BUILDING THE 3.X / 4.X / 5.X LINUX KERNEL

Important Notice : Courseware - Legal

This courseware is both the product of the author and of freely available opensource and/or public domain materials. Wherever external material has been shown, it's source and ownership have been clearly attributed. We acknowledge all copyrights and trademarks of the respective owners.

The contents of the **courseware PDFs are considered proprietary** and thus cannot be copied or reproduced in any form whatsoever without the explicit written consent of the author.

Only the programs - **source code** and binaries (where applicable) - that form part of this courseware, and that are made available to the participant, are released under the terms of the permissive **MIT license**.
Under the terms of the MIT License, you can certainly use the source code provided here; you must just attribute the original source (author of this courseware and/or other copyright/trademark holders).

VERY IMPORTANT :: Before using this source(s) in your project(s), you *MUST* check with your organization's legal staff that it is appropriate to do so.

The courseware PDFs are *not* under the MIT License, they are to be kept confidential, non-distributable without consent, for your private internal use only.

The duration, contents, content matter, programs, etc. contained in this courseware and companion participant VM are subject to change at any point in time without prior notice to individual participants.

Care has been taken in the preparation of this material, but there is no warranty, expressed or implied of any kind, and we can assume no responsibility for any errors or omisions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

kaiwanTECH Linux OS Corporate Training Programs
Please do check out our current offering of world-class, seriously-valuable, high on returns, technical Linux OS corporate training programs here: http://bit.ly/ktcorp

Building the 2.6 (onwards) Linux Kernel - Quick Step Summary

1. Download and Extract the kernel source tree
   a) Download the new kernel source; f.e. for the 5.4.1 kernel source tree:
   `wget https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.4.1.tar.xz`
   b) Extract it into some location under your home directory
   `tar xf linux-5.4.1.tar.xz`

(Alternatively, one can always use git(1) to download a particular version)

2. Configuration : select kernel support options as required for the new kernel (*make [x|g|menu]config*);

   `make [ARCH=<arch>] menuconfig`

   is recommended. `ARCH` determines the architecture (cpu) the kernel is being configured and built for, the default is x86; others are:

```
alpha  arm    c6x   h8300    ia64     m68k     mips       nios2         parisc
riscv  sh   um       x86 arc    arm64  csky  hexagon  Kconfig  microblaze  nds32
openrisc  powerpc  s390   sparc  unicore32  xtensa
```

3. Build the kernel and loadable modules:
   `make -j[n]`
   Builds the compressed kernel image (*arch/<arch>/boot/[b|z|u]image)*, uncompressed kernel image (*./vmlinux), System.map* and kernel modules.

4. Install the just-built kernel modules with
   `sudo make [INSTALL_MOD_PATH=<path/to/modules/dir>] modules_install`
   Installs the kernel modules under */lib/modules/`uname -r`*, or, if defined, under `INSTALL_MOD_PATH`

5. Set up your boot options as required (LILO / GRUB)
   `sudo make install`

   For x86: a) Creates and installs the initrd image under /boot
   b) Updates the bootloader configuration file to boot the new kernel (first entry)

6.

TIP: *Software packages to install for kernel build and kernel dev (below, on an Ubuntu 18.04.3 LTS):*

```
sudo apt update
sudo apt install gcc make perl

sudo apt install git fakeroot build-essential tar ncurses-dev tar xz-utils
libssl-dev bc python3-distutils libelf-dev linux-headers-$(uname -r) bison
flex libncurses5-dev util-linux net-tools linux-tools-$(uname -r) exuberant-
ctags cscope gnome-system-monitor curl perf-tools-unstable  gnuplot rt-tests
indent tree pmap smem numactl hwloc bpfcc-tools sparse flawfinder cppcheck
curl
```

Resources:

- ✔ How to Compile Linux Kernel from Source to Build Custom Kernel
- ✔ How to Compile and Install Linux Kernel v4.9 Source On a Debian / Ubuntu Linux
- ✔ How to Compile and Install Linux Kernel v4.5 Source <particularly for> On a Debian / Ubuntu Linux

- ✔ Useful! Which kernel configurables should we turn On minimally? Hard to answer, depends, but a good summary available here: systemd README : see the “Requirements” section – kernel configurables
- ✔ How to Configure the GRUB2 Boot Loader’s Settings
  [in brief: edit /etc/default/grub ; sudo update-grub]

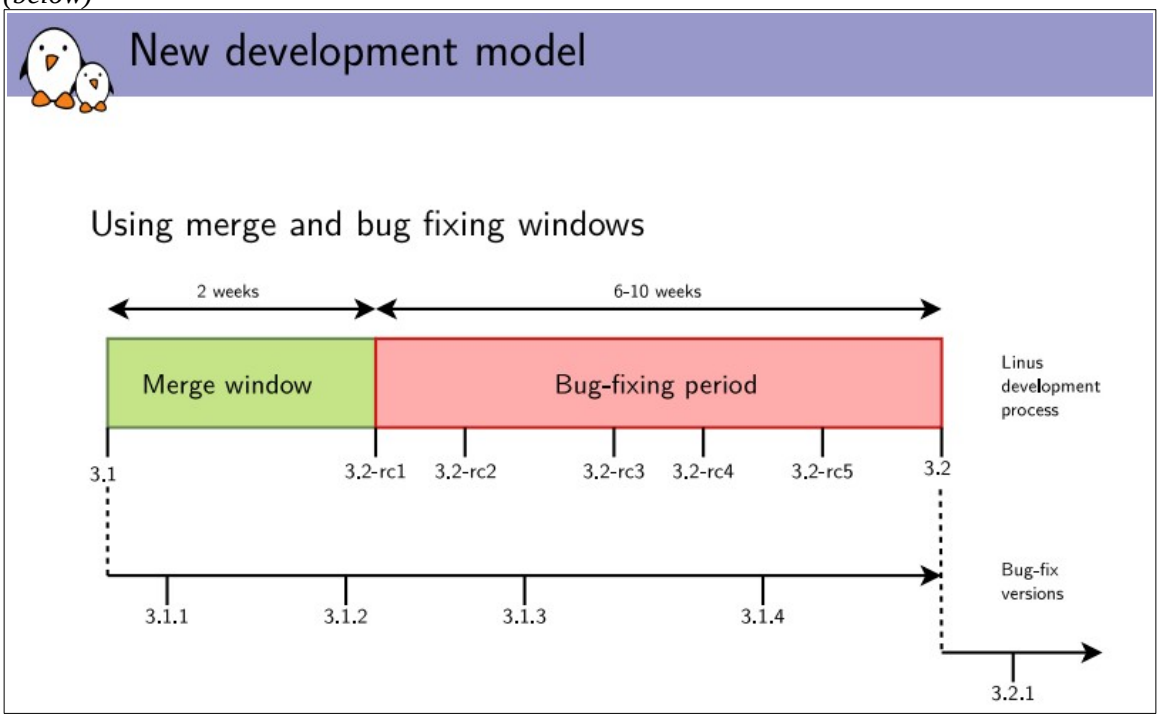
- ✔ Kbuild:

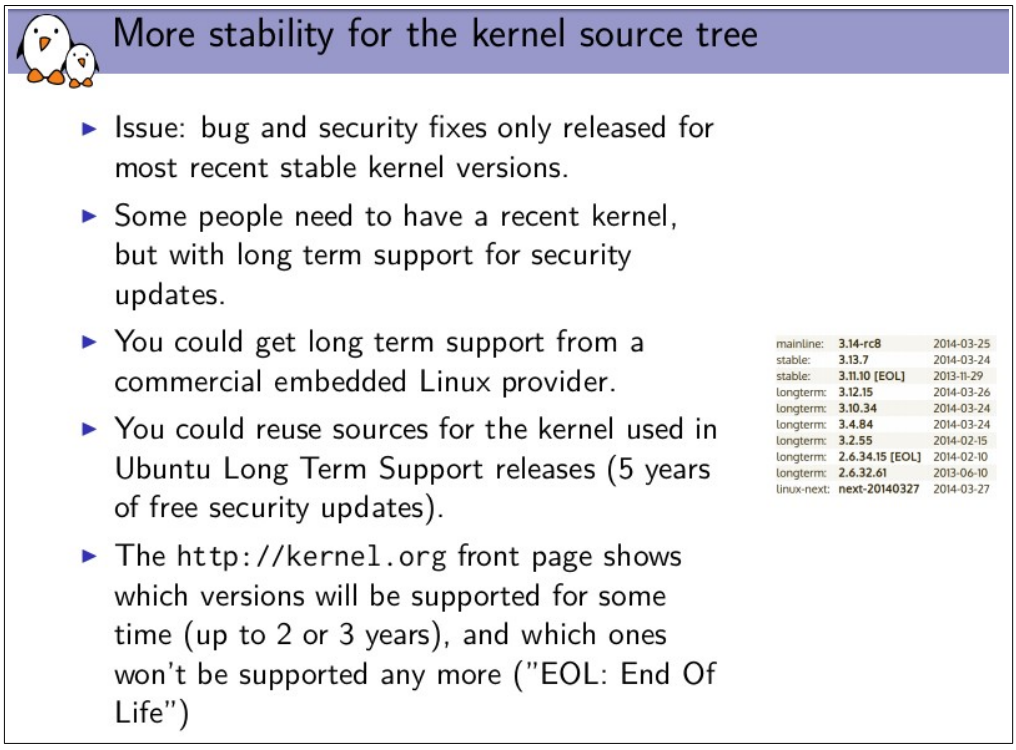
  Kbuild: the Linux Kernel Build System, LJ, Dec 2012
  How does kbuild actually work? [SO]

- ✔

Kernel Development Model and Releases

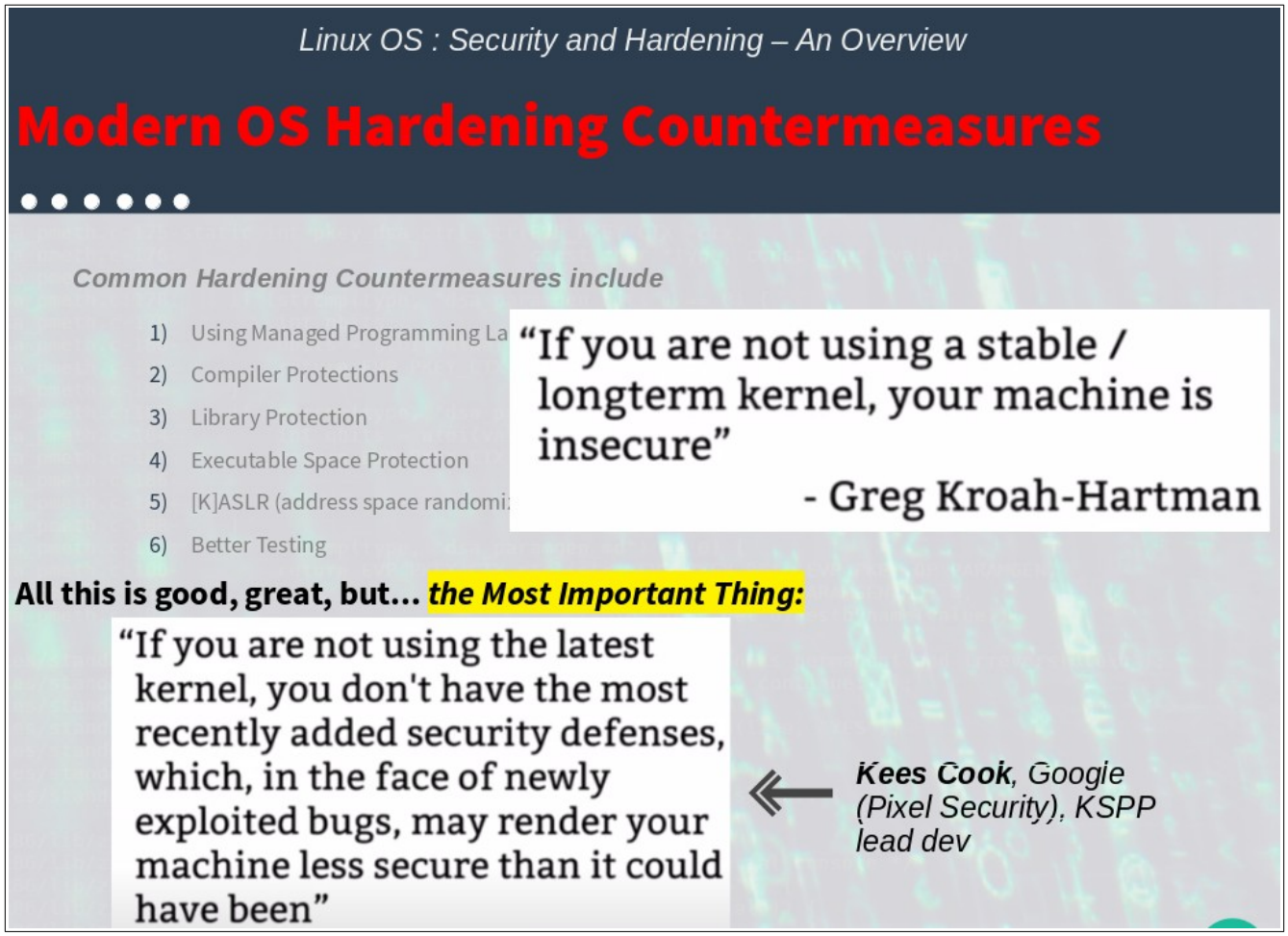
Source (below)





Kernel versions

Which kernel version to use?



Who will provide this (very) Long Term kernelSupport?

- **LTS (Long Term Stable) kernels**
- **SLTS (Super LTS) kernels too!**

from the *Civil Infrastructure Platform (CIP)* group [*link*]

A Linux Foundation (LF) project

4.4 SLTS kernel support until at least 2026, possibly 2036!

4.19 SLTS kernel support including ARM64

Quick Tips

---------- TIP ----------

USEFUL! Common case:

Build a kernel with appropriate configurables for the (Linux) system you are currently running on (or even another, for that matter):

from *Documentation/admin-guide/README.rst*

```
...
 "make localmodconfig" Create a config based on current config and loaded modules (lsmod).
Disables any module option that is not needed for the loaded modules.

To create a localmodconfig for another machine, store the lsmod of that machine into a
file and pass it in as a LSMOD parameter.

                target$ lsmod > /tmp/mylsmod

                host$ make LSMOD=/tmp/mylsmod localmodconfig
          [...]
   host$ make ...

The above also works when cross compiling.
...
```

---------- TIP ----------

Configure the kernel carefully!

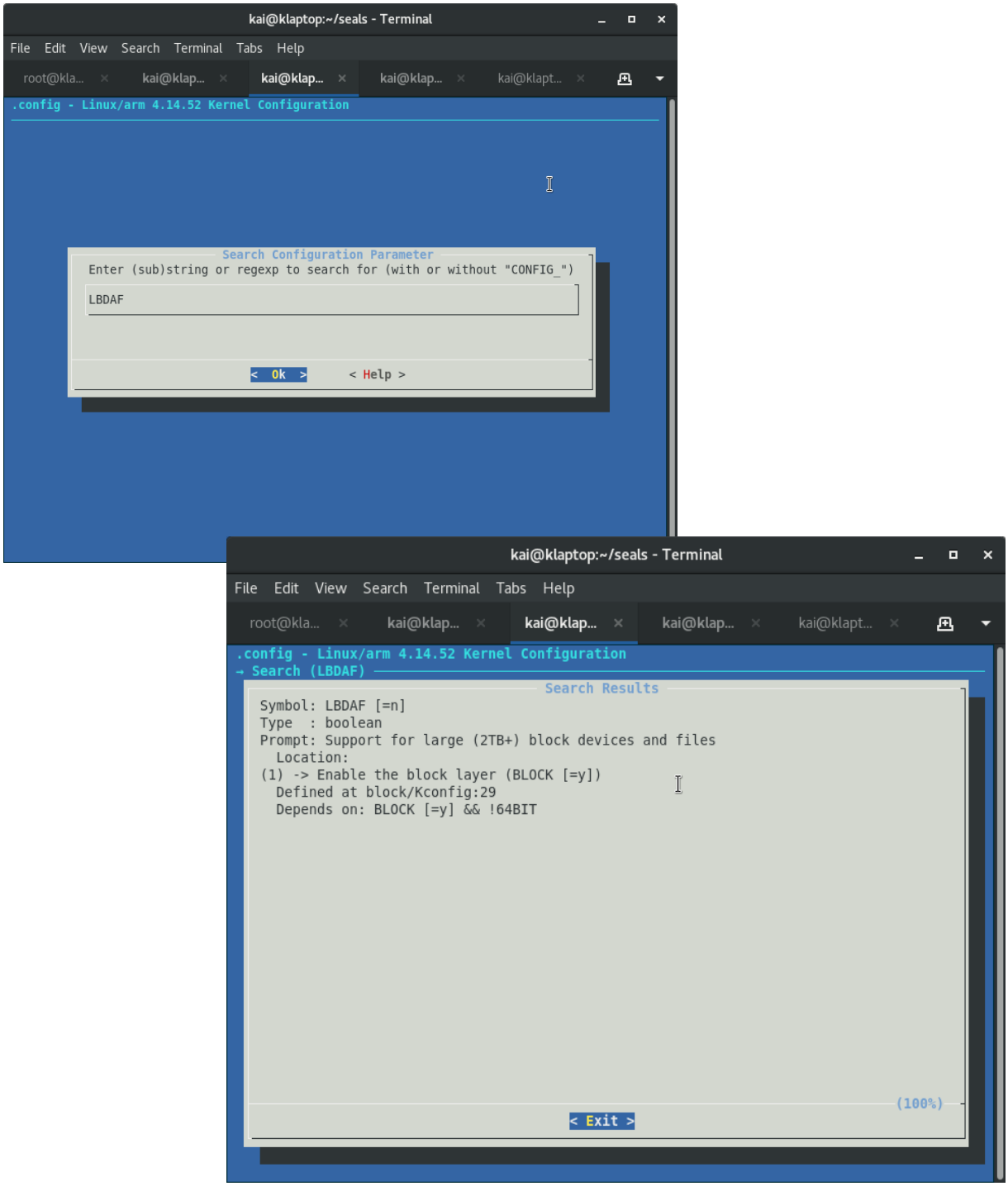
```
On an ARM system:

ARM # mount -o remount,rw /
EXT4-fs (mmcblk0): Filesystem with huge files cannot be mounted RDWR without CONFIG_LBDAF
EXT4-fs (mmcblk0): re-mounted. Opts: data=ordered
ARM #
```

In order to mount it as ‘rw’, the ext4 filesystem requires the CONFIG_LBDAF option to be set.

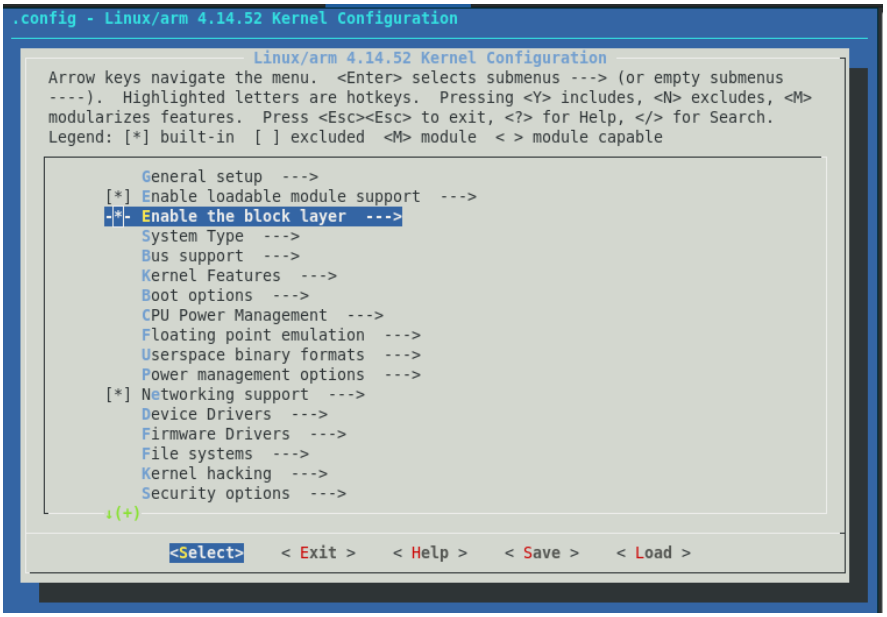
---------- TIP ----------

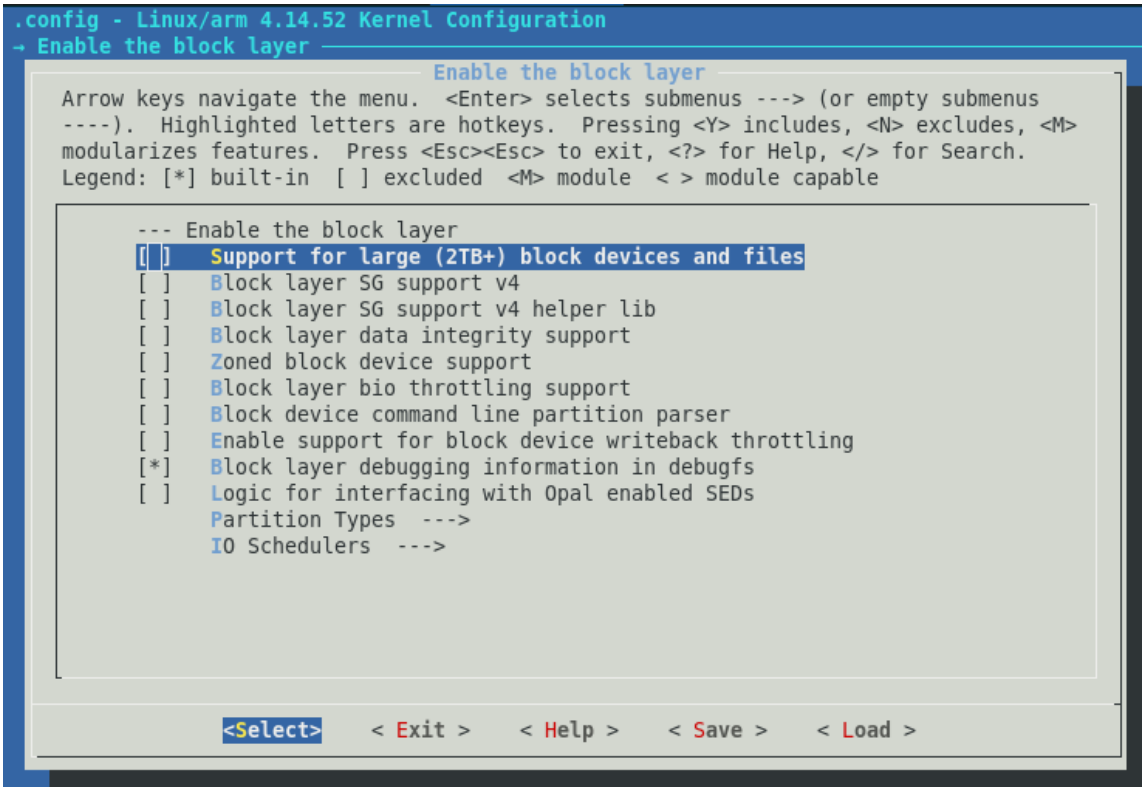
So, where in the *make menuconfig* menu is this CONFIG_LBDAF kernel config option? Search for it with the "usual" **"/"** operator!



Ah, we can now see: its under the menu item *"Enable the block layer"*.

So navigate there:





Turn on the highlighted option above by toggling it with the spacebar, save and build.

---------- TIP ----------

- With the GRUB2 bootloader, edit */etc/default/grub* as root and add your custom kernel command-line parameters:

For eg.

```
…
# GRUB_CMDLINE_LINUX_DEFAULT="quiet console=tty0 console=ttyS0,9600"
GRUB_CMDLINE_LINUX_DEFAULT="debug initcall_debug nolapic_timer 3"
…
$ sudo update-grub
...
```

---------- TIP ----------

- When booting a kernel in a VM (virtual machine), it's often useful to turn off the local APIC timer; use the below option in the target kernel's command line (via the GRUB2 bootloader menu system):

```
nolapic_timer   [X86-32,APIC] Do not use the local APIC timer.
```

---------- TIP ----------

[Oct 2016]
[PIC error!] When attempting to build a kernel (Oct 2016), got the error(s):

```
$ make
...
  CC      scripts/mod/empty.o
scripts/mod/empty.c:1:0: error: code model kernel does not support PIC mode
 /* empty file to figure out endianness / word size */
 ...
```

It turns out that it's not a kernel issue but rather a compiler settings one: modern gcc 6 and above uses the -fPIE (position independent executables) flag by default. We need to turn this OFF.

This Q&A:

Kernel doesn't support PIC mode for compiling?

Provides a patch for the Makefile (dated 10 May 2016); please apply the patch (applies to the kernel toplevel Makefile) and then build.

The patch itself:

```
--- a/Makefile
+++ b/Makefile
@@ -608,6 +608,12 @@ endif # $(dot-config)
 # Defaults to vmlinux, but the arch makefile usually adds further targets
 all: vmlinux

+# force no-pie for distro compilers that enable pie by default
```

```
+KBUILD_CFLAGS += $(call cc-option, -fno-pie)
+KBUILD_CFLAGS += $(call cc-option, -no-pie)
+KBUILD_AFLAGS += $(call cc-option, -fno-pie)
+KBUILD_CPPFLAGS += $(call cc-option, -fno-pie)
+
 # The arch Makefile can set ARCH_{CPP,A,C}FLAGS to override the default
 # values of the respective KBUILD_* variables
 ARCH_CPPFLAGS :=
```

---------- TIP ----------

The kernel build fails with:

```
[...] fatal error: openssl/openssl.h: No such file or directory
```

See: [OpenSSL missing during ./configure. How to fix?]
[…]
The OpenSSL library is usually already installed, but you have to install the header files. Depending on your Linux distribution, you'll need these packages:

- Red Hat, Fedora, CentOS - `openssl-devel`
- Debian, Ubuntu - `libssl-dev`
- Arch - `openssl`

---------- TIP ----------

Very useful! "make help"

Output below from a recent Linux kernel (4.10.0-rc2):

```
$ make help
Cleaning targets:
  clean        - Remove most generated files but keep the config and
                 enough build support to build external modules
  mrproper     - Remove all generated files + config + various backup files
  distclean    - mrproper + remove editor backup and patch files

Configuration targets:
  config       - Update current config utilising a line-oriented program
  nconfig       - Update current config utilising a ncurses menu based
                  program
  menuconfig   - Update current config utilising a menu based program
  xconfig      - Update current config utilising a Qt based front-end
  gconfig      - Update current config utilising a GTK+ based front-end
  oldconfig    - Update current config utilising a provided .config as base
  localmodconfig  - Update current config disabling modules not loaded
```

```
  localyesconfig  - Update current config converting local mods to core
  silentoldconfig - Same as oldconfig, but quietly, additionally update deps
  defconfig       - New config with default from ARCH supplied defconfig
  savedefconfig   - Save current config as ./defconfig (minimal config)
  allnoconfig       - New config where all options are answered with no
  allyesconfig      - New config where all options are accepted with yes
  allmodconfig      - New config selecting modules when possible
  alldefconfig    - New config with all symbols set to default
  randconfig      - New config with random answer to all options
  listnewconfig   - List new options
  olddefconfig      - Same as silentoldconfig but sets new symbols to their
                    default value
  kvmconfig   - Enable additional options for kvm guest kernel support
  xenconfig   - Enable additional options for xen dom0 and guest kernel support
  tinyconfig  - Configure the tiniest possible kernel

Other generic targets:
  all          - Build all targets marked with [*]
* vmlinux      - Build the bare kernel
* modules      - Build all modules
  modules_install - Install all modules to INSTALL_MOD_PATH (default: /)
  firmware_install- Install all firmware to INSTALL_FW_PATH
                    (default: $(INSTALL_MOD_PATH)/lib/firmware)
  dir/            - Build all files in dir and below
  dir/file.[ois]  - Build specified target only
  dir/file.lst    - Build specified mixed source/assembly target only
                    (requires a recent binutils and recent build (System.map))
  dir/file.ko     - Build module including final link
  modules_prepare - Set up for building external modules
  tags/TAGS   - Generate tags file for editors
  cscope      - Generate cscope index
  gtags           - Generate GNU GLOBAL index
  kernelrelease     - Output the release version string (use with make -s)
  kernelversion     - Output the version stored in Makefile (use with make -s)
  image_name  - Output the image name (use with make -s)
  headers_install - Install sanitised kernel headers to INSTALL_HDR_PATH
                    (default: ./usr)

Static analysers
  checkstack      - Generate a list of stack hogs
  namespacecheck  - Name space analysis on compiled kernel
  versioncheck    - Sanity check on version.h usage
  includecheck    - Check for duplicate included header files
  export_report   - List the usages of all exported symbols
  headers_check   - Sanity check on exported headers
  headerdep       - Detect inclusion cycles in headers
  coccicheck      - Check with Coccinelle.

Kernel selftest
  kselftest       - Build and run kernel selftest (run as root)
                    Build, install, and boot kernel before
                    running kselftest on it
  kselftest-clean - Remove all generated kselftest files
  kselftest-merge - Merge all the config dependencies of kselftest to existed
```

```
                    .config.

Kernel packaging:
  rpm-pkg             - Build both source and binary RPM kernel packages
  binrpm-pkg          - Build only the binary kernel RPM package
  deb-pkg             - Build both source and binary deb kernel packages
  bindeb-pkg          - Build only the binary kernel deb package
  tar-pkg             - Build the kernel as an uncompressed tarball
  targz-pkg           - Build the kernel as a gzip compressed tarball
  tarbz2-pkg          - Build the kernel as a bzip2 compressed tarball
  tarxz-pkg           - Build the kernel as a xz compressed tarball
  perf-tar-src-pkg    - Build perf-4.10.0-rc2.tar source tarball
  perf-targz-src-pkg  - Build perf-4.10.0-rc2.tar.gz source tarball
  perf-tarbz2-src-pkg - Build perf-4.10.0-rc2.tar.bz2 source tarball
  perf-tarxz-src-pkg  - Build perf-4.10.0-rc2.tar.xz source tarball

Documentation targets:
 Linux kernel internal documentation in different formats (Sphinx):
  htmldocs        - HTML
  latexdocs       - LaTeX
  pdfdocs         - PDF
  epubdocs        - EPUB
  xmldocs         - XML
  cleandocs       - clean all generated files

  make SPHINXDIRS="s1 s2" [target] Generate only docs of folder s1, s2
  valid values for SPHINXDIRS are: media core-api security admin-guide gpu process dev-
tools driver-api doc-guide

  make SPHINX_CONF={conf-file} [target] use *additional* sphinx-build
  configuration. This is e.g. useful to build with nit-picking config.

 Linux kernel internal documentation in different formats (DocBook):
  htmldocs        - HTML
  pdfdocs         - PDF
  psdocs          - Postscript
  xmldocs         - XML DocBook
  mandocs         - man pages
  installmandocs  - install man pages generated by mandocs
  cleandocs       - clean all generated DocBook files

  make DOCBOOKS="s1.xml s2.xml" [target] Generate only docs s1.xml s2.xml
  valid values for DOCBOOKS are: z8530book.xml kernel-hacking.xml kernel-locking.xml
deviceiobook.xml writing_usb_driver.xml networking.xml kernel-api.xml filesystems.xml
lsm.xml kgdb.xml gadget.xml libata.xml mtdnand.xml librs.xml rapidio.xml genericirq.xml
s390-drivers.xml uio-howto.xml scsi.xml sh.xml regulator.xml w1.xml
writing_musb_glue_layer.xml iio.xml

  make DOCBOOKS="" [target] Don't generate docs from Docbook
     This is useful to generate only the ReST docs (Sphinx)

Architecture specific targets (x86):
* bzImage      - Compressed kernel image (arch/x86/boot/bzImage)
  install      - Install kernel using
```

```
                  (your) ~/bin/installkernel or
                  (distribution) /sbin/installkernel or
                  install to $(INSTALL_PATH) and run lilo
  fdimage      - Create 1.4MB boot floppy image (arch/x86/boot/fdimage)
  fdimage144   - Create 1.4MB boot floppy image (arch/x86/boot/fdimage)
  fdimage288   - Create 2.8MB boot floppy image (arch/x86/boot/fdimage)
  isoimage     - Create a boot CD-ROM image (arch/x86/boot/image.iso)
                  bzdisk/fdimage*/isoimage also accept:
                  FDARGS="..."  arguments for the booted kernel
                  FDINITRD=file initrd for the booted kernel

  i386_defconfig           - Build for i386
  x86_64_defconfig         - Build for x86_64

  make V=0|1 [targets] 0 => quiet build (default), 1 => verbose build
  make V=2   [targets] 2 => give reason for rebuild of target
  make O=dir [targets] Locate all output files in "dir", including .config
  make C=1   [targets] Check all c source with $CHECK (sparse by default)
  make C=2   [targets] Force check of all c source with $CHECK
  make RECORDMCOUNT_WARN=1 [targets] Warn about ignored mcount sections
  make W=n   [targets] Enable extra gcc checks, n=1,2,3 where
1: warnings which may be relevant and do not occur too often
2: warnings which occur quite often but may still be relevant
3: more obscure warnings, can most likely be ignored
Multiple levels can be combined with W=12 or W=123

Execute "make" or "make all" to build all targets marked with [*]
For further info see the ./README file
$
```

Additional TIP

The *make help* is actually arch-specific; an example for ARM – notice how the available architecture targets are shown:

```
$ make ARCH=arm help
Cleaning targets:
  clean        - Remove most generated files but keep the config and
                     enough build support to build external modules
  mrproper     - Remove all generated files + config + various backup files
  distclean    - mrproper + remove editor backup and patch files

Configuration targets:
  config       - Update current config utilising a line-oriented program
  nconfig         - Update current config utilising a ncurses menu based program
  menuconfig   - Update current config utilising a menu based program

[...]

Devicetree:
* dtbs             - Build device tree blobs for enabled boards
  dtbs_install     - Install dtbs to /boot/dtbs/5.4.0
  dt_binding_check - Validate device tree binding documents
```

```
  dtbs_check       - Validate device tree source files

[...]

Architecture specific targets (arm):
* zImage        - Compressed kernel image (arch/arm/boot/zImage)
  Image         - Uncompressed kernel image (arch/arm/boot/Image)
* xipImage      - XIP kernel image, if configured (arch/arm/boot/xipImage)
  uImage        - U-Boot wrapped zImage
  bootpImage    - Combined zImage and initial RAM disk
                  (supply initrd image via make variable INITRD=<path>)
[...]
  am200epdkit_defconfig    - Build for am200epdkit
  aspeed_g4_defconfig      - Build for aspeed_g4
  aspeed_g5_defconfig      - Build for aspeed_g5
  assabet_defconfig        - Build for assabet
  at91_dt_defconfig        - Build for at91_dt
[...]
  versatile_defconfig      - Build for versatile
  vexpress_defconfig       - Build for vexpress
  vf610m4_defconfig        - Build for vf610m4
  viper_defconfig          - Build for viper
  vt8500_v6_v7_defconfig   - Build for vt8500_v6_v7
  xcep_defconfig           - Build for xcep
  zeus_defconfig           - Build for zeus
  zx_defconfig             - Build for zx

  make V=0|1 [targets] 0 => quiet build (default), 1 => verbose build
  make V=2   [targets] 2 => give reason for rebuild of target
  make O=dir [targets] Locate all output files in "dir", including .config
[...]
$
```

---------- TIP ----------

Attempting to compile an LKM (Loadable Kernel Module), requires the
`/lib/modules/$(uname -r)/build`
symlink to be correctly setup – pointing to the kernel source tree. Even if that's okay, the LKM build may fail with this message:

```
[...]

 ERROR: Kernel configuration is invalid.
         include/generated/autoconf.h or include/config/auto.conf are missing.
         Run 'make oldconfig && make prepare' on kernel src to fix it.
[...]
```

Follow the instructions as shown above and proceed.

---------- TIP ----------

- A simple convenience script for kernel build: (find it in the source dir : kbuild.sh)

```
#!/bin/bash
# kbuild.sh
# Simple kernel build script
name=$(basename $0)
[ $# -ne 1 ] && {
    echo "Usage: ${name} {kernel-source-tree-pathname}"
    exit 1
}
KSRC=$1
[ ! -d ${KSRC} ] && {
    echo "${name}: dir ${KSRC} invalid?"
    exit 1
}

runcmd()
{
echo "[+] $@"
eval "$@"
[ $? -ne 0 ] && {
   echo " Command \"$@\" failed, aborting ..."
   exit 1
  }
}

### "main" here
cd ${KSRC} || exit 1

# Start with the 'tiniest' config...
#runcmd "make tinyconfig"

runcmd "make defconfig"
# ... and then add on the kmods we currently use
lsmod > /tmp/lsmod
runcmd "make LSMOD=/tmp/lsmod localmodconfig"
rm -f /tmp/lsmod

runcmd "ls -l .config"
# make oldconfig: "Update current config utilising a provided .config as base"
runcmd "make oldconfig"

CPU_CORES=$(nproc)
jobs=$((2*${CPU_CORES}))
runcmd "time make -j${jobs}"
runcmd "sudo make -j${jobs} modules_install"
runcmd "sudo make install"

echo "[+] Done."
```

---------- TIP ----------

Change the default boot kernel (Ubuntu)
https://askubuntu.com/questions/216398/set-older-kernel-as-default-grub-entry

---------- TIP ----------

- Finally, remember, an (almost) guaranteed way to succeed :–)

When you get those build / boot errors, etc... that you cannot fix: copy the exact error message into the clipboard, go to Google, type “linux kernel build fails with: <paste-the-error-message>” !

One might be surprised at how often this helps ;-)

If not, post your (well thought-out) question.

Actual option switches, flags used by gcc when building the Linux kernel for an ARMv7 (Cortex-A9) ARM Versatile Express platform:

Toolchain is from Linaro for ARM (Aarch32):

```
$ arm-linux-gnueabihf-gcc --version
arm-linux-gnueabihf-gcc (Linaro GCC 7.3-2018.05) 7.3.1 20180425 [linaro-7.3-2018.05 revision
d29120a424ecfbc167ef90065c0eeb7f91977701]
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

$


$ make V=1 -j8 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- all

[...]

<< Note:
a) this is _not_ a 'debug' build
b) We've inserted newlines into the output stream below to make it more human-readable
>>

arm-linux-gnueabihf-gcc
 -Wp,-MD,arch/arm/kernel/.sys_arm.o.d  -nostdinc
 -isystem <...>/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf/bin/../lib/gcc/arm-linux-
gnueabihf/7.3.1/include
 -I./arch/arm/include -Iarch/arm/include/generated/uapi
 -Iarch/arm/include/generated  -Iinclude -I./arch/arm/include/uapi
 -Iarch/arm/include/generated/uapi -I./include/uapi
 -Iinclude/generated/uapi
 -include ./include/linux/kconfig.h
 -D__KERNEL__
 -mlittle-endian
 -Wall -Wundef -Wstrict-prototypes -Wno-trigraphs
 -fno-strict-aliasing -fno-common -Werror-implicit-function-declaration
 -Wno-format-security -std=gnu89 -fno-dwarf2-cfi-asm -fno-ipa-sra
 -mabi=aapcs-linux -mno-thumb-interwork -mfpu=vfp -funwind-tables
 -marm -D__LINUX_ARM_ARCH__=7 -march=armv7-a -msoft-float -Uarm
 -fno-delete-null-pointer-checks -fno-PIE
 -O2
 --param=allow-store-data-races=0 -DCC_HAVE_ASM_GOTO
 -Wframe-larger-than=1024 -fno-stack-protector -Wno-unused-but-set-variable
 -Wno-unused-const-variable -fomit-frame-pointer
 -fno-var-tracking-assignments
 -g -Wdeclaration-after-statement -Wno-pointer-sign -fno-strict-overflow
 -fconserve-stack -Werror=implicit-int -Werror=strict-prototypes
 -Werror=date-time    -D"KBUILD_STR(s)=#s"
 -D"KBUILD_BASENAME=KBUILD_STR(sys_arm)"
 -D"KBUILD_MODNAME=KBUILD_STR(sys_arm)"
 -c -o arch/arm/kernel/sys_arm.o arch/arm/kernel/sys_arm.c

[...]
```

Below is a 'debug' build

```
<...>/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf/bin/../libexec/gcc/arm-linux-
gnueabihf/7.3.1/cc1
 -quiet
 -nostdinc
 -I ./arch/arm/include
 -I ./arch/arm/include/generated
 -I ./include
 -I ./arch/arm/include/uapi
 -I ./arch/arm/include/generated/uapi
 -I ./include/uapi
 -I ./include/generated/uapi
 -imultilib .
 -imultiarch arm-linux-gnueabihf
 -iprefix <...>/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf/bin/../lib/gcc/arm-linux-
gnueabihf/7.3.1/
 -isysroot <...>/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf/bin/../arm-linux-
gnueabihf/libc
 -D __KERNEL__
 -D __LINUX_ARM_ARCH__=7
 -U arm
 -D CC_HAVE_ASM_GOTO
 -D KBUILD_BASENAME="ioctl"
 -D KBUILD_MODNAME="ioctl"
 -isystem <...>/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf/bin/../lib/gcc/arm-linux-
gnueabihf/7.3.1/include
 -include ./include/linux/kconfig.h
 -MD block/.ioctl.o.d block/ioctl.c
 -quiet
 -dumpbase ioctl.c
 -mlittle-endian -mapcs -mno-sched-prolog
 -mabi=aapcs-linux -mno-thumb-interwork -mfpu=vfp
 -marm -march=armv7-a -mfloat-abi=soft -mtune=cortex-a9
 -mtls-dialect=gnu
 -auxbase-strip block/ioctl.o
 -g -gdwarf-4 -O2 -Wall
 -Wundef -Wstrict-prototypes -Wno-trigraphs
 -Werror=implicit-function-declaration -Wno-format-security -Wno-frame-address
 -Wformat-truncation=0 -Wformat-overflow=0 -Wno-int-in-bool-context
 -Wframe-larger-than=1024 -Wno-unused-but-set-variable
 -Wunused-const-variable=0 -Wdeclaration-after-statement -Wno-pointer-sign
 -Werror=implicit-int -Werror=strict-prototypes -Werror=date-time
 -Werror=incompatible-pointer-types -Werror=designated-init
 -std=gnu90 -p -fno-strict-aliasing -fno-common -fshort-wchar -fno-PIE
 -fno-dwarf2-cfi-asm -fno-ipa-sra -funwind-tables
 -fno-delete-null-pointer-checks -fno-reorder-blocks -fno-ipa-cp-clone
 -fno-partial-inlining -fstack-protector -fno-omit-frame-pointer
 -fno-optimize-sibling-calls -fno-var-tracking-assignments -fno-strict-overflow
 -fno-merge-all-constants -fmerge-constants -fstack-check=no -fconserve-stack
 --param allow-store-data-races=0 -o /tmp/ccoUhjck.s

[...]


<< Another build run ... >>

arm-linux-gnueabihf-gcc
 -Wp,-MD,arch/arm/kernel/.process.o.d  -nostdinc
```

```
 -isystem <...>/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabihf/bin/../lib/gcc/arm-linux-
gnueabihf/7.2.1/include
 -I./arch/arm/include -I./arch/arm/include/generated
 -I./include -I./arch/arm/include/uapi -I./arch/arm/include/generated/uapi
 -I./include/uapi -I./include/generated/uapi
 -include ./include/linux/kconfig.h
 -D__KERNEL__
 -mlittle-endian -Wall -Wundef -Wstrict-prototypes -Wno-trigraphs
 -fno-strict-aliasing -fno-common -fshort-wchar
 -Werror-implicit-function-declaration -Wno-format-security -std=gnu89
 -fno-PIE -fno-dwarf2-cfi-asm -fno-omit-frame-pointer -mapcs -mno-sched-prolog
 -fno-ipa-sra -mabi=aapcs-linux -mno-thumb-interwork -mfpu=vfp -funwind-tables
 -marm -D__LINUX_ARM_ARCH__=7 -march=armv7-a -msoft-float -Uarm
 -fno-delete-null-pointer-checks -Wno-frame-address -Wno-format-truncation
 -Wno-format-overflow -Wno-int-in-bool-context
 -O2 --param=allow-store-data-races=0 -DCC_HAVE_ASM_GOTO -fno-reorder-blocks
 -fno-ipa-cp-clone -fno-partial-inlining -Wframe-larger-than=1024
 -fstack-protector -Wno-unused-but-set-variable -Wno-unused-const-variable
 -fno-omit-frame-pointer -fno-optimize-sibling-calls
 -fno-var-tracking-assignments
 -g -gdwarf-4 -pg -Wdeclaration-after-statement -Wno-pointer-sign
 -fno-strict-overflow -fno-merge-all-constants -fmerge-constants
 -fno-stack-check -fconserve-stack -Werror=implicit-int
 -Werror=strict-prototypes -Werror=date-time -Werror=incompatible-pointer-types
 -Werror=designated-init    -DKBUILD_BASENAME='"process"'
 -DKBUILD_MODNAME='"process"' -c -o arch/arm/kernel/process.o
 arch/arm/kernel/process.c
...
```

Kernel custom configuration -OR- Setting up your own menu entries in kernel build

The Kconfig file, within each source folder, is the relevant one . Each source folder has a *Kconfig*:

init/Kconfig - defines the 'General Setup' menu items!

Lets explain with an example:

We'd like to add a kernel menu option under the "General Setup" menu:

```
"[ ] My Amazing Kernel Feature"
```

By default it should be OFF.

When turned ON, the effect will be that we compile the kernel with some additional gcc switches (details below).

How is this setup?

1. Edit the *init/Kconfig* file

(why this one? -because the "General Setup" menu items are defined here. As another eg, the "Kernel Hacking" menu is defined in *lib/Kconfig.debug*).

1.1 Add the following paragraph to set things up:

```
config AMAZING
    bool "My Amazing Kernel Feature"
    default n
    help
      Turns on the hook that will cause this kernel to ...
```

```
       blah blah blah
      ...
```

1.2 Save & exit

2. Run '*make menuconfig*'

2.1 Goto the General Setup menu

2.2 You should npow see a new entry - the one just created, like this:

```
   [ ] My Amazing Kernel Feature
```

2.3 Explore, change it, see the help..

2.4 Once done, save & exit (the config).

3. The resulting .config file will reflect whether the user selected the new entry or not:

```
$ grep CONFIG_AMAZING .config
```

If turned ON, the result of the grep above will be:

```
CONFIG_AMAZING=y
```

If turned OFF (the default in our example), the result of the grep above will be:

```
# CONFIG_AMAZING is not set
```

Additionally, the corresponding "CONFIG_FOO" define is auto-generated in the *include/generated/autoconf.h* header: (example below shows when it was selected (ON)):

```
<..>/include/generated/autoconf.h:49:#define CONFIG_AMAZING 1
```

4. Edit the (toplevel or other) Makefile to figure action based on our new directive:

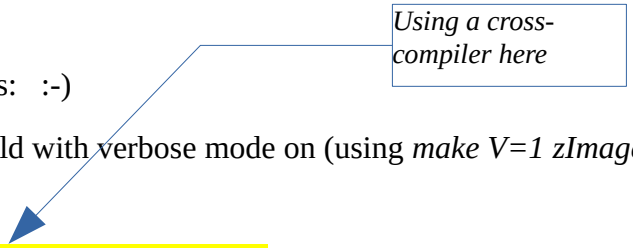
... <here we inserted these lines into the toplevel Makefile>

```
ifdef CONFIG_AMAZING
KBUILD_CFLAGS += -finstrument-functions -g
endif
```

...

So, if the "My Amazing Kernel Feature" option is used, all C source files will now be compiled with the additional "-finstrument-functions -g" gcc switches. If left unselected (during the kernel config step), these switches will not be used during compilation.

Cool!



Proof that it works: :-)

See the kernel build with verbose mode on (using *make V=1 zImage ...*):

```
...
arm-buildroot-linux-uclibcgnueabi-gcc -Wp,-MD,block/.elevator.o.d  -nostdinc
[...]
-Wframe-larger-than=1024 -fno-stack-protector -Wno-unused-but-set-variable -fno-omit-
frame-pointer -fno-optimize-sibling-calls -g -finstrument-functions -g -Wdeclaration-
after-statement -Wno-pointer-sign -fno-strict-overflow -fconserve-stack -DCC_HAVE_ASM_GOTO
-D"KBUILD_STR(s)=#s" -D"KBUILD_BASENAME=KBUILD_STR(elevator)"  -
D"KBUILD_MODNAME=KBUILD_STR(elevator)" -c -o block/.tmp_elevator.o block/elevator.c
```

<<
For working with Git in general, (and with the Linux kernel upstream development in particular), please refer to the ***'Git – The Basics'*** PDF tutorial.
>>

The "kbuild Test Robot" - employed by the kernel community

From:
0-DAY kernel test infrastructure Open Source Technology Center
https://lists.01.org/ Intel Corporation

An example:

[kernel-hardening] [PATCH 4/6] Protectable Memory

kbuild test robot <lkp@intel.com> Fri, Feb 2, 2018 at 11:11 AM
To: Igor Stoppa <igor.stoppa@huawei.com>
Cc: kbuild-all@01.org, jglisse@redhat.com, keescook@chromium.org, mhocko@kernel.org, labbott@redhat.com, hch@infradead.org, willy@infradead.org, cl@linux.com, linux-security-module@vger.kernel.org, linux-mm@kvack.org, linux-kernel@vger.kernel.org, kernel-hardening@lists.openwall.com, Igor Stoppa <igor.stoppa@huawei.com>

Hi Igor,

Thank you for the patch! Perhaps something to improve:

[auto build test WARNING on linus/master]
[also build test WARNING on v4.15]
[cannot apply to next-20180201]
[if your patch is applied to the wrong git tree, please drop us a note to help improve the system]

url: https://github.com/0day-ci/
config: i386-randconfig-x071-201804 (attached as .config)
compiler: gcc-7 (Debian 7.2.0-12) 7.2.1 20171025
reproduce:
```
# save the attached .config to linux build tree
make ARCH=i386
```

All warnings (new ones prefixed by >>):

```
   mm/pmalloc.c: In function 'pmalloc_pool_show_avail':
>> mm/pmalloc.c:71:25: warning: format '%lu' expects argument of type 'long unsigned int', but argument 3 has type 'size_t {aka unsigned int}' [-Wformat=]
     return sprintf(buf, "%lu\n", gen_pool_avail(data->pool));
                         ~~^     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                         %u
   mm/pmalloc.c: In function 'pmalloc_pool_show_size':
   mm/pmalloc.c:81:25: warning: format '%lu' expects argument of type 'long unsigned int', but argument 3 has type 'size_t {aka unsigned int}' [-Wformat=]
     return sprintf(buf, "%lu\n", gen_pool_size(data->pool));
                         ~~^     ~~~~~~~~~~~~~~~~~~~~~~~~~~~
                         %u
```

vim +71 mm/pmalloc.c

```
   63
   64  static ssize_t pmalloc_pool_show_avail(struct kobject *dev,
   65                                         struct kobj_attribute *attr,
   66                                         char *buf)
   67  {
   68          struct pmalloc_data *data;
   69
   70          data = container_of(attr, struct pmalloc_data, attr_avail);
  > 71           return sprintf(buf, "%lu\n", gen_pool_avail(data->pool));
   72  }
   73
```

0-DAY kernel test infrastructure Open Source Technology Center
https://lists.01.org/ Intel Corporation

<< Attached: the .config.gz >>

Initramfs / initrd

Rationale behind initrd

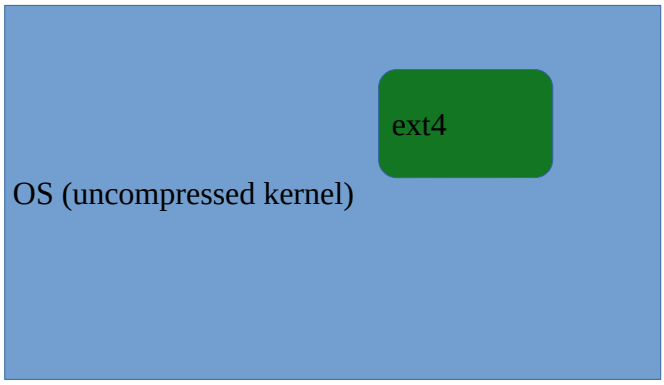
initrd – initial RAM disk / initramfs ;

mount “/” : kernel at boot
... init ... ; runs /sbin/init

Kernel must ‘understand’ the root fs : implies it has the filesystem driver !

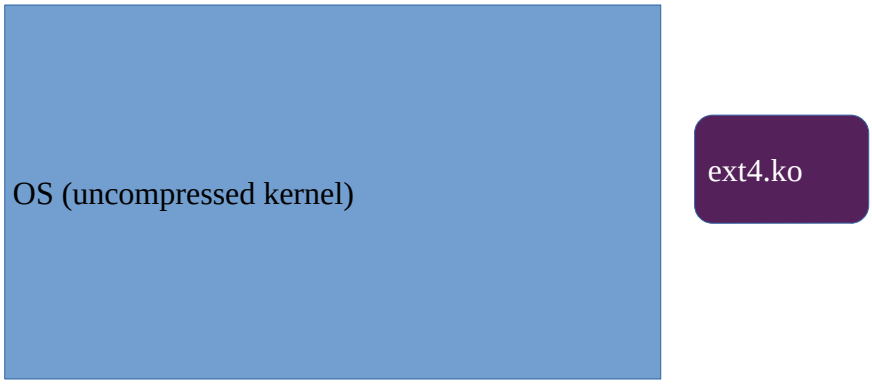
ext4 ; fs ‘driver’

Builtin:



mount *will succeed*

Else, it’s a kernel module:



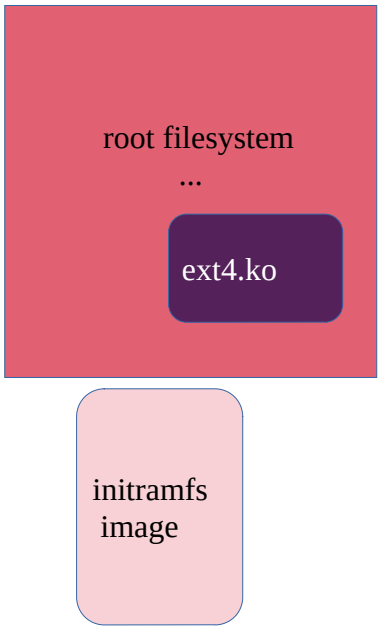
mount of rootfs will not succeed until the ext4 kernel module is loaded into RAM

insmod / modprobe <pathname.ko>

BUT: the kernel module is here : /lib/modules/4.14.183/
in the root filesystem !

In order to mount the rootfs, we require the ext4.ko
BUT it's in the rootfs !

Chicken and Egg problem !!!



How to solve this??

initramfs !

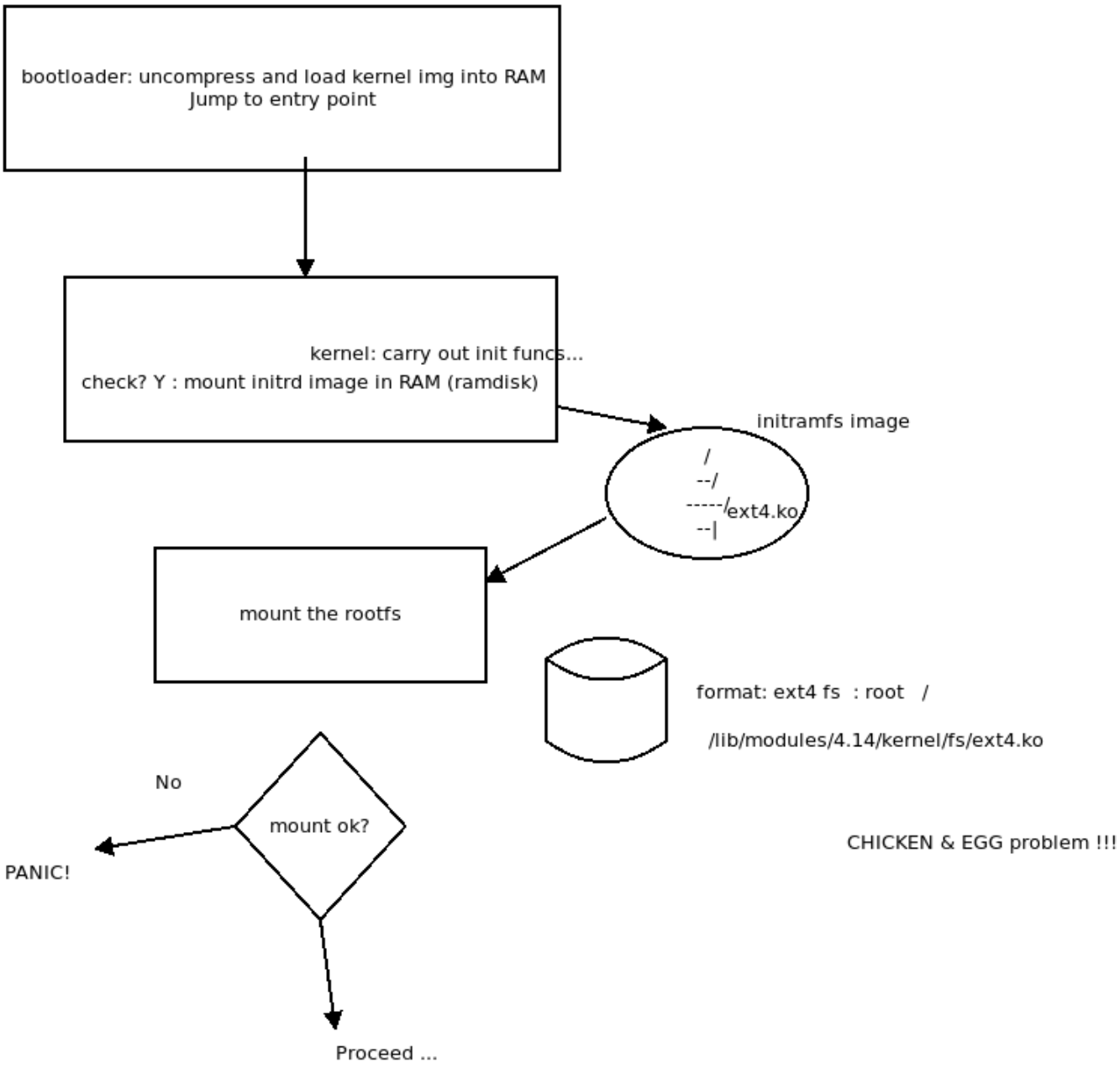
contents of / scaled-down;
/bin; /dev/ ; /etc/ ; /lib ; /usr ; ...

kernel mounts initramfs as a temporary rootfs;
scripts within load up all required drivers etc;
kernel then mounts the real root filesystem;
once done, it unmounts the temporary initramfs.

From the "Kernel Rebuild Guide" by Kwan Lowe, Digital Hermit
[http://www.digitalhermit.com/linux/Kernel-Build-HOWTO.html] :

If you have built your main boot drivers as modules (e.g., SCSI host adapter, filesystem, RAID drivers) then you will need to create an initial RAMdisk image. The initrd is a way of sidestepping the chicken and egg problem of booting -- drivers are needed to load the root filesystem but the filesystem cannot be loaded because the drivers are on the filesystem. As the manpage for **mkinitrd** states:

> mkinitrd creates filesystem images which are suitable for use as Linux initial ramdisk (initrd) images. Such images are often used for preloading the block device modules (such as IDE, SCSI or RAID) which are needed to access the root filesystem. mkinitrd automatically loads filesystem modules (such as ext3 and jbd), IDE modules, all scsi_hostadapter entries in /etc/modules.conf, and raid modules if the systems root partition is on raid, which makes it simple to build and use kernels using modular device drivers.



Initramfs useful for stuff like:

- running an app *before* the kernel fully comes up
  - get password for encrypted block device(s)
  - set console font
  - set keyboard map
- sometimes *only* an initrd image is required
  - no / WORM storage device (kiosk, etc)

- kdump (second) kernel – boots into an initrd environment (simply to send the */proc/vmcore* file over the network to a server system)

Also see a good article on unpacking, changing and packing back an initrd image here:
http://www.alexonlinux.com/opening-and-modifying-the-initrd

Listing the initramfs contents

[On Fedora 27 x86_64 ; on Ubuntu, use `lsinitramfs` *!]*

```
$ sudo lsinitrd
[sudo] password for xxx:
Image: /boot/initramfs-4.15.10-300.fc27.x86_64.img: 21M
================================================================================
Early CPIO image
================================================================================
drwxr-xr-x   3 root     root            0 Jan  5 14:48 .
-rw-r--r--   1 root     root            2 Jan  5 14:48 early_cpio
drwxr-xr-x   3 root     root            0 Jan  5 14:48 kernel
drwxr-xr-x   3 root     root            0 Jan  5 14:48 kernel/x86
drwxr-xr-x   2 root     root            0 Jan  5 14:48 kernel/x86/microcode
-rw-r--r--   1 root     root        99328 Jan  5 14:48
kernel/x86/microcode/GenuineIntel.bin
================================================================================
Version: dracut-046-8.git20180105.fc27

Arguments: -f

dracut modules:
bash
systemd
systemd-initrd

[...]

shutdown
================================================================================
drwxr-xr-x  12 root     root            0 Jan  5 14:48 .
crw-r--r--   1 root     root       5,   1 Jan  5 14:48 dev/console
crw-r--r--   1 root     root       1,  11 Jan  5 14:48 dev/kmsg
crw-r--r--   1 root     root       1,   3 Jan  5 14:48 dev/null
crw-r--r--   1 root     root       1,   8 Jan  5 14:48 dev/random
crw-r--r--   1 root     root       1,   9 Jan  5 14:48 dev/urandom
lrwxrwxrwx   1 root     root            7 Jan  5 14:48 bin -> usr/bin
drwxr-xr-x   2 root     root            0 Jan  5 14:48 dev
drwxr-xr-x  11 root     root            0 Jan  5 14:48 etc
drwxr-xr-x   2 root     root            0 Jan  5 14:48 etc/cmdline.d
drwxr-xr-x   2 root     root            0 Jan  5 14:48 etc/conf.d
-rw-r--r--   1 root     root          124 Jan  5 14:48 etc/conf.d/systemd.conf
-rw-r--r--   1 root     root          303 Jan  5 14:48 etc/dhclient.conf
```

```
...

-rw-r--r--   1 root     root         1377 Aug  4  2017 usr/share/terminfo/v/vt220
lrwxrwxrwx   1 root     root           20 Jan  5 14:48 usr/share/unimaps ->
/usr/lib/kbd/unimaps
drwxr-xr-x   3 root     root            0 Jan  5 14:48 var
lrwxrwxrwx   1 root     root           11 Jan  5 14:48 var/lock -> ../run/lock
lrwxrwxrwx   1 root     root            6 Jan  5 14:48 var/run -> ../run
drwxr-xr-x   2 root     root            0 Jan  5 14:48 var/tmp
================================================================================
$
```

Note: the newer version of mkinitrd is *mkinitramfs.*
A good article on Initramfs.
