BUILDING THE MODERN LINUX KERNEL

© 2000-2024 Kaiwan N Billimoria, kaiwanTECH

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.

2000-2024 Kaiwan N Billimoria
kaiwanTECH, Bangalore, India.

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: https://bit.ly/32bTE8A?r=lp

Building the modern (2.6 onwards) Linux Kernel
Quick Step Summary, biased to x86_64

0. Install all required packages (see below)

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) Verify it with gpg [see procedure below]
   c) 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)
   For x86:
   `sudo make install`
   a) Creates and installs the initrd image under /boot
   b) Updates the bootloader configuration file to boot the new kernel (first entry)

Step 1b. Verifying the kernel source

Ref: https://www.kernel.org/signature.html

For the 6.1.25 kernel:

1. Download the kernel source; we download this file:
   https://mirrors.edge.kernel.org/pub/linux/kernel/v6.x/linux-6.1.25.tar.xz

2. We require the public key(s); retrieve it from the keyserver (let's get Linus's and Greg-KH's public keys):

   ```
   $ gpg2 --locate-keys torvalds@kernel.org gregkh@kernel.org
   gpg: key 38DBBDC86092693E: public key "Greg Kroah-Hartman
   <gregkh@kernel.org>" imported
   gpg: Total number processed: 1
   gpg:               imported: 1
   gpg: key 79BE3E4300411886: public key "Linus Torvalds <torvalds@kernel.org>"
   imported
   gpg: Total number processed: 1
   ...
   ```

3. Now let's try to verify the tarred-and-compressed source tree:

   ```
   $ xzcat ../linux-6.1.25.tar.xz | gpg2 --verify ./linux-6.1.25.tar.sign -
   gpg: Signature made Thursday 20 April 2023 04:06:06 PM IST
   gpg:                using RSA key 647F28654894E3BD457199BE38DBBDC86092693E
   gpg: Good signature from "Greg Kroah-Hartman <gregkh@kernel.org>" [unknown]
   gpg: WARNING: This key is not certified with a trusted signature!
   gpg:          There is no indication that the signature belongs to the
   owner.
   ...
   ```

4. As mentioned here:
   To make the "`WARNING`" message go away you can indicate that you choose to trust that key using TOFU - "Trust on First Use" :

   ```
   $ xzcat ../linux-6.1.25.tar.xz | gpg2 --trust-model tofu --verify ./linux-
   6.1.25.tar.sign -
   gpg: Signature made Thursday 20 April 2023 04:06:06 PM IST
   gpg:                using RSA key 647F28654894E3BD457199BE38DBBDC86092693E
   gpg: Good signature from "Greg Kroah-Hartman <gregkh@kernel.org>" [full]
   gpg: gregkh@kernel.org: Verified 1 signatures in the past 0 seconds.
   Encrypted
        0 messages.
   $
   ```

   There, it's verified!

5. If it shows "BAD SIGNATURE" then there's an issue; don't use this kernel source! else, all well, untar it and build it!

A script to do precisely this is available here!

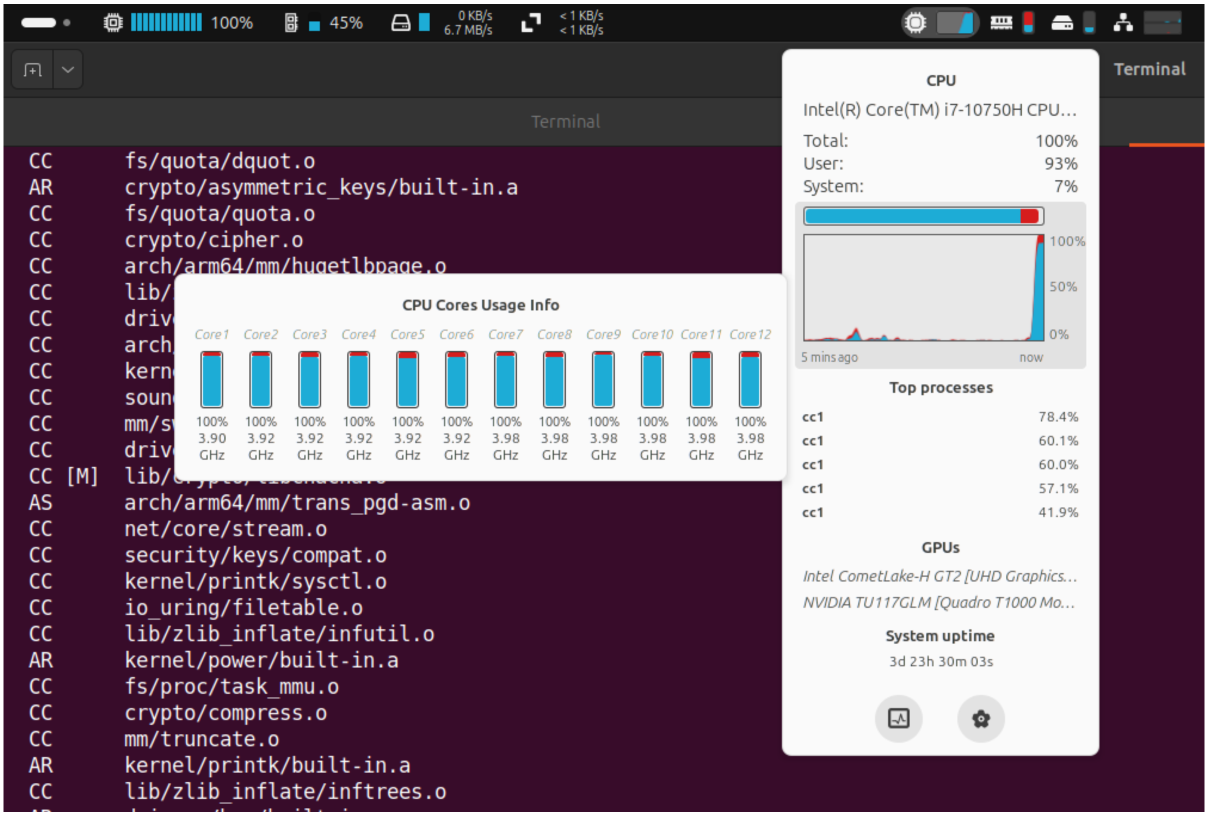
***TIP:** Software packages to install for kernel build and kernel dev (below, on an Ubuntu 20.04 LTS system):*

(Not all packages listed below are required for the kernel build itself)

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

sudo apt install -y \
 bison build-essential flex \
 libelf-dev libssl-dev ncurses-dev \
 pahole tar util-linux xz-utils zstd
```

As well, see this convenience script:
https://github.com/PacktPublishing/Linux-Kernel-Programming_2E/blob/main/ch1/pkg_install4ubuntu_lkp.sh

Partial screenshot while building the Linux kernel on a host with 12 CPU cores (with `make -j24` *)!*
(The applets on the top bar, right side are Astra Monitor!; on the left are the TopHat applets):



```
CC      fs/quota/dquot.o
AR      crypto/asymmetric_keys/built-in.a
CC      fs/quota/quota.o
CC      crypto/cipher.o
CC      arch/arm64/mm/hugetlbpage.o
CC      lib/
CC      driv
CC      arch
CC      kern
CC      soun
CC      mm/s
CC      driv
CC [M]  lib/
AS      arch/arm64/mm/trans_pgd-asm.o
CC      net/core/stream.o
CC      security/keys/compat.o
CC      kernel/printk/sysctl.o
CC      io_uring/filetable.o
CC      lib/zlib_inflate/infutil.o
AR      kernel/power/built-in.a
CC      fs/proc/task_mmu.o
CC      crypto/compress.o
CC      mm/truncate.o
AR      kernel/printk/built-in.a
CC      lib/zlib_inflate/inftrees.o
```

CPU Cores Usage Info

Core1	Core2	Core3	Core4	Core5	Core6	Core7	Core8	Core9	Core10	Core11	Core12
100% 3.90 GHz	100% 3.92 GHz	100% 3.92 GHz	100% 3.92 GHz	100% 3.92 GHz	100% 3.92 GHz	100% 3.98 GHz	100% 3.98 GHz	100% 3.98 GHz	100% 3.98 GHz	100% 3.98 GHz	100% 3.98 GHz

CPU

Intel(R) Core(TM) i7-10750H CPU…

Total:	100%
User:	93%
System:	7%

Top processes

cc1	78.4%
cc1	60.1%
cc1	60.0%
cc1	57.1%
cc1	41.9%

GPUs

Intel CometLake-H GT2 [UHD Graphics…
NVIDIA TU117GLM [Quadro T1000 Mo…

System uptime

3d 23h 30m 03s

Resources:

- ✔ *How to Build Linux Kernel From Scratch {Step-By-Step Guide}, Nov 2020 [builds the recent 6.0.7 kernel!]*
- ✔ *How to compile and install Linux Kernel 5.6.9 from source code* Author: Vivek Gite Last updated: May 2, 2020
- ✔ How to Compile Linux Kernel from Source to Build Custom Kernel
- ✔ 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]]()

Quick Tips

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

CONFIGURING the kernel:

> Do not skip this step even if you are only upgrading one minor version. New configuration options are added in each release, and odd problems will turn up if the configuration files are not set up as expected. **If you want to carry your existing configuration to a new version with minimal work, use "make oldconfig", which will only ask you for the answers to new questions.**

```
cd <kernel-src-tree>
make help
…
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
[...]
```

<<

FYI

Android (AOSP) kernel configuration defaults:
https://android.googlesource.com/kernel/configs/

Also, **clang** (LLVM) is used to build the AOSP (for all- kernel, modules and the root filesystem); to build the Android kernel:

```
make LLVM=1 LLVM_IAS=1 all
```

>>

------------------- 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):

Approach 1

Simple approach: simply copy in the existing kernel config, of the kernel you're currently running and use it as a starting point:

```
cp /boot/config-$(uname -r) .config
```

Tip:

Check which kernel configs exist under /boot and copy the one closest to the kernel version you're now going to build:

```
$ ls /boot/config-*
/boot/config-5.10.140-lkd-kernel  /boot/config-6.1.0-lkp2e-01+
/boot/config-6.1.11-lkp-kernel    /boot/config-5.15.0-43-generic
/boot/config-6.1.0-lkp2e-01+.old
```

So, if we're building, say, 6.1.23 kernel, copy the */boot/config-6.1.11-lkp-kernel* to *.config* and then proceed!

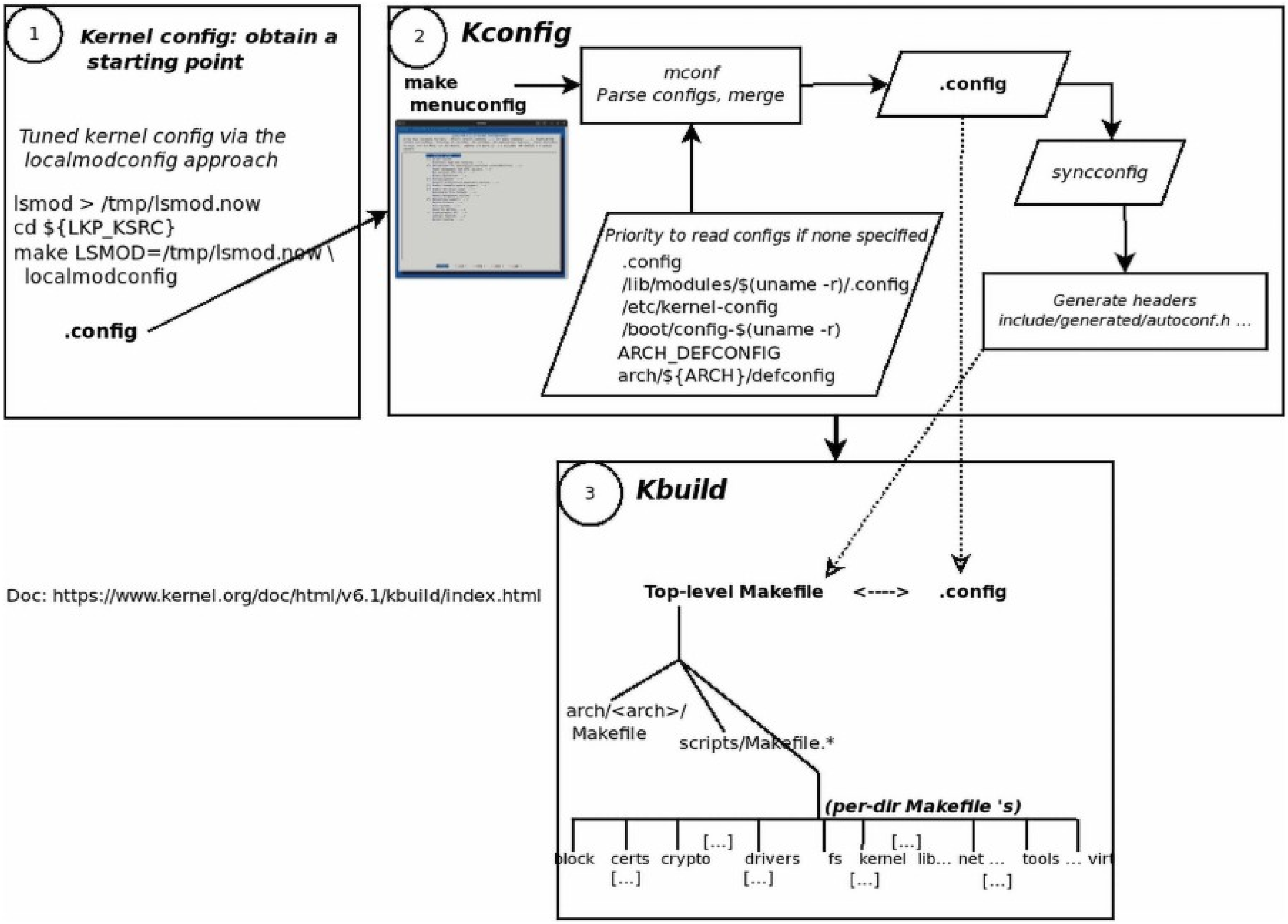
Approach 2

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.
...
```

<<
Src: LKP 2E book

A diagram (inspired by Cao Jin's articles) that attempts to communicate the kernel's Kconfig/Kbuild system is shown here. The diagram conveys more information than has been covered by now; worry not, we'll get to it.



```
lsmod > /tmp/lsmod.now
cd ${LKP_KSRC}
make LSMOD=/tmp/lsmod.now \
  localmodconfig
```

Figure 2.8: The kernel's Kconfig/Kbuild system in a simplified form

`* syncconfig:` "… a hidden target called *syncconfig*, which was earlier misnamed silentoldconfig. This target has Kconfig generate a few header files that are further used in the setup to build the kernel. These files include some meta-headers under *include/config*, as well as the *include/generated/autoconf.h* header file, which stores the kernel config as C macros, thus enabling both the kernel Makefiles and kernel code to make decisions based on whether a kernel feature is available."
>>

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

With kernel ver >= 5.10 kernel builds FAIL with:

```
make
...
  AS      certs/system_certificates.o
make[1]: *** No rule to make target 'debian/canonical-revoked-certs.pem', needed by 'certs/x509_revocation_list'.  Stop.
make[1]: *** Waiting for unfinished jobs....
  CC      certs/blacklist.o
...
```

Tip:
https://askubuntu.com/questions/1362455/i-am-installing-kernel-in-my-ubuntu-but-getting-an-error

Do this

```
scripts/config --disable SYSTEM_REVOCATION_KEYS
scripts/config --disable SYSTEM_TRUSTED_KEYS
```

Once these configs are disabled, it builds perfectly.

On Fedora, same thing, plus ensure that the *openssl* package is installed.

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

>= 5.10 (or so) kernel build FAILS with:
`pahole required ...`

This is due to the kernel config CONFIG_DEBUG_INFO_BTF being turned On. To fix this, turn it Off.

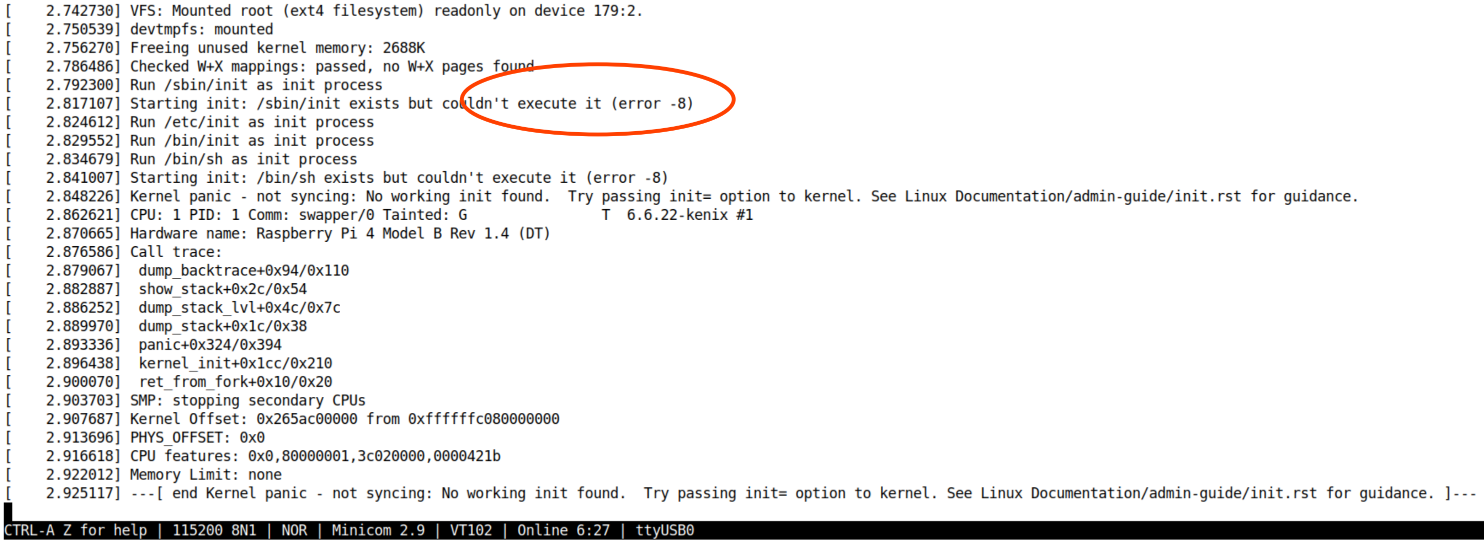
```
Kernel Hacking
      Compile-time checks and compiler options
            Generate BTF typeinfo
```

NOTE- pahole (ver >= 1.16) being installed prevents this issue (but Ubuntu 20.04 and lower don’t have a recent enough ver of pahole).

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

Builds, boots, mounts the root filesystem BUT fails to run any 'init' program!

(This happened when working on an embedded Linux project using Yocto:



```
[    2.742730] VFS: Mounted root (ext4 filesystem) readonly on device 179:2.
[    2.750539] devtmpfs: mounted
[    2.756270] Freeing unused kernel memory: 2688K
[    2.786486] Checked W+X mappings: passed, no W+X pages found
[    2.792300] Run /sbin/init as init process
[    2.817107] Starting init: /sbin/init exists but couldn't execute it (error -8)
[    2.824612] Run /etc/init as init process
[    2.829552] Run /bin/init as init process
[    2.834679] Run /bin/sh as init process
[    2.841007] Starting init: /bin/sh exists but couldn't execute it (error -8)
[    2.848226] Kernel panic - not syncing: No working init found.  Try passing init= option to kernel. See Linux Documentation/admin-guide/init.rst for guidance.
[    2.862621] CPU: 1 PID: 1 Comm: swapper/0 Tainted: G                T  6.6.22-kenix #1
[    2.870665] Hardware name: Raspberry Pi 4 Model B Rev 1.4 (DT)
[    2.876586] Call trace:
[    2.879067]  dump_backtrace+0x94/0x110
[    2.882887]  show_stack+0x2c/0x54
[    2.886252]  dump_stack_lvl+0x4c/0x7c
[    2.889970]  dump_stack+0x1c/0x38
[    2.893336]  panic+0x324/0x394
[    2.896438]  kernel_init+0x1cc/0x210
[    2.900070]  ret_from_fork+0x10/0x20
[    2.903703] SMP: stopping secondary CPUs
[    2.907687] Kernel Offset: 0x265ac00000 from 0xffffffc080000000
[    2.913696] PHYS_OFFSET: 0x0
[    2.916618] CPU features: 0x0,80000001,3c020000,0000421b
[    2.922012] Memory Limit: none
[    2.925117] ---[ end Kernel panic - not syncing: No working init found.  Try passing init= option to kernel. See Linux Documentation/admin-guide/init.rst for guidance. ]---
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.9 | VT102 | Online 6:27 | ttyUSB0
```

The root cause :
i looked up errno 8 : it's this: #define ENOEXEC 8 /* Exec format error */

Made me realize that while performing kernel hardening, I inadvertently left out a key config (CONFIG_BINFMT_ELF); once added, it worked fine.)

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

Configure the kernel carefully!

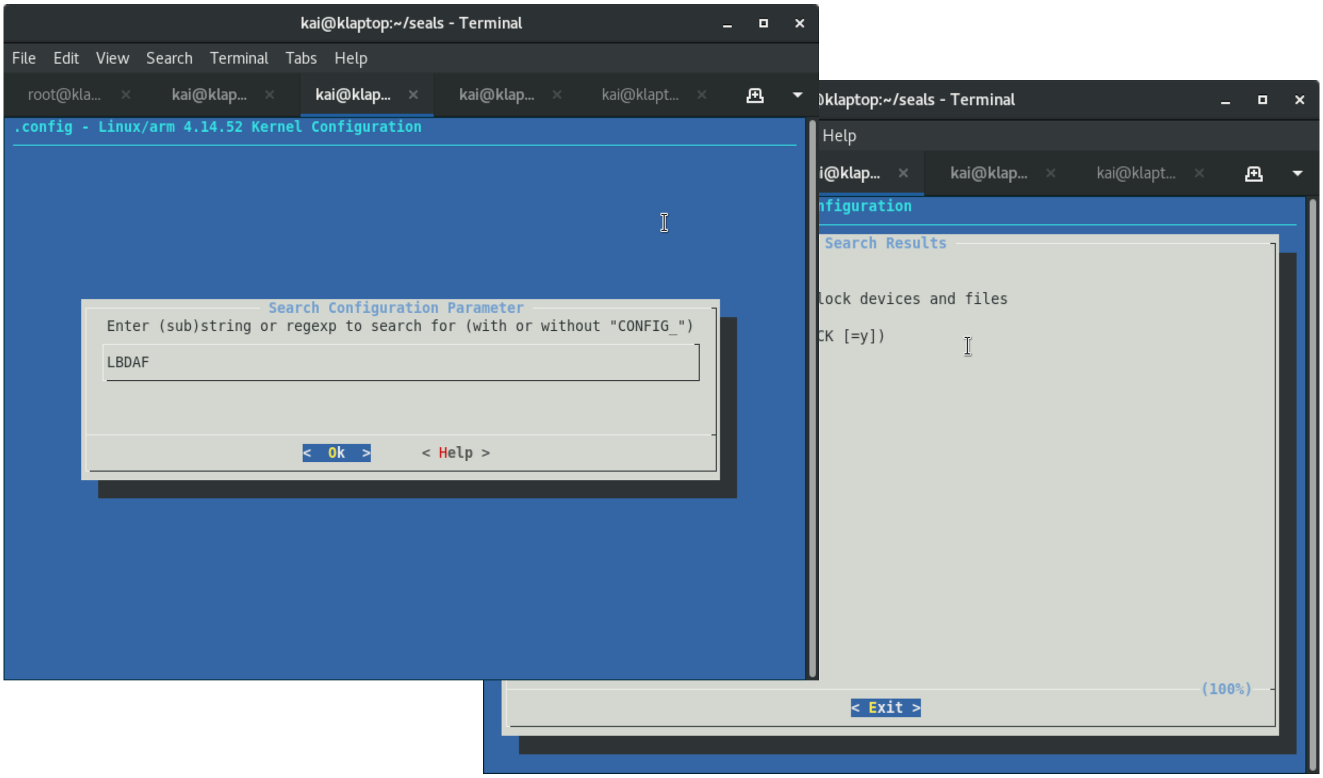
On an ARM system:

```
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 ---

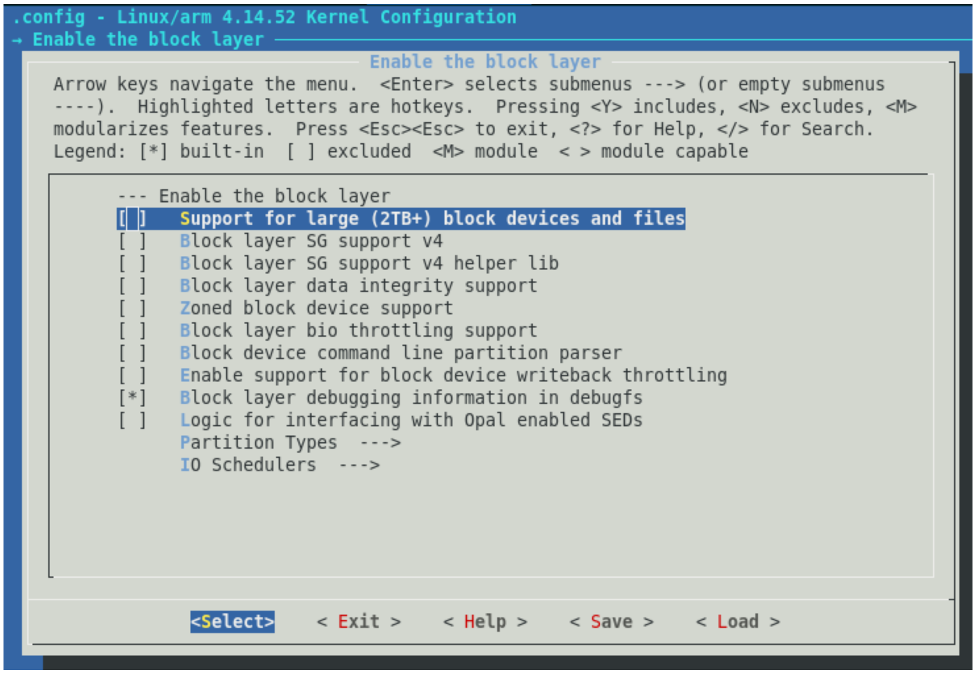
So, where in the *make menuconfig* menu is this CONFIG_LBDAF kernel config option?
Search for it with the “usual” **“/”** operator!


kai@klaptop:~/seals - Terminal
File Edit View Search Terminal Tabs Help
root@kla... kai@klap... kai@klap... kai@klap... kai@klapt...
)klaptop:~/seals - Terminal
Help
i@klap... kai@klap... kai@klapt...

```
.config - Linux/arm 4.14.52 Kernel Configuration

 Search Configuration Parameter
 Enter (sub)string or regexp to search for (with or without "CONFIG_")
 LBDAF

            < Ok >      < Help >
```

```
nfiguration
 Search Results

lock devices and files

CK [=y])

                                   (100%)
            < Exit >
```

Ah, we can now see: its under the menu item *“Enable the block layer”*.
So navigate there:



```
.config - Linux/arm 4.14.52 Kernel Configuration
→ Enable the block layer
                       Enable the block layer
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
  ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M>
  modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

      --- Enable the block layer
      [ ]   Support for large (2TB+) block devices and files
      [ ]   Block layer SG support v4
      [ ]   Block layer SG support v4 helper lib
      [ ]   Block layer data integrity support
      [ ]   Zoned block device support
      [ ]   Block layer bio throttling support
      [ ]   Block device command line partition parser
      [ ]   Enable support for block device writeback throttling
      [*]   Block layer debugging information in debugfs
      [ ]   Logic for interfacing with Opal enabled SEDs
            Partition Types  --->
            IO Schedulers  --->

        <Select>   < Exit >   < Help >   < Save >   < Load >
```

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

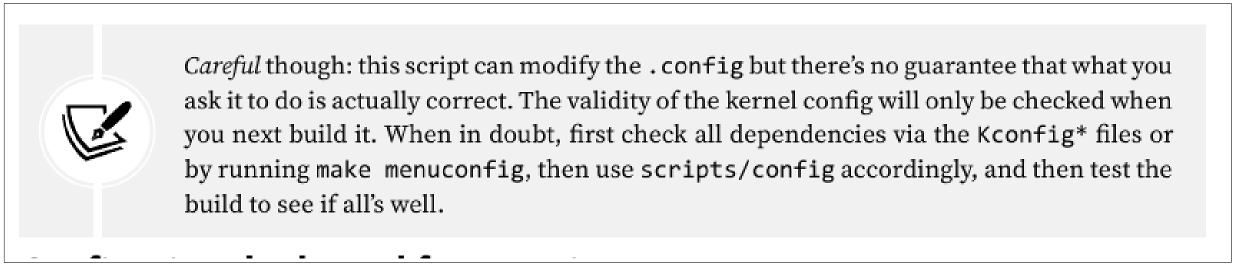
© 2000-2024 Kaiwan N Billimoria, kaiwanTECH

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

Configuring the kernel non-interactively
Use the ***scripts/config*** script !

The help screen shows the options:

```
$ scripts/config
Manipulate options in a .config file from the command line.
Usage:
config options command ...
commands:
      --enable|-e option   Enable option
      --disable|-d option  Disable option
      --module|-m option   Turn option into a module
      --set-str option string
                           Set option to "string"
      --set-val option value
                           Set option to value
      --undefine|-u option Undefine option
      --state|-s option    Print state of option (n,y,m,undef)

      --enable-after|-E beforeopt option
                             Enable option directly after other option
      --disable-after|-D beforeopt option
                             Disable option directly after other option
      --module-after|-M beforeopt option
                             Turn option into module directly after other option

      commands can be repeated multiple times

options:
      --file config-file   .config file to change (default .config)
      --keep-case|-k       Keep next symbols' case (dont' upper-case it)

config doesn't check the validity of the .config file. This is done at next
make time.

By default, config will upper-case the given symbol. Use --keep-case to keep
the case of all following symbols unchanged.

config uses 'CONFIG_' as the default symbol prefix. Set the environment
variable CONFIG_ to the prefix to use. Eg.: CONFIG_="FOO_" config ...
$
```



Careful though: this script can modify the `.config` but there's no guarantee that what you ask it to do is actually correct. The validity of the kernel config will only be checked when you next build it. When in doubt, first check all dependencies via the `Kconfig*` files or by running `make menuconfig`, then use `scripts/config` accordingly, and then test the build to see if all's well.

------------------- *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* --

The kernel build fails with:

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

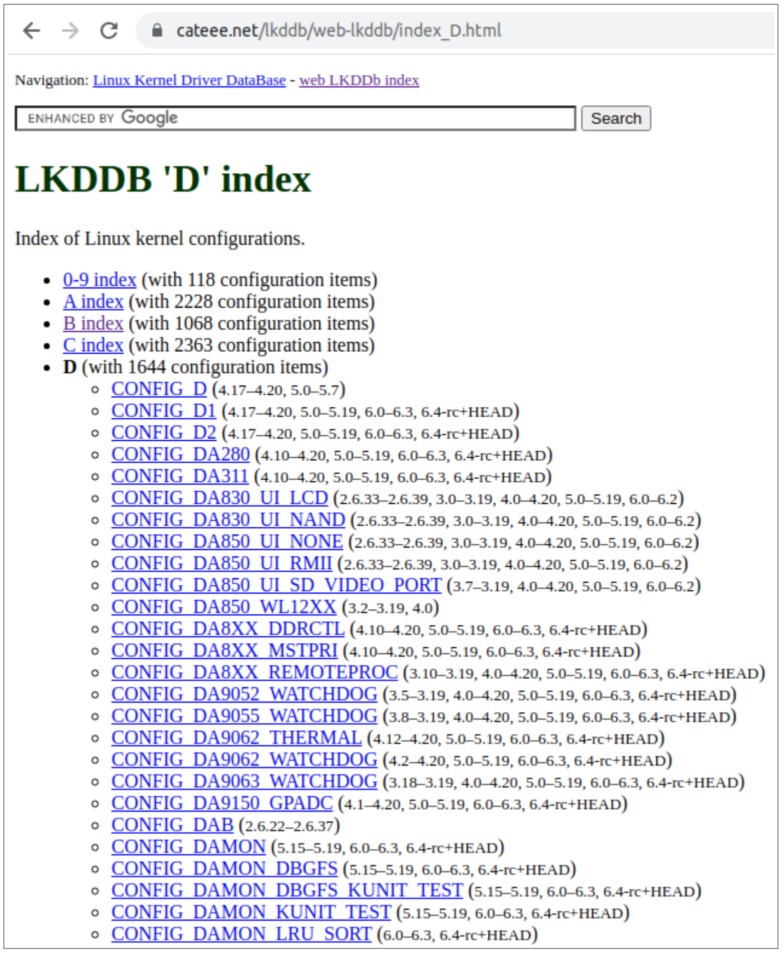
See: *OpenSSL missing during ./configure. How to fix?*
[…]
The OpenSSL library is usually already installed, but you have to install the header files. Depending on your Linux distribution, you'll need these packages:

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

------------------- *TIP* --

Q. Are all kernel configurables (CONFIG_FOO) shown in one place, along with which kernel versions they're supported on?
A. Yes!
https://cateee.net/lkddb/web-lkddb/

Eg. screenshot:



Navigation: Linux Kernel Driver DataBase - web LKDDb index

LKDDB 'D' index

Index of Linux kernel configurations.

- 0-9 index (with 118 configuration items)
- A index (with 2228 configuration items)
- B index (with 1068 configuration items)
- C index (with 2363 configuration items)
- **D** (with 1644 configuration items)
  - CONFIG_D (4.17–4.20, 5.0–5.7)
  - CONFIG_D1 (4.17–4.20, 5.0–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_D2 (4.17–4.20, 5.0–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DA280 (4.10–4.20, 5.0–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DA311 (4.10–4.20, 5.0–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DA830_UI_LCD (2.6.33–2.6.39, 3.0–3.19, 4.0–4.20, 5.0–5.19, 6.0–6.2)
  - CONFIG_DA830_UI_NAND (2.6.33–2.6.39, 3.0–3.19, 4.0–4.20, 5.0–5.19, 6.0–6.2)
  - CONFIG_DA850_UI_NONE (2.6.33–2.6.39, 3.0–3.19, 4.0–4.20, 5.0–5.19, 6.0–6.2)
  - CONFIG_DA850_UI_RMII (2.6.33–2.6.39, 3.0–3.19, 4.0–4.20, 5.0–5.19, 6.0–6.2)
  - CONFIG_DA850_UI_SD_VIDEO_PORT (3.7–3.19, 4.0–4.20, 5.0–5.19, 6.0–6.2)
  - CONFIG_DA850_WL12XX (3.2–3.19, 4.0)
  - CONFIG_DA8XX_DDRCTL (4.10–4.20, 5.0–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DA8XX_MSTPRI (4.10–4.20, 5.0–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DA8XX_REMOTEPROC (3.10–3.19, 4.0–4.20, 5.0–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DA9052_WATCHDOG (3.5–3.19, 4.0–4.20, 5.0–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DA9055_WATCHDOG (3.8–3.19, 4.0–4.20, 5.0–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DA9062_THERMAL (4.12–4.20, 5.0–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DA9062_WATCHDOG (4.2–4.20, 5.0–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DA9063_WATCHDOG (3.18–3.19, 4.0–4.20, 5.0–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DA9150_GPADC (4.1–4.20, 5.0–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DAB (2.6.22–2.6.37)
  - CONFIG_DAMON (5.15–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DAMON_DBGFS (5.15–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DAMON_DBGFS_KUNIT_TEST (5.15–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DAMON_KUNIT_TEST (5.15–5.19, 6.0–6.3, 6.4-rc+HEAD)
  - CONFIG_DAMON_LRU_SORT (6.0–6.3, 6.4-rc+HEAD)

------------------- 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*** --

Generating the kernel in a package format

```
make help
```

```
…
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
  snap-pkg            - Build only the binary kernel snap package
                               (will connect to external hosts)
…
```

Say, you want the RPM package format for the source and binary kernel packages:
Requires *rpmbuild* :

```
sudo apt install rpm
```

Generate them:

```
make [-jn] rpm-pkg
…
```

(For the Debian package format:

```
make [-jn] deb-pkg
…
).
```

Alternately, one can always supply a custom script that builds a custom package.

------------------- ***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*** ---

Accurately Measuring / Benchmarking kernel build times:

See this detailed post by Ingo Molnar, May 2018:
https://lkml.org/lkml/2018/5/2/74

© 2000-2024 Kaiwan N Billimoria, kaiwanTECH

------------------ ***TIP*** ---

Configuring the Linux kernel as RTL – Real Time Linux – i.e. Linux as an RTOS!

1. Download and extract the kernel source tree (here, as an example, we're using the 5.10.152 kernel):
   `wget https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.10.152.tar.xz`
2. Download the kernel RTL patch for this source tree from here:
   https://mirrors.edge.kernel.org/pub/linux/kernel/projects/rt/

   F.e.:
   wget https://mirrors.edge.kernel.org/pub/linux/kernel/projects/rt/5.10/patch-5.10.152-rt75.patch.xz

NOTE!

- *The particular RTL kernel patch version may differ from what's shown here*
- *you'll need to exactly match the vanilla kernel source source version and the RTL kernel patch version*

3. Extract it (in the dir above the 5.10.106 source)

4. Apply the patch…

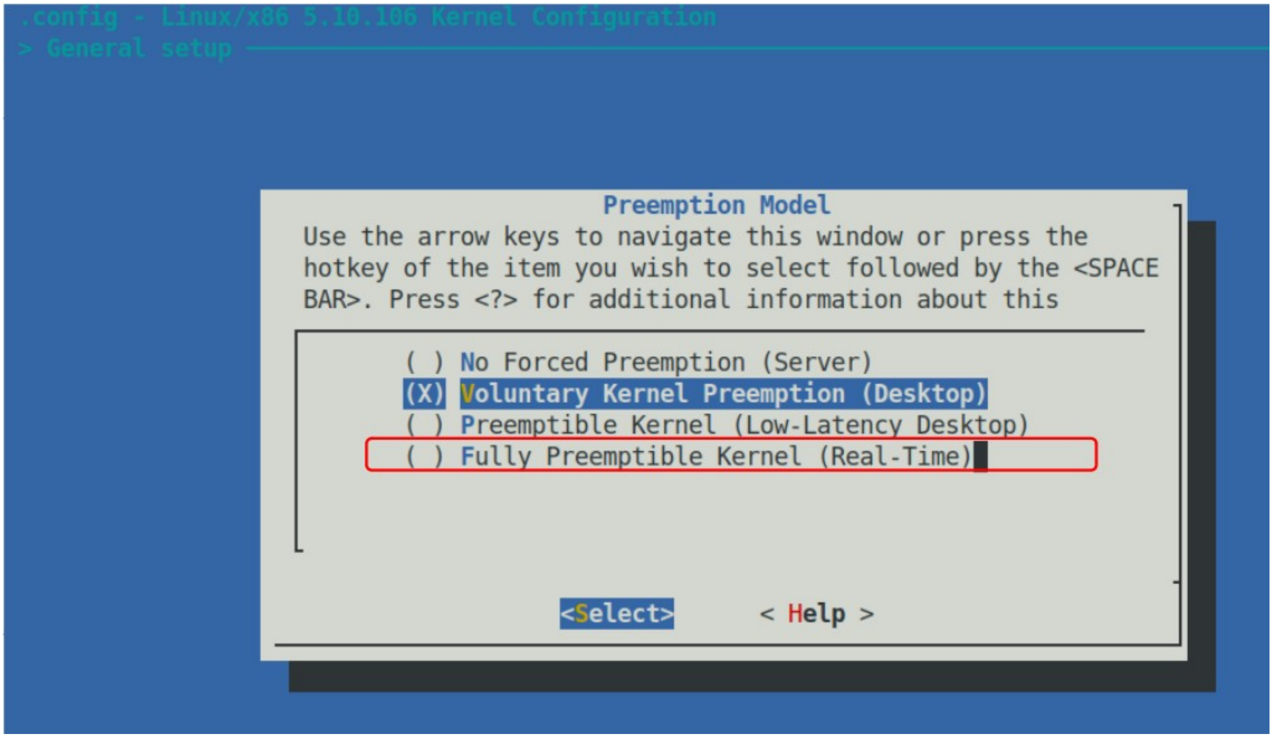
   ```
   cd linux-5.10.106
   patch -p1  < ../patch-5.10.106-rt64.patch
   ```
   (Verify it applies cleanly; can use the `--dry-run` switch to patch first to check)

5. Run make menuconfig; goto the *General Setup* → *Preeption Model* menu: select the last one – *Fully preemtible kernel (Real-Time)* : **`CONFIG_PREEMPT_RT=y`**

<<

UPDATE !

In September 2024, the RTL patch has been fully merged into the 6.11 kernel!
Details: *The Linux kernel now with RTL fully merged*
>>



```
.config - Linux/x86 5.10.106 Kernel Configuration
> General setup
                          Preemption Model
 Use the arrow keys to navigate this window or press the
 hotkey of the item you wish to select followed by the <SPACE
 BAR>. Press <?> for additional information about this

        ( ) No Forced Preemption (Server)
        (X) Voluntary Kernel Preemption (Desktop)
        ( ) Preemptible Kernel (Low-Latency Desktop)
        ( ) Fully Preemptible Kernel (Real-Time)

                 <Select>      < Help >
```

Make any other required mods to the kernel config, save and exit.
Build and boot from the kernel in the usual way; ***you're now running Linux as an RTOS!***

------------------ ***TIP*** ---

- A simple convenience script for kernel build (lightly tested… YMMV!):
https://github.com/PacktPublishing/Linux-Kernel-Programming_2E/blob/main/ch3/kbuild.sh

------------------ ***TIP*** ---

How can I check which distro kernels are installed?
Deb/Ubuntu:

```
dpkg --list | grep linux-image
```

RedHat/Fedora/…

```
rpm -qa kernel
```

or

```
dnf list installed kernel
```

Ref-
https://www.cyberciti.biz/faq/howto-display-all-installed-linux-kernel-version/

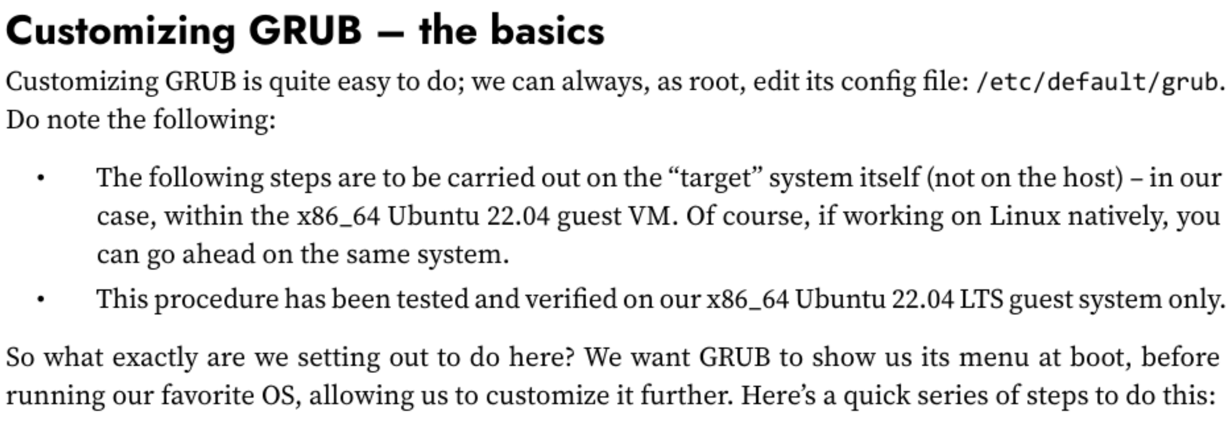
<< These are package-based kernels- they won't include 'manually' built ones of course! >>

© 2000-2024 Kaiwan N Billimoria, kaiwanTECH

------------------- ***TIP*** --

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

GRUB bootloader (basic) customizations:
(Materials below from my LKP 2E book):

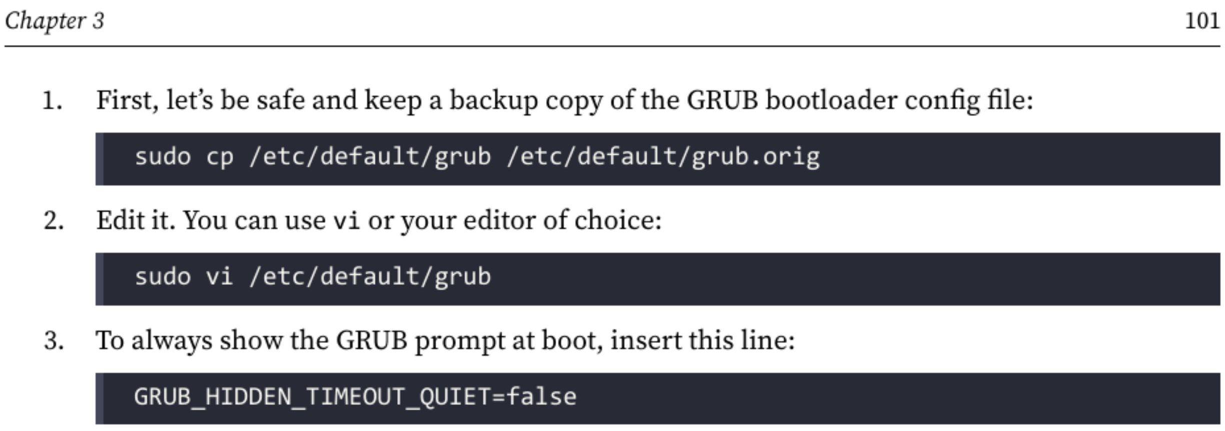


Customizing GRUB – the basics

Customizing GRUB is quite easy to do; we can always, as root, edit its config file: `/etc/default/grub`. Do note the following:

- The following steps are to be carried out on the "target" system itself (not on the host) – in our case, within the x86_64 Ubuntu 22.04 guest VM. Of course, if working on Linux natively, you can go ahead on the same system.
- This procedure has been tested and verified on our x86_64 Ubuntu 22.04 LTS guest system only.

So what exactly are we setting out to do here? We want GRUB to show us its menu at boot, before running our favorite OS, allowing us to customize it further. Here's a quick series of steps to do this:



Chapter 3 101

1. First, let's be safe and keep a backup copy of the GRUB bootloader config file:

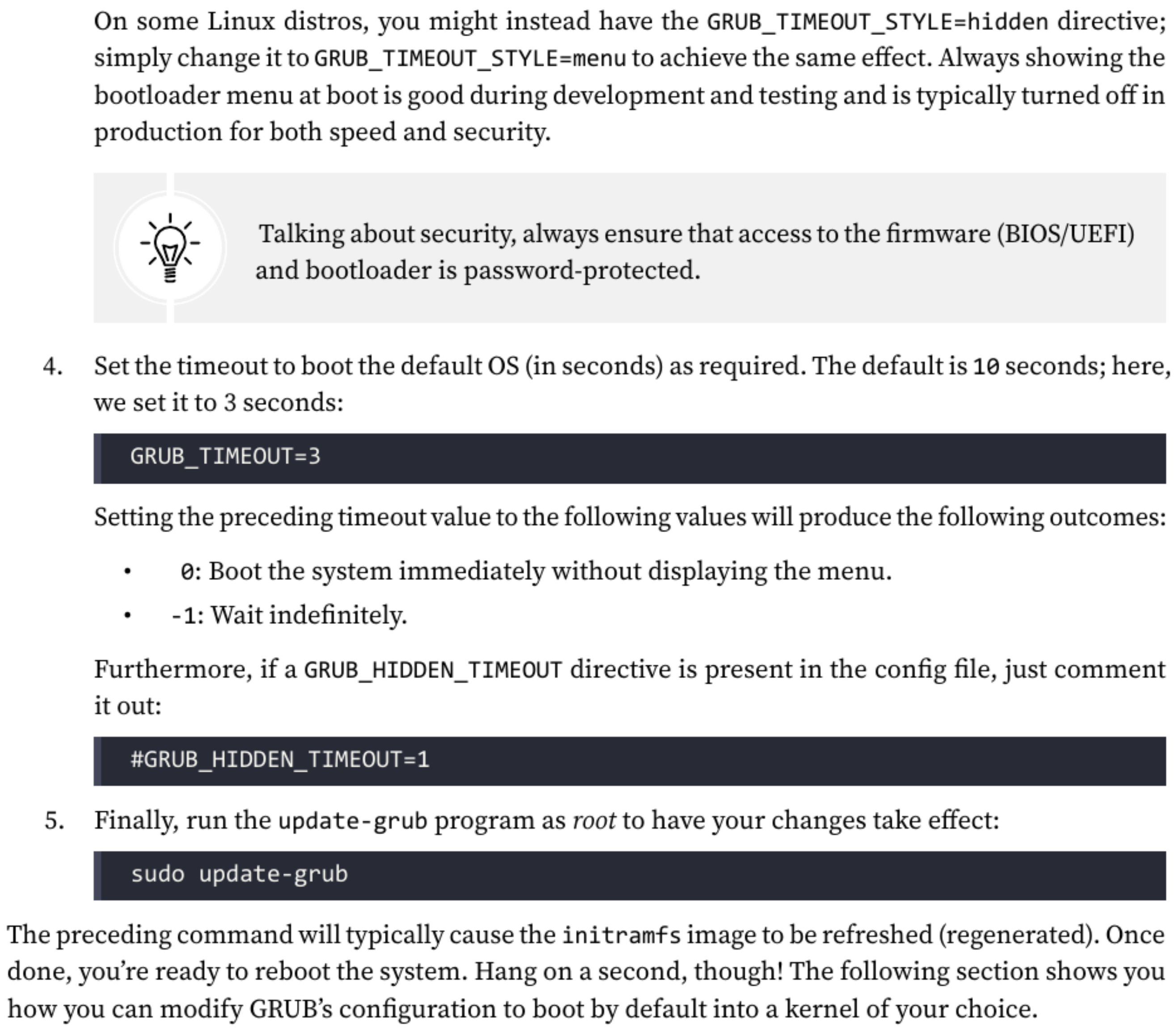
```
sudo cp /etc/default/grub /etc/default/grub.orig
```

2. Edit it. You can use `vi` or your editor of choice:

```
sudo vi /etc/default/grub
```

3. To always show the GRUB prompt at boot, insert this line:

```
GRUB_HIDDEN_TIMEOUT_QUIET=false
```



On some Linux distros, you might instead have the `GRUB_TIMEOUT_STYLE=hidden` directive; simply change it to `GRUB_TIMEOUT_STYLE=menu` to achieve the same effect. Always showing the bootloader menu at boot is good during development and testing and is typically turned off in production for both speed and security.

> Talking about security, always ensure that access to the firmware (BIOS/UEFI) and bootloader is password-protected.

4. Set the timeout to boot the default OS (in seconds) as required. The default is `10` seconds; here, we set it to 3 seconds:

```
GRUB_TIMEOUT=3
```

Setting the preceding timeout value to the following values will produce the following outcomes:

- `0`: Boot the system immediately without displaying the menu.
- `-1`: Wait indefinitely.

Furthermore, if a `GRUB_HIDDEN_TIMEOUT` directive is present in the config file, just comment it out:

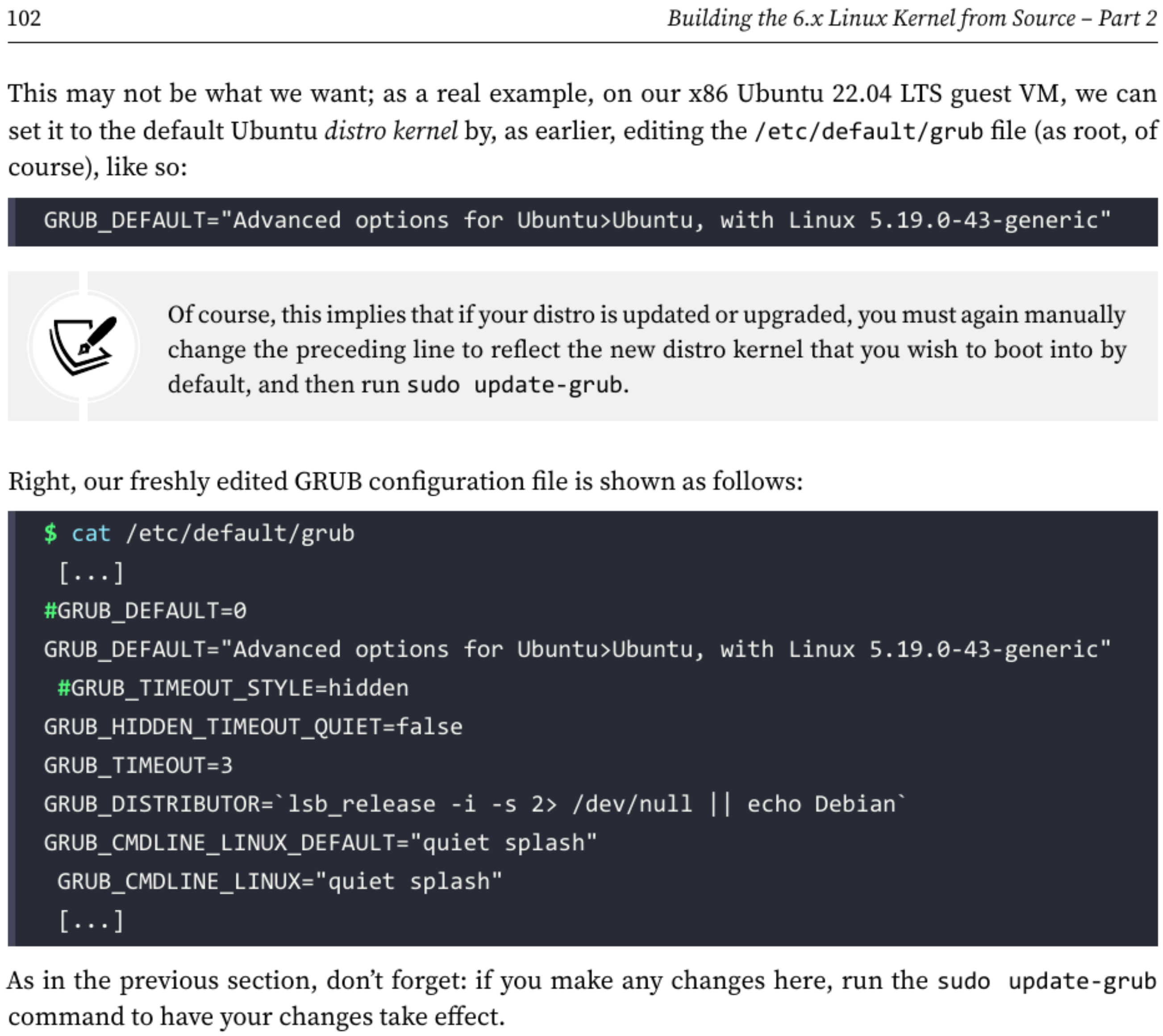
```
#GRUB_HIDDEN_TIMEOUT=1
```

5. Finally, run the `update-grub` program as *root* to have your changes take effect:

```
sudo update-grub
```

The preceding command will typically cause the `initramfs` image to be refreshed (regenerated). Once done, you're ready to reboot the system. Hang on a second, though! The following section shows you how you can modify GRUB's configuration to boot by default into a kernel of your choice.

…



102 *Building the 6.x Linux Kernel from Source – Part 2*

This may not be what we want; as a real example, on our x86 Ubuntu 22.04 LTS guest VM, we can set it to the default Ubuntu *distro kernel* by, as earlier, editing the `/etc/default/grub` file (as root, of course), like so:

```
GRUB_DEFAULT="Advanced options for Ubuntu>Ubuntu, with Linux 5.19.0-43-generic"
```

> Of course, this implies that if your distro is updated or upgraded, you must again manually change the preceding line to reflect the new distro kernel that you wish to boot into by default, and then run `sudo update-grub`.

Right, our freshly edited GRUB configuration file is shown as follows:

```
$ cat /etc/default/grub
 [...]
#GRUB_DEFAULT=0
GRUB_DEFAULT="Advanced options for Ubuntu>Ubuntu, with Linux 5.19.0-43-generic"
 #GRUB_TIMEOUT_STYLE=hidden
GRUB_HIDDEN_TIMEOUT_QUIET=false
GRUB_TIMEOUT=3
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
 GRUB_CMDLINE_LINUX="quiet splash"
 [...]
```

As in the previous section, don't forget: if you make any changes here, run the `sudo update-grub` command to have your changes take effect.

…

To see the GRUB menu at boot:

- Depending on your UEFI (or legacy BIOS), the boot order can differ; so press F12 (or perhaps F1/F2) to see all available options
- Fedora: When Linux is selected to boot, keep the Shift key(s) pressed; only then does the GRUB menu seem to appear (in spite of the changes done above!)

Fedora:
No update-grub command; the equivalent is this:
`# grub2-mkconfig --output=/boot/grub2/grub.cfg`

© 2000-2024 Kaiwan N Billimoria, kaiwanTECH

<<
SECURITY ALERT

With GRUB, one can do this:

- navigate to 'Advanced options for Ubuntu'
- select a kernel to boot from (just highlight that line)
- type 'e' to edit
- edit the 'linux ...' cmdline, adding '1' to boot into single-user / rescue mode
    - by default, you get dropped to a *root shell* (after pressing [Enter] when asked) with no password!

So how can we make this safer?

ChatGPT 4 answer to this:

"Booting Linux in single-user mode (`runlevel 1` or `rescue.target`) often bypasses normal multi-user authentication, meaning the system might not prompt for a root password by default. To ensure that the root password is required when entering single-user mode, you can modify the system's configuration. Here's how to do it:

### 1. **Secure `single-user` mode in `systemd`**

Modern Linux distributions use `systemd`, where single-user mode corresponds to `rescue.target`. Here's how to enforce a password:

#### a. Modify the Rescue Service Configuration

Edit the systemd unit file for `rescue`:

```bash
sudo systemctl edit rescue.service
```

Add the following lines (if they don't already exist):

```ini
[Service]
ExecStart=
ExecStart=-/usr/lib/systemd/systemd-sulogin-shell rescue
```

#### b. Modify the Emergency Service Configuration

Similarly, for emergency mode (`emergency.target`):

```bash
sudo systemctl edit emergency.service
```

Add the same lines as above:

```ini
[Service]
ExecStart=
ExecStart=-/usr/lib/systemd/systemd-sulogin-shell emergency
```

#### c. Enable Password Protection
Ensure that password protection is enforced by setting the `SULOGIN` environment variable. Edit the `systemd` configuration:
```bash
sudo nano /etc/systemd/system.conf
```
Look for or add the following line:
```ini
SULOGIN_FORCE=1
```

#### d. Reload systemd and reboot
Apply your changes:
```bash
sudo systemctl daemon-reexec
sudo reboot
```

### 2. **Legacy Approach for Non-systemd Systems**
If you are using an older distribution or one without `systemd`:
- Modify the kernel parameters in your bootloader (`GRUB`):
  - Ensure that the `init` process points to `/bin/sulogin` or `/sbin/sulogin` for single-user mode.

### 3. **Secure GRUB to Prevent Unauthorized Edits**
If an attacker has physical access to the system, they could edit the GRUB command line to disable the password prompt. To prevent this, secure GRUB with a password.

#### a. Generate a Password
```bash
grub2-mkpasswd-pbkdf2
```
Note the hash generated.

#### b. Edit the GRUB Configuration
Open the GRUB config file:

```bash
sudo nano /etc/grub.d/40_custom
```
Add a section like this:
```bash
set superusers="root"
password_pbkdf2 root <hashed-password>
```
Replace `<hashed-password>` with the hash you generated.

#### c. Update GRUB
Apply changes:
```bash
sudo update-grub
```

### 4. **Test the Configuration**
Reboot the system, choose single-user mode, and verify that the root password is required.

### Summary
- Modify `rescue.service` and `emergency.service` to invoke `sulogin`.
- Enable `SULOGIN_FORCE=1` in `/etc/systemd/system.conf`.
- Secure GRUB to prevent bypassing these configurations."

>>

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

Security for the Linux kernel:
Ref: How To Secure the Linux Kernel, B Day, Jan 2022

------------------ 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.

$
```

Run make in verbose mode by passing the V=1 option:

```
$ 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
```

© 2000-2024 Kaiwan N Billimoria, kaiwanTECH

...

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!

Using a cross-compiler here

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
```

TIP

The kernel config is sometimes available within the proc filesystem, as the pseudofile */proc/config.gz* – a gzip'ed config file.

If it doesn't show up, try this:

```
# modprobe configs
# ls -l /proc/config.gz
-r--r--r-- 1 root root 44216 Jun 23 10:13 /proc/config.gz
# zcat /proc/config.gz |grep INITRD
...
```

<<

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_size(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

An ‘initrd’ – now called initramfs – is a block device in RAM (a ramdisk).

initrd – initial RAM disk / initramfs :

Difference between initrd and initramfs?

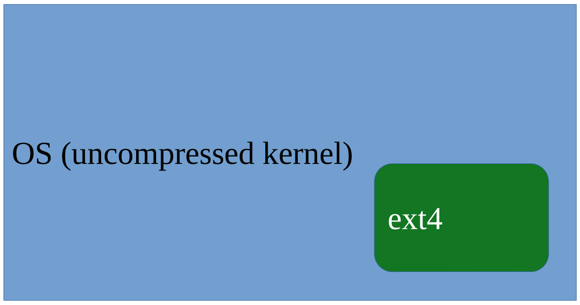
- initrd is the older style tech
- to build an initrd image with cur dir content, can do:
```
find . | sudo cpio -R root:root | gzip -9 > initrd.img
```
- to build an initramfs image with cur dir content, can do (uses the ‘newc’ format):
```
find . | sudo cpio -o --format=newc -R root:root | gzip -9 > initramfs.img
```

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

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

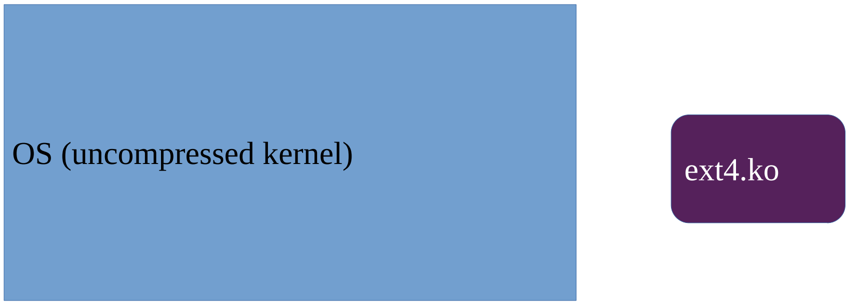
ext4 ; fs ‘driver’

Builtin:



mount *will succeed*

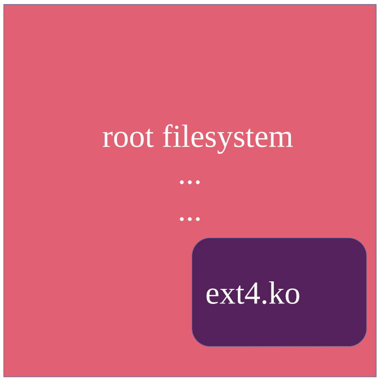
Else, it’s a kernel module:



Now, 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/ - this location is **within** the root filesystem !

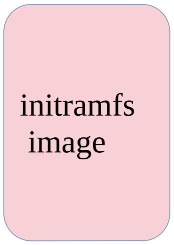


In order to mount the rootfs, we require the *ext4.ko* module
BUT it's in the rootfs !

Thus we have a classic 'Chicken and Egg' problem !!!

How to solve this??

initramfs !



Recall, an 'initrd' – now called initramfs –
is a block device in RAM (a ramdisk).

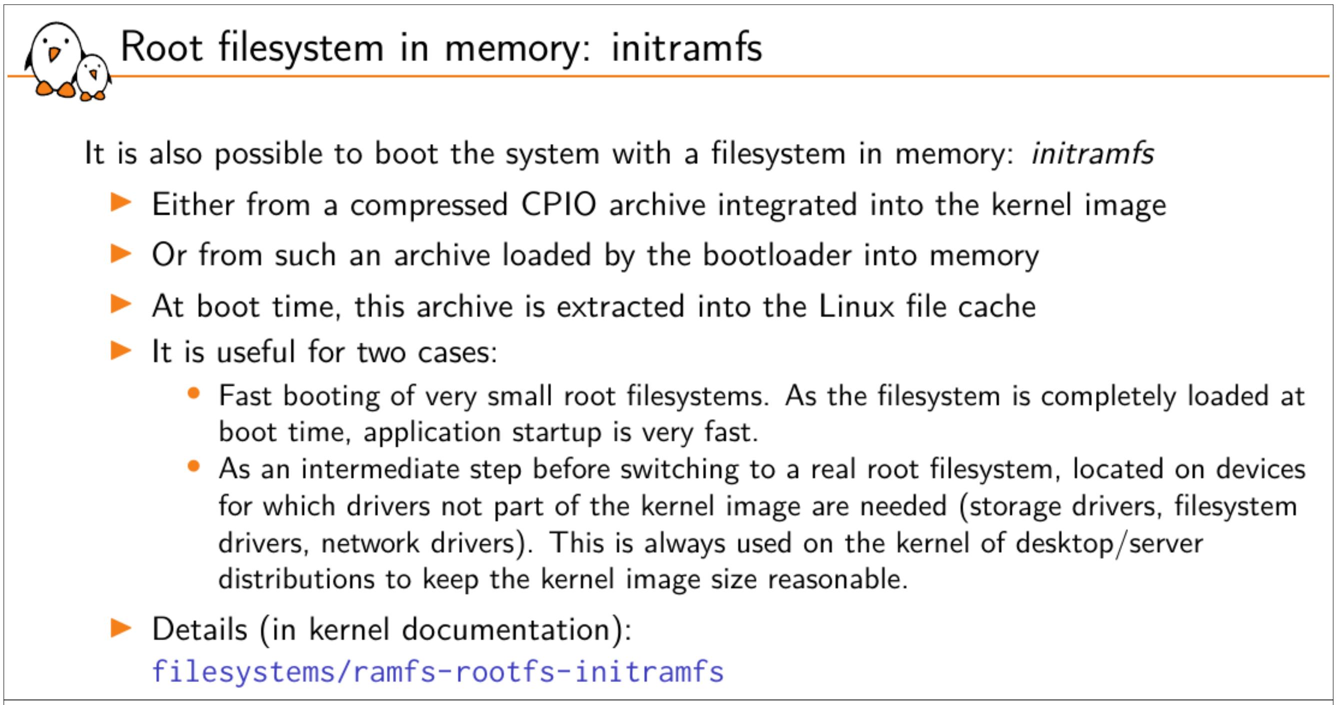
Contents of / (rootfs) are scaled-down;
/bin; /dev/ ; /etc/ ; /lib ; /usr ; ...

Essentially:

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

<<

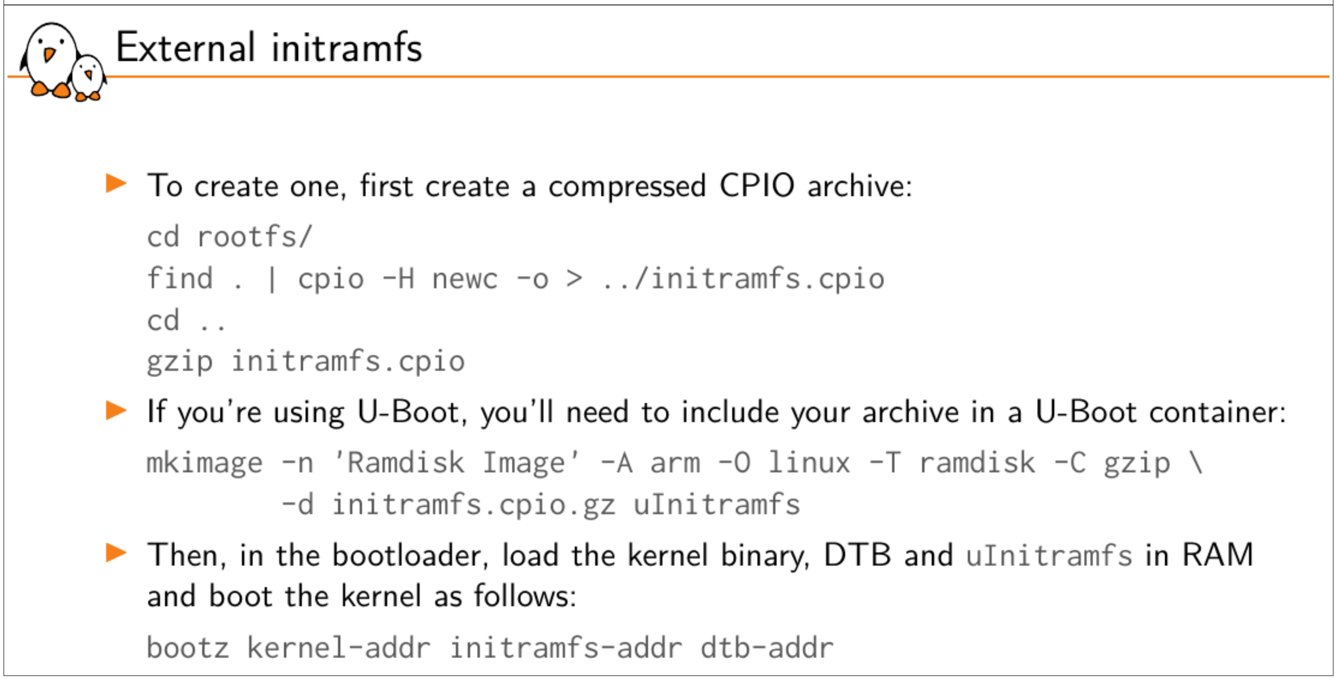
From the excellent material from Bootlin:



Root filesystem in memory: initramfs

It is also possible to boot the system with a filesystem in memory: *initramfs*

- Either from a compressed CPIO archive integrated into the kernel image
- Or from such an archive loaded by the bootloader into memory
- At boot time, this archive is extracted into the Linux file cache
- It is useful for two cases:
  - Fast booting of very small root filesystems. As the filesystem is completely loaded at boot time, application startup is very fast.
  - As an intermediate step before switching to a real root filesystem, located on devices for which drivers not part of the kernel image are needed (storage drivers, filesystem drivers, network drivers). This is always used on the kernel of desktop/server distributions to keep the kernel image size reasonable.
- Details (in kernel documentation): `filesystems/ramfs-rootfs-initramfs`



External initramfs

- To create one, first create a compressed CPIO archive:

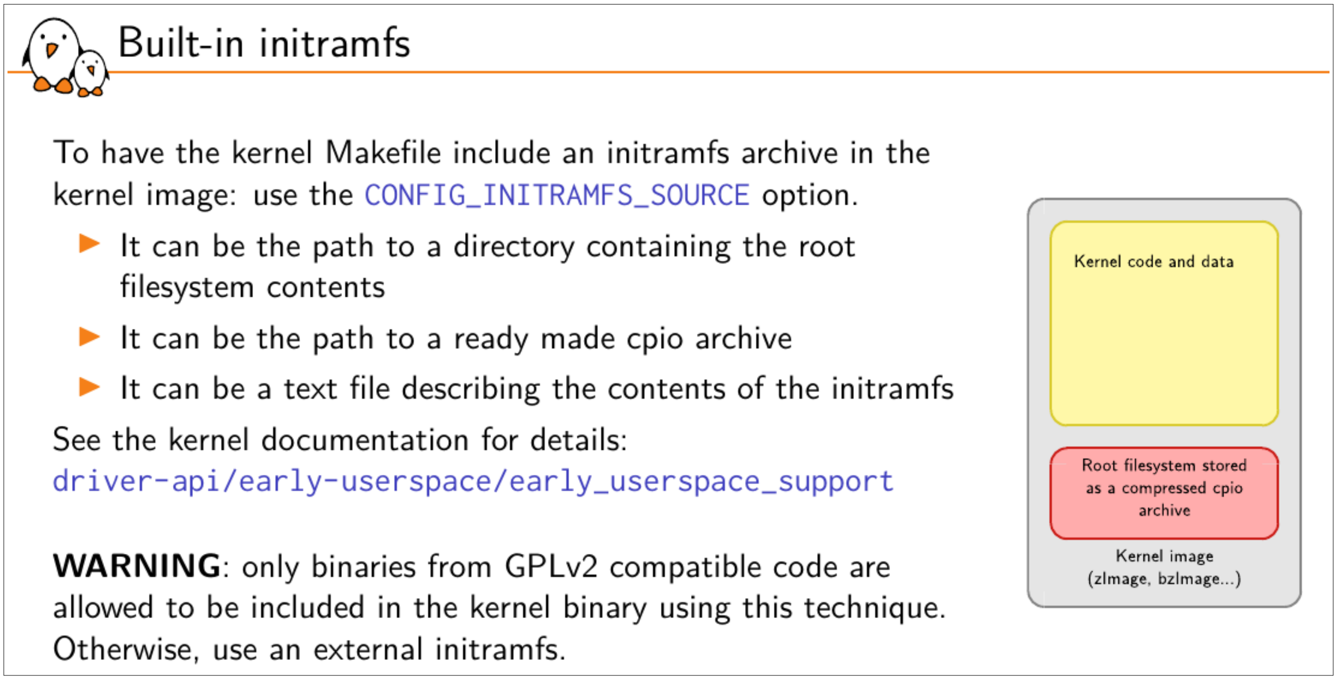
```
cd rootfs/
find . | cpio -H newc -o > ../initramfs.cpio
cd ..
gzip initramfs.cpio
```

- If you're using U-Boot, you'll need to include your archive in a U-Boot container:

```
mkimage -n 'Ramdisk Image' -A arm -O linux -T ramdisk -C gzip \
        -d initramfs.cpio.gz uInitramfs
```

- Then, in the bootloader, load the kernel binary, DTB and `uInitramfs` in RAM and boot the kernel as follows:

```
bootz kernel-addr initramfs-addr dtb-addr
```



Built-in initramfs

To have the kernel Makefile include an initramfs archive in the kernel image: use the `CONFIG_INITRAMFS_SOURCE` option.

- It can be the path to a directory containing the root filesystem contents
- It can be the path to a ready made cpio archive
- It can be a text file describing the contents of the initramfs

See the kernel documentation for details:
`driver-api/early-userspace/early_userspace_support`

WARNING: only binaries from GPLv2 compatible code are allowed to be included in the kernel binary using this technique. Otherwise, use an external initramfs.

Kernel code and data
Root filesystem stored as a compressed cpio archive
Kernel image (zImage, bzImage...)

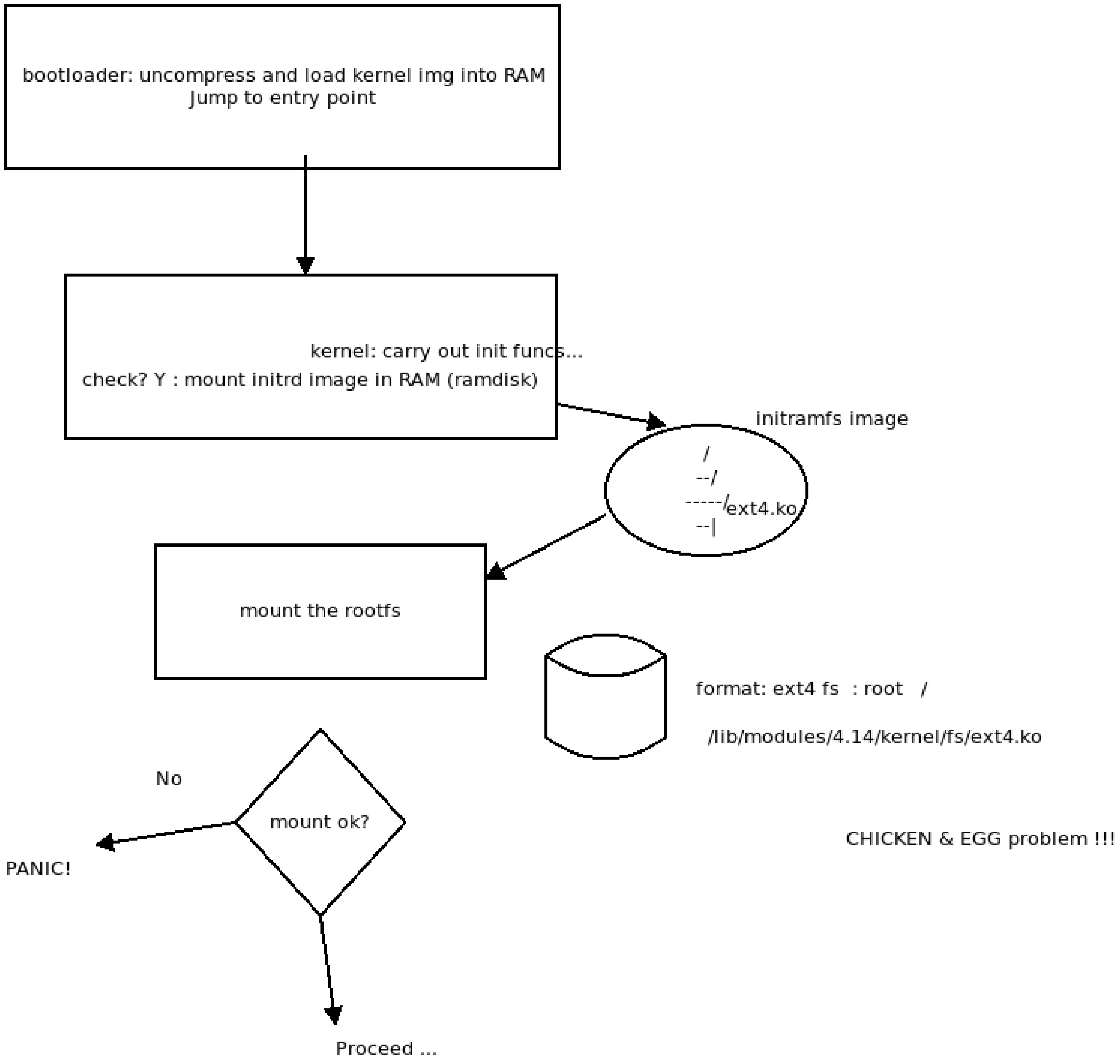
>>

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.

<< P.T.O. >>


bootloader: uncompress and load kernel img into RAM
Jump to entry point
kernel: carry out init funcs...
check? Y : mount initrd image in RAM (ramdisk)
initramfs image
/
--/
-----/ext4.ko
--|
mount the rootfs
format: ext4 fs : root /
/lib/modules/4.14/kernel/fs/ext4.ko
No
mount ok?
PANIC!
CHICKEN & EGG problem !!!
Proceed ...

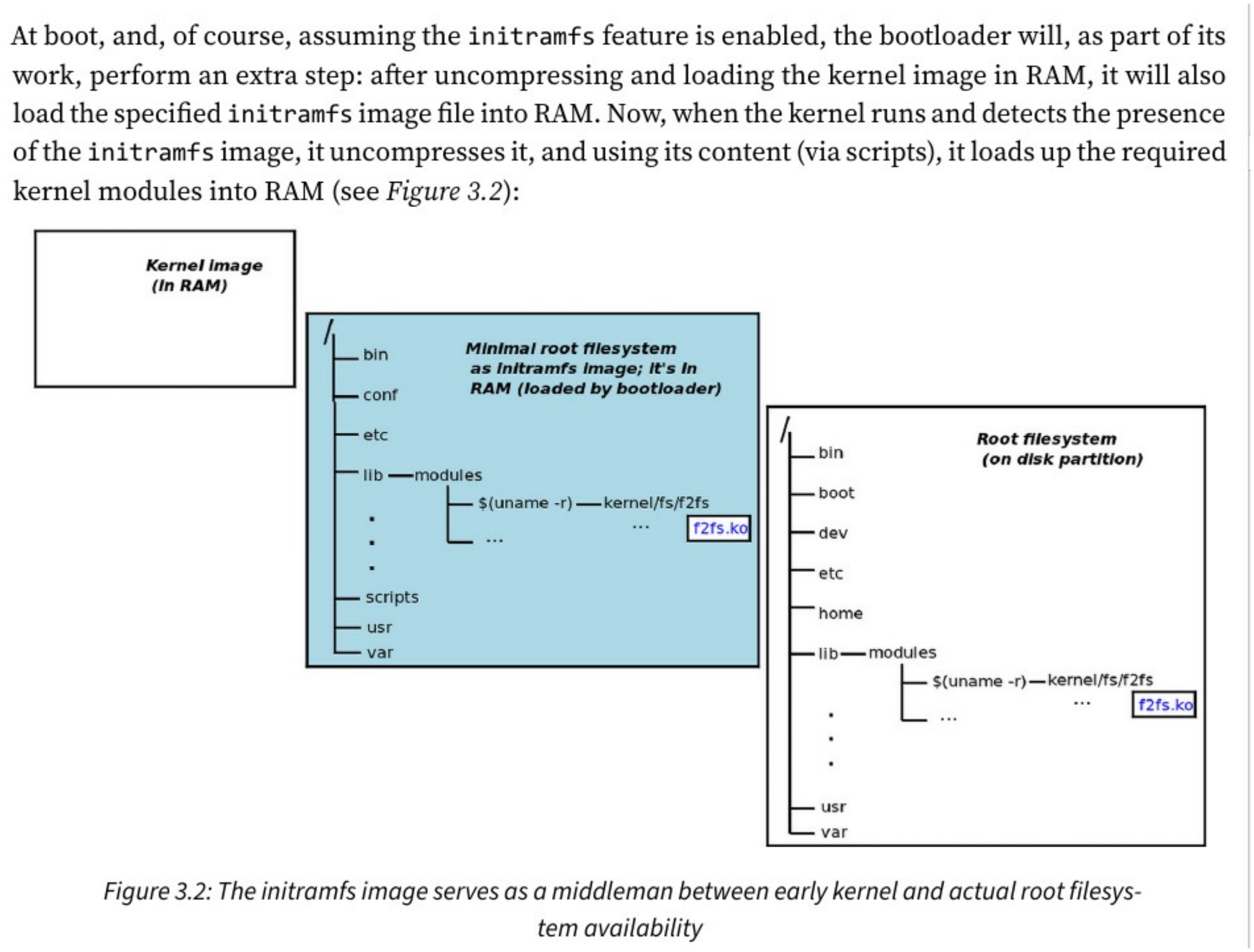
Initramfs is very useful for stuff like:

- running an app *before* the kernel fully comes up (for a time critical app too!)
  - get password for encrypted block device(s)
    - `man systemd-ask-password-console.service`
      `“systemd-ask-password-console.service is a system service that queries the user for system passwords (such as hard disk encryption keys and SSL certificate passphrases) on the console. It is intended to be used during boot to ensure proper handling of passwords necessary for boot. …”`
      more details here

  - 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

From the LKP 2E book:



At boot, and, of course, assuming the `initramfs` feature is enabled, the bootloader will, as part of its work, perform an extra step: after uncompressing and loading the kernel image in RAM, it will also load the specified `initramfs` image file into RAM. Now, when the kernel runs and detects the presence of the `initramfs` image, it uncompresses it, and using its content (via scripts), it loads up the required kernel modules into RAM (see *Figure 3.2*):

Figure 3.2: The initramfs image serves as a middleman between early kernel and actual root filesystem availability

…

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
```

```
==========================================================================
$
```

Script to extract and thus see initramfs content (on Ubuntu)

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

© 2000-2024 Kaiwan N Billimoria, kaiwanTECH

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: https://kaiwantech.com

Linux Operating System Specialized

The highest quality Training on:

Linux Fundamentals, CLI and Scripting
Linux Systems Programming
Linux Kernel Internals
Linux Device Drivers
Embedded Linux
Linux Debugging Techniques
New! *Linux OS for Technical Managers*

Please do visit our website for details:
http://kaiwantech.in