
- Create the target directory (build_ARCH/root/ by default) and the target filesystem skeleton.
  - This directory will contain the final root filesystem.
  - To setup it up, it first deletes it, then it uncompress the target/generic/skel.tar.gz file to create the main subdirectories and symlinks, copies the skeleton available in target/generic/target_skeleton and then removes useless .svn/ directories.

Using the uClibc tool chain

- You may want to compile your own programs or other software that are not packaged in Buildroot.
- In order to do this, you can use the toolchain that was generated by Buildroot.
- The toolchain generated by Buildroot by default is located in build_ARCH/staging_dir/.
- The simplest way to use it is to add build_ARCH/staging_dir/bin/ to your PATH environnement variable, and then to use arch-linux-gcc, arch-linux-objdump, arch-linux-ld, etc.

Using the uClibc tool chain

- You can also configure Buildroot to generate elsewhere using the option of the configuration tool: `Build options -> Toolchain and header file location`
- For example:
  - Add the following to your .bashrc
    - `export PATH="$PATH:~/buildroot/build_mips/staging_dir/bin/"`
  - Then you can use :
    - `mips-linux-gcc -o foo foo.c`

Location of downloaded packages

- It might be useful to know that the various tarballs that are downloaded by the Makefiles are all stored in the DL_DIR which by default is the dl directory.
- It's useful for example if you want to keep a complete version of Buildroot which is know to be working with the associated tarballs.
- This will allow you to regenerate the tool chain and the target file system with exactly the same versions.

Extending Buildroot

- First of all, create a directory under the `package` directory for your software, for example foo.
- In it, create a file named Config.in.
- This file will contain the portion of options description related to our foo software that will be used and displayed in the configuration tool.

Extending Buildroot

- Here is an example :

```
config BR2_PACKAGE_FOO
        bool "foo"
        default n
        help
    This is a comment that explains what foo is.
    http://foosoftware.org/foo/
```

- Of course, you can add other options to configure particular things in your software.

Extending Buildroot

- Finally, here's the hardest part.
- Create a file named foo.mk.
- It will contain the Makefile rules that are in charge of downloading, configuring, compiling and installing the software.
- As you can see, adding a software to buildroot is simply a matter of writing a Makefile using an already existing example and to modify it according to the compilation process of the software.
- If you package software that might be useful for other persons, don't forget to send a patch to Buildroot developers !

Driver development

New Device Model

Device Model features (1)

- Originally created to make power management simpler
  Now goes much beyond.
- Used to represent the architecture and state of the system
- Has a representation in userspace: sysfs
  Now the preferred interface with userspace (instead of `/proc`)
- Easy to implement thanks to the device interface: `include/linux/device.h`

Device model features (2)

Allows to view the system for several points of view:

- From devices existing in the system: their power state, the bus they are attached to, and the driver responsible for them.
- From the system bus structure: which bus is connected to which bus (e.g. USB bus controller on the PCI bus), existing devices and devices potentially accepted (with their drivers)
- From available device drivers: which devices they can support, and which bus type they know about.
- From the various kinds ("classes") of devices: `input`, `net`, `sound`... Existing devices for each class. Convenient to find all the input devices without actually knowing how they are physically connected.

sysfs

- Userspace representation of the Device Model.
- Configure it with
  `CONFIG_SYSFS=y` (Filesystems -> Pseudo filesystems)
- Mount it with
  `mount -t sysfs none /sys`
- Spend time exploring `/sys` on your workstation!

sysfs tools

http://linux-diag.sourceforge.net/Sysfsutils.html

- `libsysfs` - The library's purpose is to provide a consistent and stable interface for querying system device information exposed through sysfs. Used by `udev` (see later)
- `systool` - A utility built upon `libsysfs` that lists devices by bus, class, and topology.

Device Model references

- Very useful and clear documentation in the kernel sources!
- `Documentation/driver-model/`
- `Documentation/filesystems/sysfs.txt`

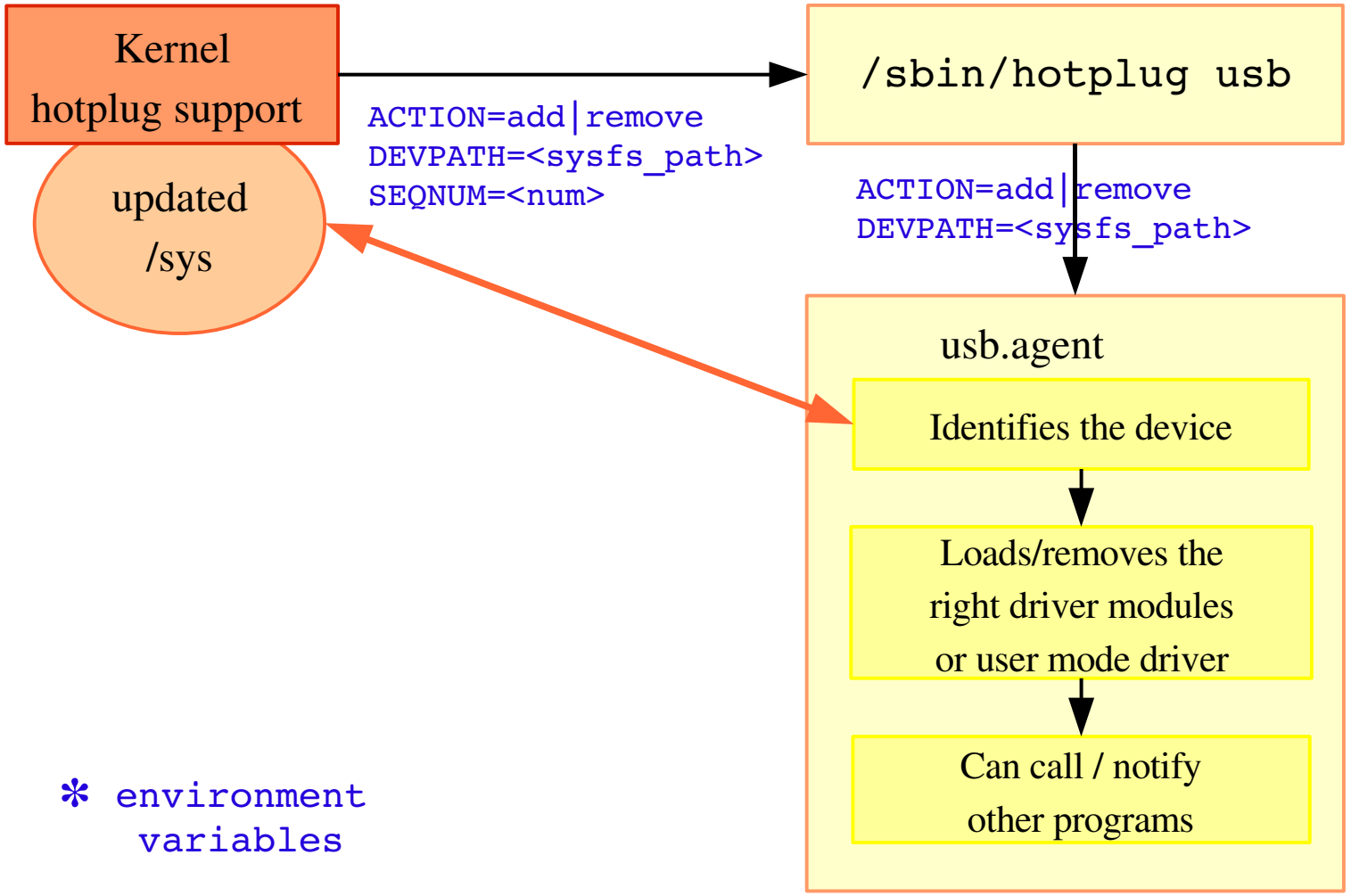
Driver development

hotplug

hotplug overview

- Introduced in Linux 2.4. Pioneered by USB.
- Kernel mechanism to notify user space programs that a device has been inserted or removed.
- User space scripts then take care of identifying the hardware and inserting/removing the right driver modules.
- Linux 2.6: much easier device identification thanks to sysfs
- Makes it possible to load external firmware
- Makes it possible to have user-mode only driver (e.g. `libsane`)
- Kernel configuration:
  `CONFIG_HOTPLUG=y` (General setup section)

hotplug flow example



hotplug files

`/lib/modules/*/modules.*map`
: `depmod` output

`/proc/sys/kernel/hotplug`
: specifies hotplug program path

`/sbin/hotplug`
: hotplug program (default path name)

`/etc/hotplug/*`
: hotplug files

`/etc/hotplug/NAME*`
: subsystem-specific files, for agents

`/etc/hotplug/NAME/DRIVER`
: driver setup scripts, invoked by agents

`/etc/hotplug/usb/DRIVER.usermap`
: `depmod` data for user-mode drivers

`/etc/hotplug/NAME.agent`
: hotplug subsystem-specific agents

Firmware hotplugging

Reasons for keeping firmware data outside their device drivers

Legal issues

- Some firmware is not legal to distribute and can't be shipped in a Free Software driver
- Some firmware may not be considered as free enough to distribute (Debian example)

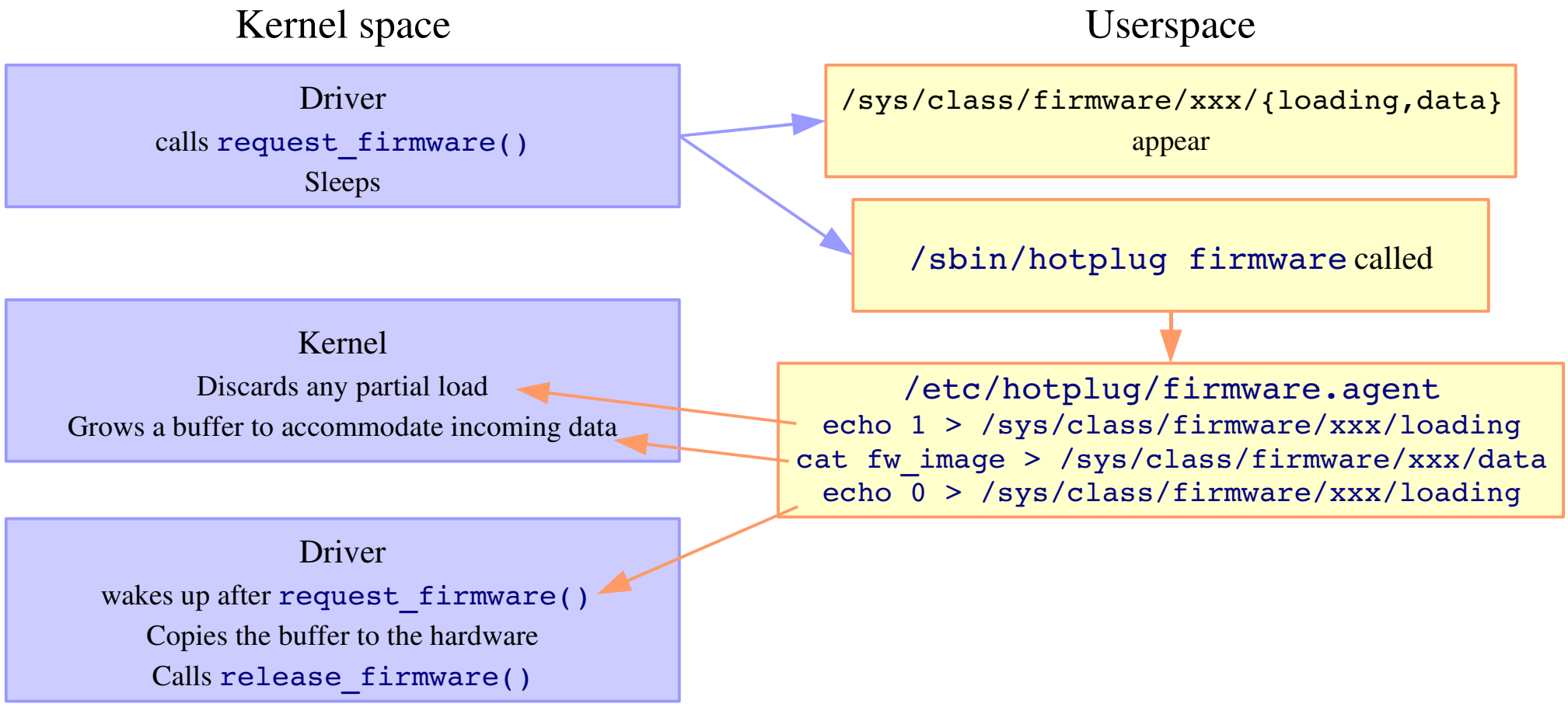
Technical issues

- Firmware in kernel code would occupy memory permanently, even if just used once.

Firmware hotplugging setup

- Kernel configuration: needs to be set in `CONFIG_FW_LOADER` (Device Drivers -> Generic Driver Options -> hotplug firmware loading support)
- Need `/sys` to be mounted
- Location of firmware files: check `/etc/hotplug/firmware.agent`

Firmware hotplugging implementation



See `Documentation/firmware_class/` for a nice overview

hotplug references

- Project page and documentation
  http://linux-hotplug.sourceforge.net/
- Mailing list:
  http://lists.sourceforge.net/lists/listinfo/linux-hotplug-devel

Driver development

udev: user-space device file management

/dev issues and limitations

- On Red Hat 9, 18000 entries in `/dev`!
  All entries for all possible devices need to be created at system installation.
- Need for an authority to assign major numbers
  http://lanana.org/: Linux Assigned Names and Numbers Authority
- Not enough numbers in 2.4, limits extended in 2.6
- Userspace doesn't know what devices are present in the system.
- Userspace can't tell which `/dev` entry is which device

devfs solutions and limitations

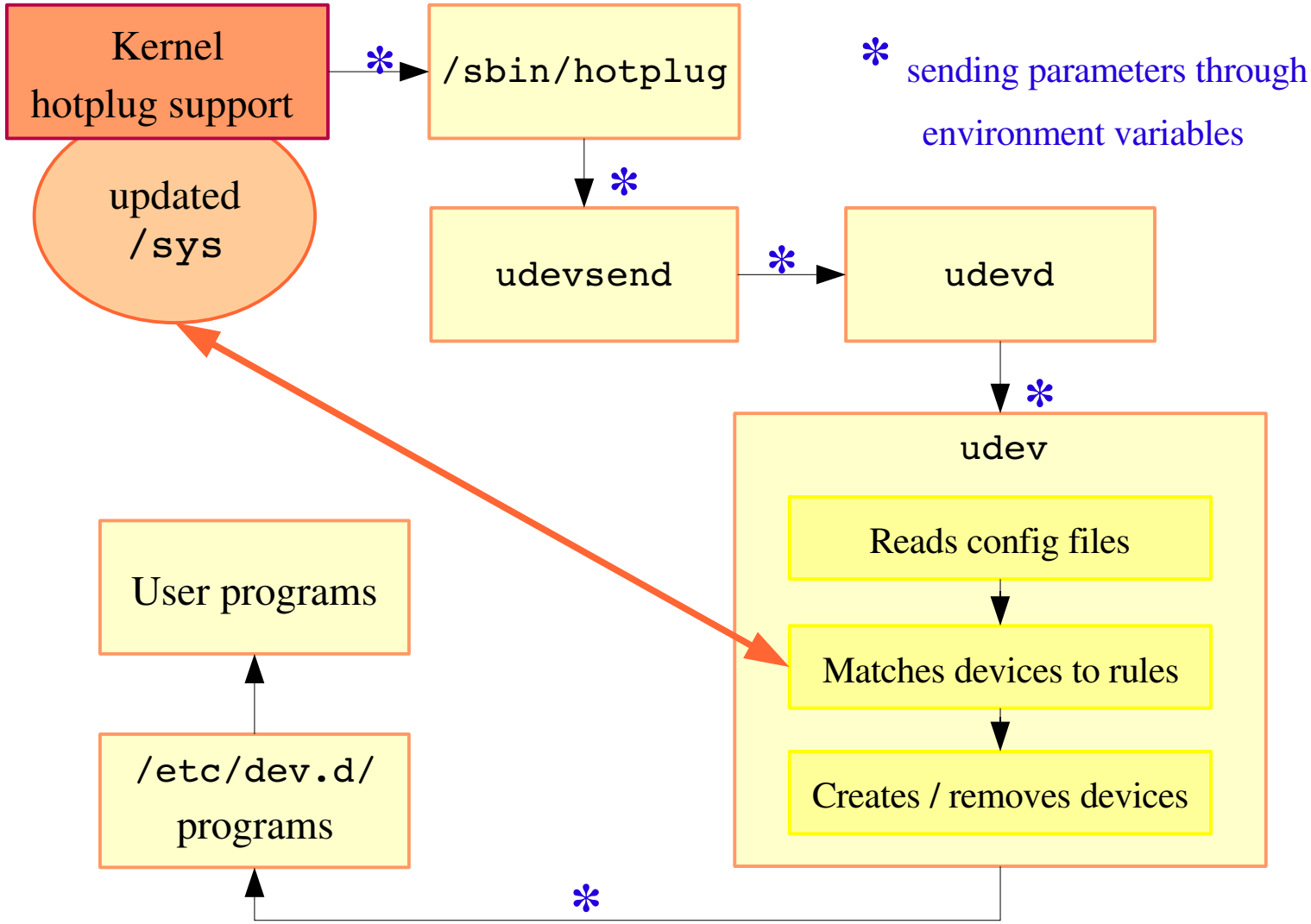
- Only shows present devices
- But uses different names as in `/dev`, causing issues in scripts.
- But no flexibility in device names, unlike with `/dev/`, e.g. the 1st IDE disk device has to be called either `/dev/hda` or `/dev/ide/hd/c0b0t0u0`.
- But doesn't allow dynamic major and minor number allocation.
- But requires to store the device naming policy in kernel memory. Can't be swapped out!

The udev solution

Takes advantage of both hotplug and sysfs

- Entirely in user space
- Automatically creates device entries (by default in `/udev`)
- Called by `/sbin/hotplug`, uses information from sysfs.
- Major and minor device numbers found in sysfs
- Requires no change to the driver code
- Small size

How udev works



udev toolset (1)

Major components

- `udevsend` (8KB in Fedora Core 3)
  Takes care of handling the `/sbin/hotplug` events, and sending them to `udevd`
- `udevd` (12KB)
  Takes care of reordering hotplug events, before calling `udev` instances for each of them.
- `udev` (68KB)
  Takes care of creating or removing device entries, entry naming, and then executing programs in `/etc/dev.d/`

udev toolset (2)

Other utilities

- `udevinfo` (48KB)
  Lets users query the udev database
- `udevstart` (functionality brought by `udev`)
  Populates the initial device directory from valid devices found in the sysfs device tree.
- `udevtest <sysfs_device_path>` (64KB)
  Simulates a `udev` run to test the configured rules

udev configuration file

`/etc/udev/udev.conf`

Easy to edit and configure. Sets the below parameters:

- Device directory (`/udev`)
- udev database file (`/dev/.udev.tdb`)
- udev rules (`/etc/udev/rules.d/`)
  udev permissions (`/etc/udev/permissions.d/`)
- default mode (`0600`), default owner (`root`) and group (`root`), when not found in udev's permissions.
- Enable logging (`yes`)
  Debug messages available in `/var/log/messages`

udev naming capabilities

Device names can be defined

- from a label or serial number
- from a bus device number
- from a location on the bus topology
- from a kernel name

udev can also create device links

udev rules file example

```
# if /sbin/scsi_id returns "OEM 0815" device will be called disk1
BUS="scsi", PROGRAM="/sbin/scsi_id", RESULT="OEM 0815", NAME="disk1"

# USB printer to be called lp_color
BUS="usb", SYSFS{serial}="W09090207101241330", NAME="lp_color"

# SCSI disk with a specific vendor and model number will be called boot
BUS="scsi", SYSFS{vendor}="IBM", SYSFS{model}="ST336", NAME="boot%n"

# sound card with PCI bus id 00:0b.0 to be called dsp
BUS="pci", ID="00:0b.0", NAME="dsp"

# USB mouse at third port of the second hub to be called mouse1
BUS="usb", PLACE="2.3", NAME="mouse1"

# ttyUSB1 should always be called pda with two additional symlinks
KERNEL="ttyUSB1", NAME="pda", SYMLINK="palmtop handheld"

# multiple USB webcams with symlinks to be called webcam0, webcam1, ...
BUS="usb", SYSFS{model}="XV3", NAME="video%n", SYMLINK="webcam%n"
```

udev sample permissions

Sample udev permission file (in `/etc/udev/permissions.d/`):

```
#name:user:group:mode
input/*:root:root:644
ttyUSB1:0:8:0660
video*:root:video:0660
dsp1:::0666
```

/etc/dev.d/

After device nodes are created, removed or renamed, udev can call programs found in the below search order:

- `/etc/dev.d/$(DEVNAME)/*.dev`
- `/etc/dev.d/$(SUBSYSTEM)/*.dev`
- `/etc/dev.d/default/*.dev`

The programs in each directory are sorted in lexical order.

This is useful to notify user applications of device changes.

udev links

- Home page
  http://kernel.org/pub/linux/utils/kernel/hotplug/udev.html
- Sources
  http://kernel.org/pub/linux/utils/kernel/hotplug/
- Mailing list:
  linux-hotplug-devel@lists.sourceforge.net
- Greg Kroah-Hartman, udev presentation
  http://www.kroah.com/linux/talks/oscon_2004_udev/
- Greg Kroah-Hartman, udev whitepaper
  http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf

Advice and resources

System security

- In production: disable loadable kernel modules if you can.
- Carefully check data from input devices (if interpreted by the driver) and from user programs (buffer overflows)
- Check kernel sources signature.
- Beware of uninitialized memory.
  Sensitive memory: clear it before freeing it.
  The same page could later be allocated to a user process.
- Compile modules by yourself (beware of binary modules)

Advice and resources

Choosing filesystems

Block device or MTD filesystems

Block devices

- Floppy or hard disks (SCSI, IDE)
- Compact Flash (seen as a regular IDE drive)
- RAM disks
- Loopback devices

Memory Technology Devices (MTD)

- Flash, ROM or RAM chips
- MTD emulation on block devices

Filesystems are either made for block or MTD storage devices.
See `Documentation/filesystems/` for details.

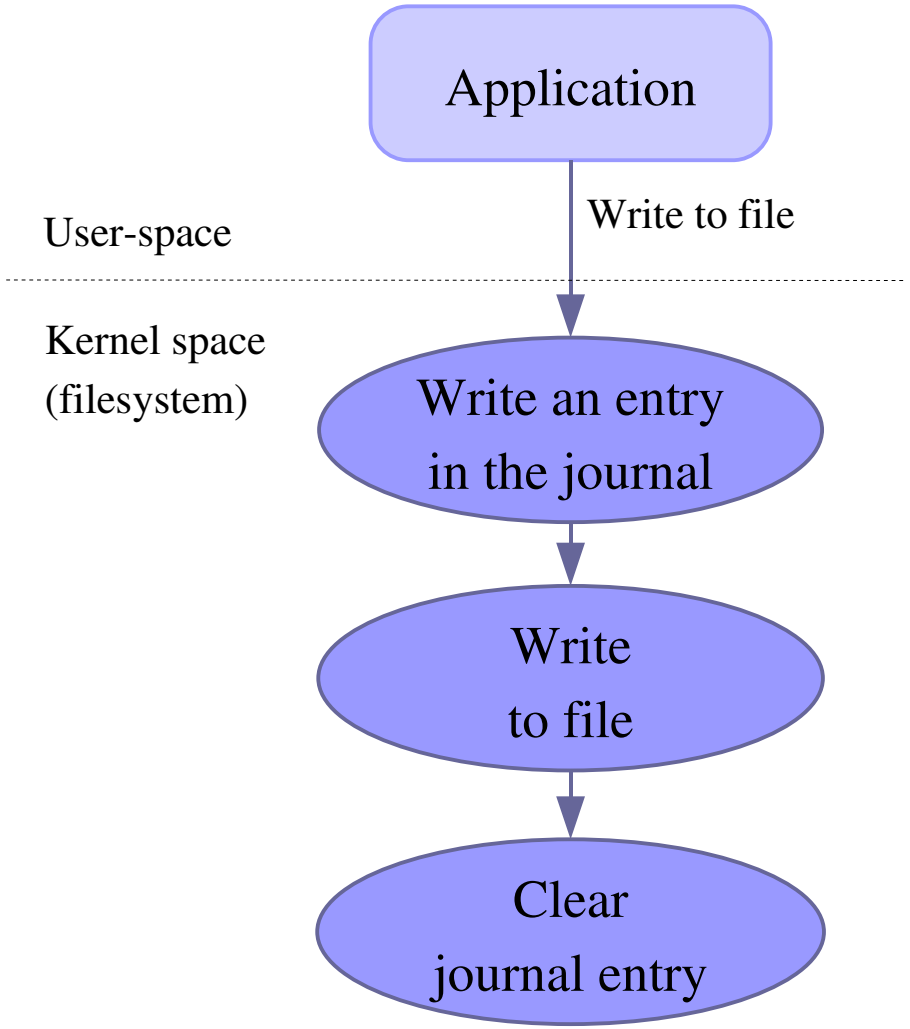
Traditional block filesystems

Traditional filesystems

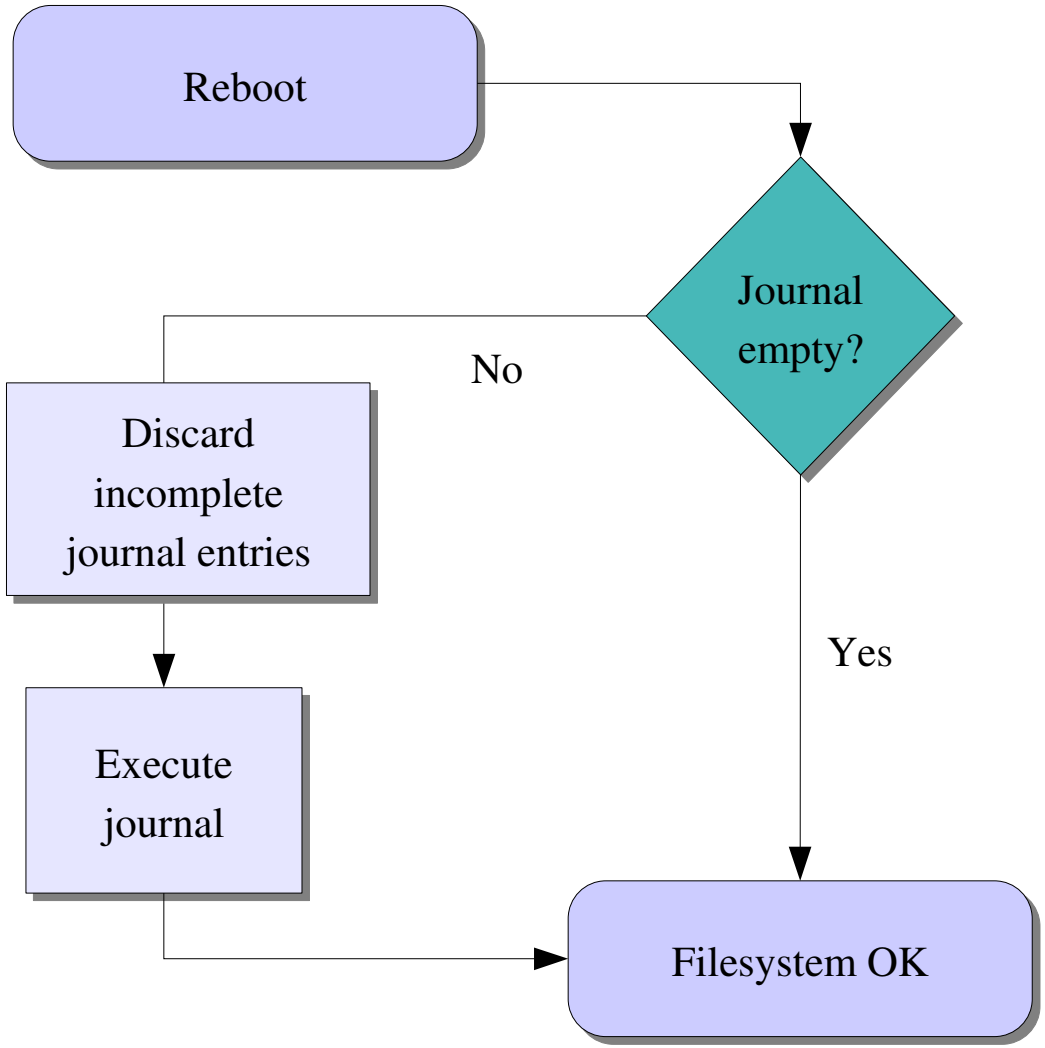
- Hard to recover from crashes. Can be left in a corrupted ("half finished") state after a system crash or sudden power-off.
- `ext2`: traditional Linux filesystem
  (repair it with `fsck.ext2`)
- `vfat`: traditional Windows filesystem
  (repair it with `fsck.vfat` on GNU/Linux or `Scandisk` on Windows)

Journaled filesystems

- Designed to stay in a correct state even after system crashes or a sudden power-off
- All writes are first described in the journal before being committed to files



Filesystem recovery after crashes



- Thanks to the journal, the filesystem is never left in a corrupted state
- Recently saved data could still be lost

Journaled block filesystems

Journaled filesystems

- `ext3`: `ext2` with journal extension
- `reiserFS`: most innovative (fast and extensible)
- Others: `JFS` (IBM), `XFS` (SGI)
- `NTFS`: well supported by Linux in read-mode

Compressed block filesystems (1)

Cramfs

- Simple, small, read-only compressed filesystem designed for embedded systems .
- Maximum filesystem size: 256 MB
- Maximum file size: 16 MB

See `Documentation/filesystems/cramfs.txt` in kernel sources.

Compressed block filesystems (2)

Squashfs: http://squashfs.sourceforge.net

- A must-use replacement for Cramfs! Also read-only.
- Maximum filesystem and file size: 2^{32} bytes (`4 GB`)
- Achieves better compression and much better performance.
- Fully stable but released as a separate patch so far (waiting for Linux 2.7 to start).
- Successfully tested on `i386`, `ppc`, `arm` and `sparc`.

See benchmarks on
http://tree.celinuxforum.org/CelfPubWiki/SquashFsComparisons

ramdisk filesystems

Useful to store temporary data not kept after power off or reboot: system log files, connection data, temporary files...

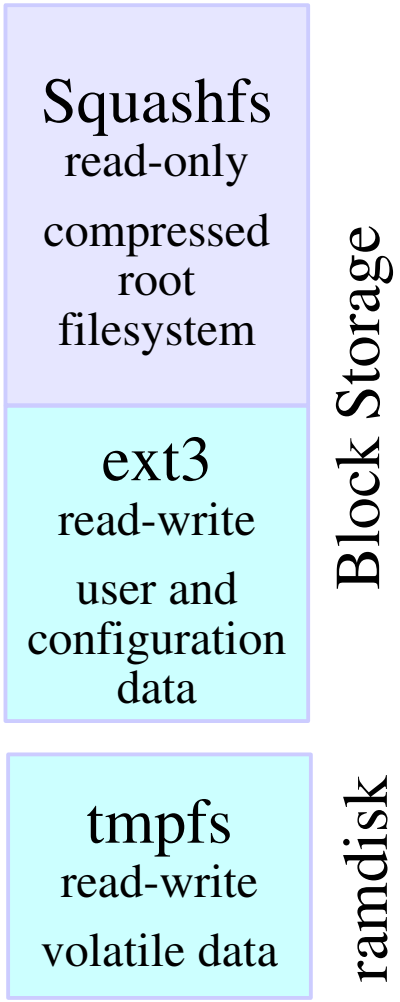
- Traditional block filesystems: journaling not needed.
  Many drawbacks: fixed in size. Remaining space not usable as RAM.
  Files duplicated in RAM (in the block device and file cache)!
- tmpfs (Config: `File systems` -> `Pseudo filesystems`)
  Doesn't waste RAM: grows and shrinks to accommodate stored files
  Saves RAM: no duplication; can swap out pages to disk when needed.

See `Documentation/filesystems/tmpfs.txt` in kernel sources.

Mixing read-only and read-write filesystems

Good idea to split your block storage into

- A compressed read-only partition (Squashfs)
  Typically used for the root filesystem (binaries, kernel...).
  Compression saves space. Read-only access protects your system from mistakes and data corruption.
- A read-write partition with a journaled filesystem (like ext3)
  Used to store user or configuration data.
  Guarantees filesystem integrity after power off or crashes.
- A ramdisk for temporary files (tmpfs)



The MTD subsystem

Linux filesystem interface

MTD "User" modules

jffs2 | Char device | Block device

yaffs2 | Read-only block device

Flash Translation Layers
for block device emulation
Caution: patented algorithms!

FTL | NFTL | INFTL

MTD Chip drivers

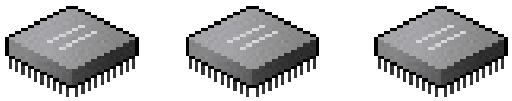
CFI flash | RAM chips

NAND flash | DiskOnChip flash | ROM chips

Block device | Virtual memory

Virtual devices appearing as MTD devices

Memory devices hardware



MTD filesystems - jffs2

`jffs2`: Journaling Flash File System v2

- Designed to write flash sectors in an homogeneous way.
  Flash bits can only be rewritten a relatively small number of times (often < 100 000).
- Compressed to fit as many data as possible on flash chips. Also compensates for slower access time to those chips.
- Power down reliable: can restart without any intervention
- Shortcomings: low speed, big RAM consumption (4 MB for 128 MB of storage).

Mounting a jffs2 image

Useful to create or edit `jffs2` images on your GNU / Linux PC!

- Mounting an MTD device as a loop device is a bit complex task. Here's an example for `jffs2`:

```
modprobe loop
modprobe mtdblock
losetup /dev/loop0 <file>.jffs2
modprobe blkmtd erasesz=256 device=/dev/loop0
mknod /dev/mtdblock0 b 31 0                    (if not done yet)
mkdir /mnt/jffs2                    (example mount point, if not done yet)
mount -t jffs2 /dev/mtdblock0 /mnt/jffs2/
```

- It's very likely that your standard kernel misses one of these modules. Check the corresponding `.c` file in the kernel sources and look in the corresponding `Makefile` which option you need to recompile your kernel with.

MTD filesystems - yaffs2

`yaffs2`: Yet Another Flash Filing System, version 2

- `yaffs2` home: http://aleph1.co.uk/drupal/?q=node/35
  Caution: site under reconstruction. Lots of broken links!
- Features: NAND flash only. No compression. Several times faster than `jffs2` (mainly significant in boot time) Consumes much less RAM. Also includes ECC and is power down reliable.
- License: GPL or proprietary
- Ships outside the Linux kernel. Get it from CVS:
  http://aleph1.co.uk/cgi-bin/viewcvs.cgi/yaffs/

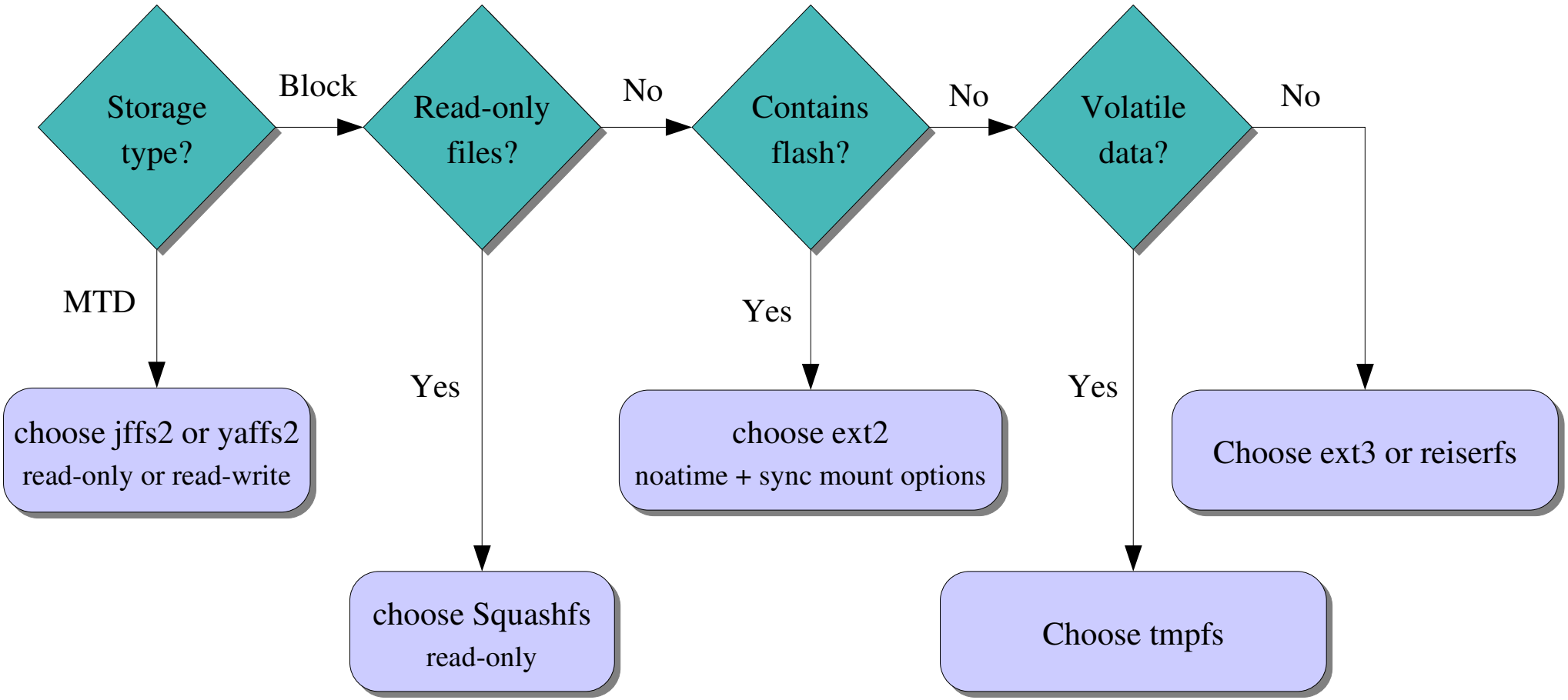
Filesystem choices for block flash devices

Typically for Compact Flash storage

- Can't use `jffs2` or `yaffs2` on CF storage (block device). MTD Block device emulation could be used, but `jffs2` / `yaffs2` writing schemes could interfere with on-chip flash management (manufacturer dependent).
- Never use block device journaled filesystems on unprotected flash chips! Keeping the journal would write the same sectors over and over again and quickly damage them.

- Can use `ext2` or `vfat`, with the below mount options:
  `noatime`: doesn't write access time information in file inodes
  `sync`: to perform writes immediately (reduce power down failure risks)

Filesystem choice summary



Magic SysRq Key

- A 'magical' key combo you can hit which the kernel will respond to. ALT-SysRq-<command key> for PC.
- Can also be triggered via serial port by sending BREAK and then the command code.
- Can also be triggered by writing to /proc/sysrq-trigger:

```
echo t > /proc/sysrq-trigger
```

- To enable in build time turn on MAGIC_SYSRQ in kernel config. To enable in run time write “1” to /proc/sys/kernel/sysrq:

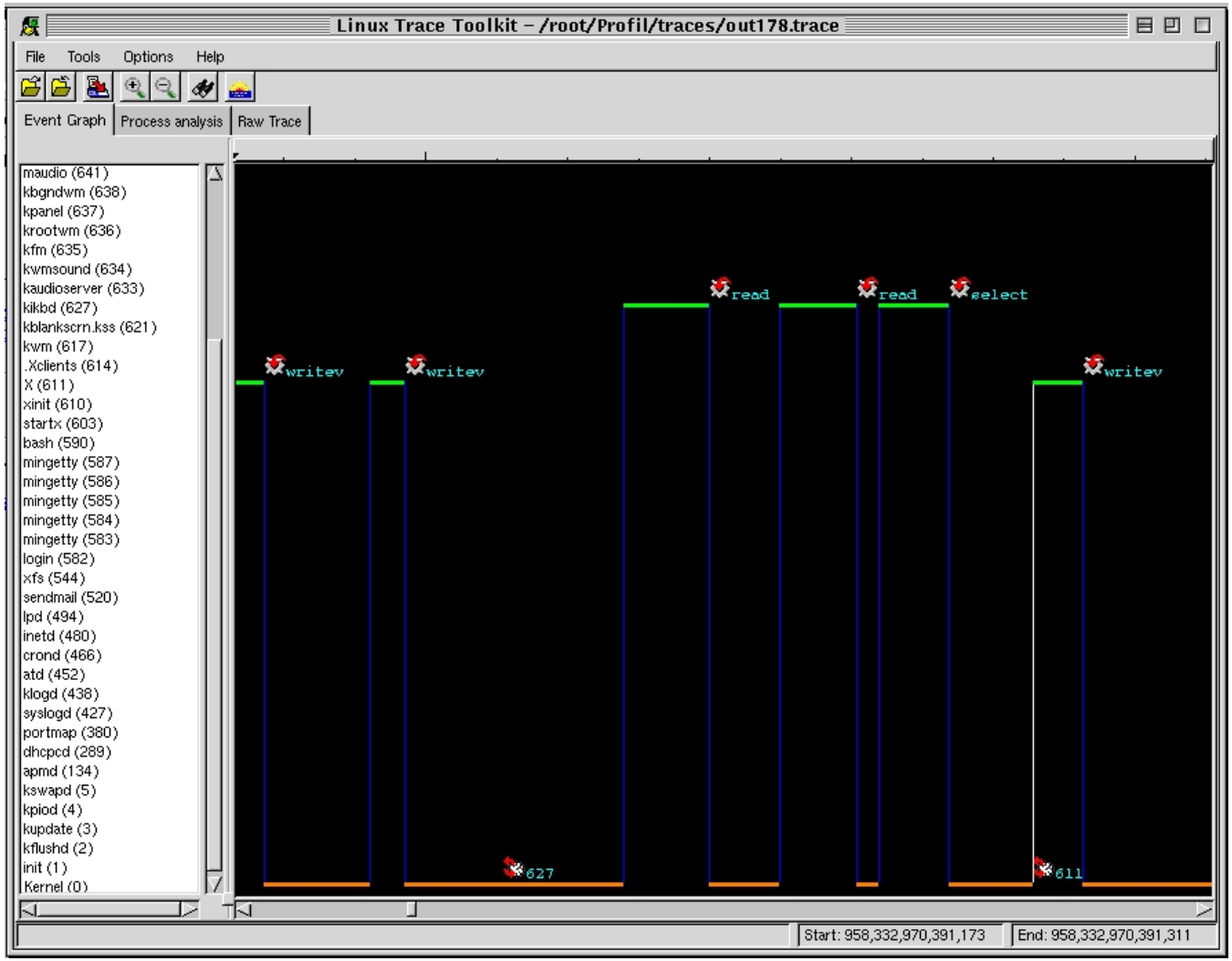
```
echo 1 > /proc/sys/kernel/sysrq
```

Linux Trace Tool Kit

- LTT is a tool that allow tracing of events in a Linux systems
- Events can be interrupts, system call, context switches or custom events.
- LTT is composed from 4 parts:
  - Kernel patch that adds tracing hook to the kernel.
  - Kernel module that uses the hooks to register events.
  - A tracing daemon that drives the module
  - A visual data analyzer.
- Grab it from http://ltt.polymtl.ca/

Linux Trace Tool Kit

Tasks



Events

OProfile

- OProfile is a system-wide profiler for Linux systems, capable of profiling all running code at low overhead.
- It consists of a kernel driver and a daemon for collecting sample data, and post-profiling tools for analysis
- OProfile uses the hardware performance counters of the CPU to enable profiling of a wide variety of interesting statistics, which can also be used for basic time-spent profiling.
- All code is profiled: hardware and software interrupt handlers, kernel modules, the kernel, shared libraries, and applications.

Oprofile Cheat sheet

- Your 4-step plan to profiling.
- Compile and install oprofile, making sure to use --with-linux option to point to your current kernel source, or --with-kernel-support for 2.6 kernels.
- Start up the profiler :

```
# opcontrol –vmlinux=/path/to/vmlinux
# opcontrol --start
```

- Now the profiler is running, go do what ever you want to profile.

Oprofile Cheat sheet

- Now let's generate a profile summary :

```
# opreport -l /path/to/mybinary
```

Or, if we built our binary with -g, we can produce some annotated source :

```
# opannotate --source —output-dir=/output \
  /path/to/mybinary
```

- Or we can look at the rankings of the various system components as a whole :

```
opreport
```

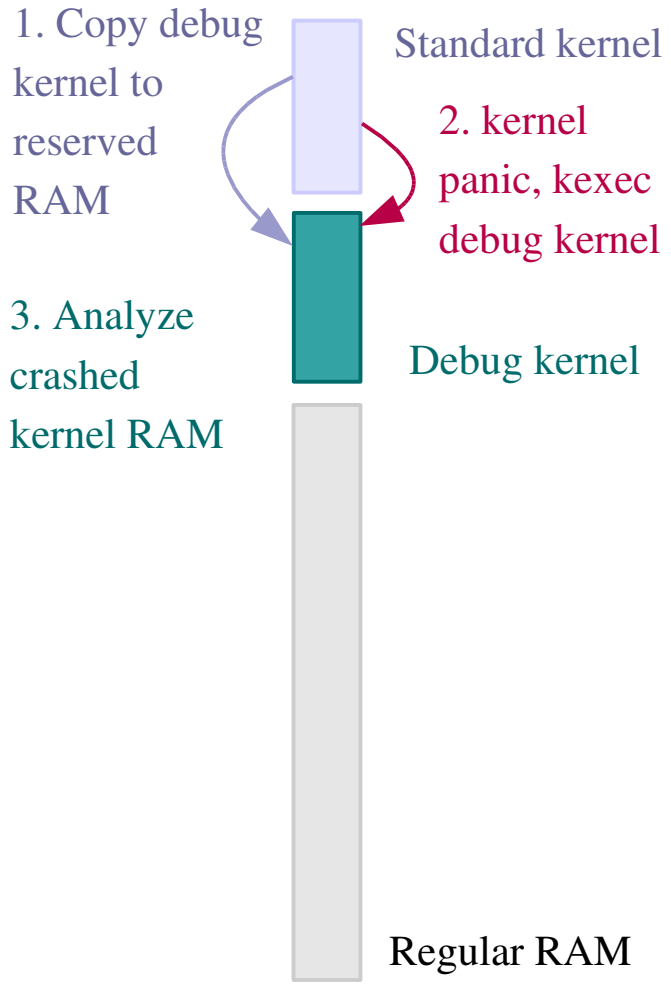
kgdb kernel patch

http://kgdb.linsyssoft.com/

- The execution of the patched kernel is fully controlled by `gdb` from another machine, connected through a serial line.
- Can do almost everything, including inserting breakpoints in interrupt handlers.
- Supported architectures: `i386`, `x86_64`, `ppc` and `s390`.

Kernel crash analysis with kexec

- `kexec` system call: makes it possible to call a new kernel, without rebooting and going through the BIOS / firmware.
- Idea: after a kernel panic, make the kernel automatically execute a new, clean kernel from a reserved location in RAM, to perform post-mortem analysis of the memory of the crashed kernel.
- See Documentation/kdump/kdump.txt in the kernel sources for details.



Advice and resources

Getting help and contributions

Solving issues

- If you face an issue, and it doesn't look specific to your work but rather to the tools you are using, it is very likely that someone else already faced it.
- Search the Internet for similar error reports
  - On web sites or mailing list archives (using a good search engine)
  - On newsgroups: http://groups.google.com/
- You have great chances of finding a solution or workaround, or at least an explanation for your issue.
- Otherwise, reporting the issue is up to you!

Getting help

- If you have a support contract, ask your vendor
- Otherwise, don't hesitate to share your questions and issues on mailing lists
  - Either contact the Linux mailing list for your architecture (like linux-arm-kernel or linuxsh-dev...)
  - Or contact the mailing list for the subsystem you're dealing with (linux-usb-devel, linux-mtd...). Don't ask the maintainer directly!
  - Most mailing lists come with a FAQ page. Make sure you read it before contacting the mailing list
  - Refrain from contacting the Linux Kernel mailing list, unless you're an experienced developer and need advice

Advice and resources

Bug report and patch submission

Reporting Linux bugs

- First make sure you're using the latest version
- Make sure you investigate the issue as much as you can:
  see `Documentation/BUG-HUNTING`
- Make sure the bug has not been reported yet. A bug tracking system (http://bugzilla.kernel.org/) exists but very few kernel developers use it. Best to use web search engines (accessing public mailing list archives)
- If the subsystem you report a bug on has a mailing list, use it. Otherwise, contact the official maintainer (see the `MAINTAINERS` file). Always give as many useful details as possible.

How to submit patches or drivers

- Don't merge patches addressing different issues
- You should identify and contact the official maintainer for the files to patch.
- See `Documentation/SubmittingPatches` for details. For trivial patches, you can copy the Trivial Patch Monkey.
- Special subsystems:
  - ARM platform: it's best to submit your ARM patches to Russell King's patch system: http://www.arm.linux.org.uk/developer/patches/