Out-of-tree kernel modules: some thoughts

September 23, 2017

Out-of-tree modules

- Modules ported to a different kernel version
  - Backporting, forward portting of removed staging drivers
- Externally developed modules
  - Intel, Qlogic networking drivers; ZFS on Linux, OpenAFS, ...

Driver development

- There are several good books that cover driver development for Linux
- One of the latest and greatest is "Linux Device Drivers, 3rd Edition"

Driver development

- There are several good books that cover driver development for Linux
- One of the latest and greatest is "Linux Device Drivers, 3rd Edition"
- Unfortunately, it has been written against Linux 2.6.10, which came out in the year of 2004, 13 years ago

Driver development

- There are several good books that cover driver development for Linux
- One of the latest and greatest is "Linux Device Drivers, 3rd Edition"
- Unfortunately, it has been written against Linux 2.6.10, which came out in the year of 2004, 13 years ago
- And there are no plans for the fourth edition: `https://www.reddit.com/r/linux/comments/61q6y8/`

Driver development

- There are several good books that cover driver development for Linux
- One of the latest and greatest is "Linux Device Drivers, 3rd Edition"
- Unfortunately, it has been written against Linux 2.6.10, which came out in the year of 2004, 13 years ago
- And there are no plans for the fourth edition: `https://www.reddit.com/r/linux/comments/61q6y8/`
- So the only actual reference is the Linux source code

Porting issues

- Linux doesn't have stable APIs (and internal ABIs, for that matter)
- Luckily, most of the changes either introduce new APIs or are syntactically incompatible with the existing code

Support for multiple kernel versions

- There are `LINUX_KERNEL_VERSION` and `LINUX_VERSION_CODE` macros

Support for multiple kernel versions

- There are `LINUX_KERNEL_VERSION` and `LINUX_VERSION_CODE` macros
- Mostly useless

Support for multiple kernel versions

One can implement various configure checks...

```
# CR_CHECK_KERNEL_SYMBOL(SYMBOL,[INCLUDES],[TEMPLATE_TEXT])
# Wrapper to invoke CR_CHECK_KERNEL_COMPILE for a given symbol
AC_DEFUN([CR_CHECK_KERNEL_SYMBOL],[
  CR_CHECK_KERNEL_COMPILE([$1],[$2],[
     int x = sizeof(&$1);
  ],[$3]) ])
```

Support for multiple kernel versions

One can implement various configure checks...

```
# CR_CHECK_KERNEL_SYMBOL(SYMBOL,[INCLUDES],[TEMPLATE_TEXT])
# Wrapper to invoke CR_CHECK_KERNEL_COMPILE for a given symbol
AC_DEFUN([CR_CHECK_KERNEL_SYMBOL],[
  CR_CHECK_KERNEL_COMPILE([$1],[$2],[
     int x = sizeof(&$1);
  ],[$3]) ])

# CR_CHECK_KERNEL_CALL_NARGS(SYMBOL,INCLUDES,ARGS1[,ARGS2...])
# See if each given SYMBOL(ARGSn) will compile
# Defines HAVE_[N]_ARG_[uppercase(£1)] for N equal to argument
AC_DEFUN([CR_CHECK_KERNEL_CALL_NARGS],[
  m4_if(m4_eval([$# >= 3]),1,[
    pushdef([cr_nargs],[len(patsubst([$3],[[^,]+,?],[@]))])
    CR_CHECK_KERNEL_CALL_FULL(cr_nargs[-arg $1],[$1],[$2],[],[$3],
    [[Define to 1 if the kernel has ]cr_nargs[-arg $1().]])
    popdef([cr_nargs])])
  m4_if(m4_eval([$# > 3]),1,[$0([$1],[$2],
    m4_shift(m4_shift(m4_shift($@))))])dnl tail recursion
])
```

From BLCR's acinclude.m4.

Support for multiple kernel versions

...or perform various trickery.

```
/*
 * "[kernel] list: fix order of arguments for hlist_add_after(_rcu)"
 * (commit 1d023284c31a) introduced new hlist_add_behind function,
 * which fixes unnatural argument order used in hlist_add_after function.
 */
#ifndef hlist_add_behind
# define hlist_add_behind(a, b) hlist_add_after(b, a)
#endif
```

Support for multiple kernel versions

...or perform various trickery.

```
/*
 * "[kernel] list: fix order of arguments for hlist_add_after(_rcu)"
 * (commit 1d023284c31a) introduced new hlist_add_behind function,
 * which fixes unnatural argument order used in hlist_add_after function.
 */
#ifndef hlist_add_behind
# define hlist_add_behind(a, b) hlist_add_after(b, a)
#endif

#if LINUX_VERSION_CODE < KERNEL_VERSION(3,6,0)
/* Added in 786e2288 */
/* Working around SLES 11 where it is defined despite 3.0 kernel version */
#define pci_pcie_type(dev) my_pci_pcie_type(dev)
static inline int my_pci_pcie_type(struct pci_dev *dev)
{
        return (pcie_caps_reg(dev) & PCI_EXP_FLAGS_TYPE) >> 4;
}
#endif /* #if LINUX_VERSION_CODE < KERNEL_VERSION(3,6,0) */
```

Building

- Kbuild has some support for building out of tree modules:

```
make -C ${KERNEL_DIR} M=${BUILD_DIR} modules
```

Building

- Kbuild has some support for building out of tree modules:

```
make -C ${KERNEL_DIR} M=${BUILD_DIR} modules
```

- Not much more than that

Building

- Kbuild has some support for building out of tree modules:

```
make -C ${KERNEL_DIR} M=${BUILD_DIR} modules
```

- Not much more than that

Two useful pieces of information:

- Kbuild has rather poor support for out-of-tree building: it can't use generated Makefile (which is put in the build directory by `config.status`), for example
- GCC version used for module building must match GCC version used for kernel building; most distributions specify it explicitly in kernel make files, but SuSE doesn't

Example Makefile

```
M ?= .
SRC ?= $(filter-out %.mod.c,$(subst $(M)/,,$(wildcard $(M)/*.c)))
HDR ?= $(subst $(M),,$(wildcard $(M)/*.h))
OBJ := $(patsubst %.c,%.o,$(SRC))

-include config.mk

obj-m += $(MODULE_NAME).o
$(MODULE_NAME)-y := $(OBJ)

all: module

module: $(MODULE_NAME).ko

$(MODULE_NAME).ko: $(SRC) $(HDR)
	$(MAKE) -C $(KERNELDIR) M=$(MODULE_SRC_DIR) modules

.PHONY: all module
```

Packaging

- No established way to package out-of-tree modules (Fedora just ignores their existence in Packaging Guidelines[1], for example)
- Users can build their own kernels (with different internal APIs and ABIs), which makes distribution of pre-built kernel modules impossible.

[1] "Fedora strongly encourages kernel module packagers to submit their code into the upstream kernel tree", they say

dkms

- Provides an infrastructure for managing out-of-tree modules
- Support building of modules, distribution tarballs, RPMs, DEBs
- De-facto standard for maintaining out-of-tree kernel modules in most community distributions.

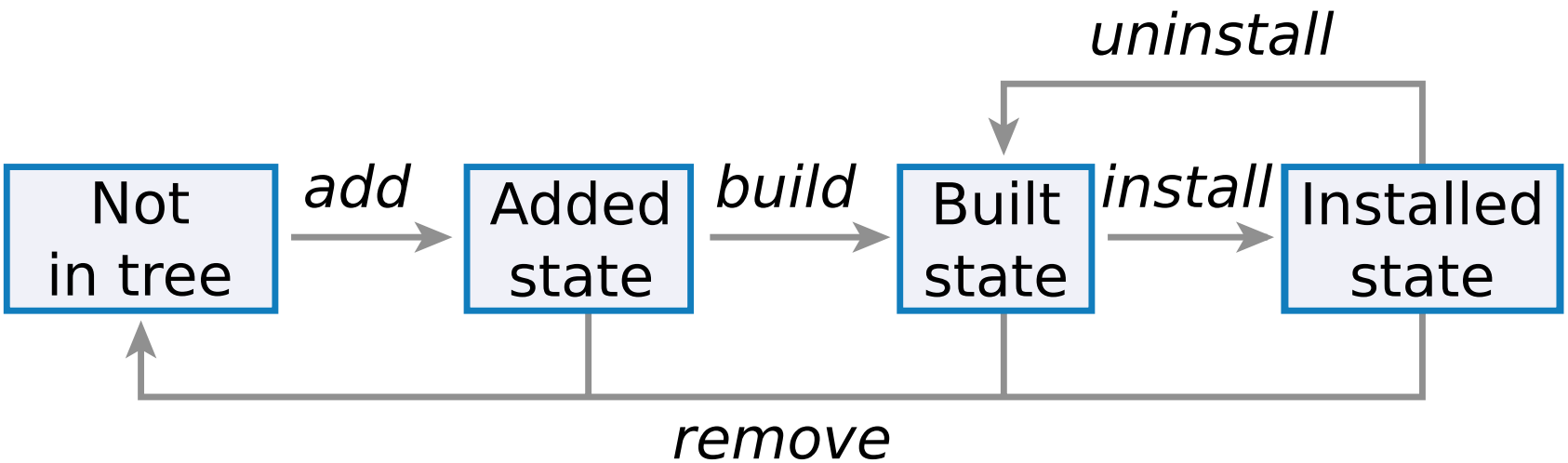

Figure: State machine for a driver managed by DKMS

dkms: issues

- Assumes (direct) control. Otherwise:
  - We have a makefile...

dkms: issues

- Assumes (direct) control. Otherwise:
  - We have a makefile...
  - ...that is embedded by Kbuild in order to get list of objects to build...

dkms: issues

- Assumes (direct) control. Otherwise:
  - We have a makefile...
  - ...that is embedded by Kbuild in order to get list of objects to build...
  - ...and has rules for calling dkms build...

dkms: issues

- Assumes (direct) control. Otherwise:
  - We have a makefile...
  - ...that is embedded by Kbuild in order to get list of objects to build...
  - ...and has rules for calling dkms build...
  - ...that uses config that calls make with this makefile.

dkms: issues

- Assumes (direct) control. Otherwise:
  - We have a makefile...
  - ...that is embedded by Kbuild in order to get list of objects to build...
  - ...and has rules for calling dkms build...
  - ...that uses config that calls make with this makefile.
- Implements full module management functionality that conflicts with the one distribution has.

akmods

- Set of scripts for rebuilding kmod source RPMs for various kernels, tailored specially for Fedora
- kmod RPM specs consist of lots of boilerplate code and are written aroung big fat %{`kernel_module_tool`} macro and `kmodtool` script
- Packaging is not automated at all, "just grab spec template and shape it to suit your needs"

ddiskit

- Kernel module backporting assitance tool
- Tailored for the Red Hat Driver Disk building workflow
- Initially, it was basically a set of make rules that automate building of set of kernel modules for a set of kernels

ddiskit

- Kernel module backporting assitance tool
- Tailored for the Red Hat Driver Disk building workflow
- Initially, it was basically a set of make rules that automate building of set of kernel modules for a set of kernels
- The second version was the same, but builds RPMs

ddiskit

- Kernel module backporting assitance tool
- Tailored for the Red Hat Driver Disk building workflow
- Initially, it was basically a set of make rules that automate building of set of kernel modules for a set of kernels
- The second version was the same, but builds RPMs
- The third incarnation is a Python rewrite that introduced more automation

ddiskit

- Kernel module backporting assitance tool
- Tailored for the Red Hat Driver Disk building workflow
- Initially, it was basically a set of make rules that automate building of set of kernel modules for a set of kernels
- The second version was the same, but builds RPMs
- The third incarnation is a Python rewrite that introduced more automation
- %{`kernel_module_tool`}-free!

ddiskit

- Import sources from kernel tree:

```
ddiskit prepare_sources \
    -d drivers/net/ethernet/ibm -r ${GIT_HASH}
```

- Generate spec:

```
ddiskit generate_spec
```

- Build RPM:

```
ddiskit build_rpm -G -a -e -m
```

- Generate Driver Disk:

```
ddiskit build_iso
```

Links

- LDD3: `https://lwn.net/Kernel/LDD3/`
- BLCR: `http://crd.lbl.gov/departments/computer-science/CLaSS/research/BLCR/`
- dkms: `https://github.com/dell/dkms`
- akmods: `https://rpmfusion.org/Packaging/KernelModules/Akmods`
- ddiskit: `https://github.com/orosp/ddiskit`