Kernel Rookie Guide

Workshop

Andrzej Pietrasiewicz
andrzej.p@samsung.com

Samsung R&D Institute Poland
Warsaw

Open Source Summit Europe
21st October 2018

SAMSUNG

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

About me

- A Husband and Lord Father of Two Sons
- Senior Software Engineer
- involved in kernel development since Oct'10
  - ARM/Exynos
  - ~300 commits (JPEG codec - co-maintainer, USB gadget, others)
  - >30k LOC
  - >50k LOP
  - #19 in top 20 contributors for kernel 4.0
    (by lines changed 3.19→4.0)
- Kernel Rookie Guide I..V 2012-2017 (internally @ Samsung)
  - Author
  - Instructor

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Requirements 1/3

- Some C knowledge
- Ability to use 'make'
- Own laptop with Linux available
- Network connectivity
- Internet browser

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Requirements 2/3

- In Linux:
  - qemu-system-arm version at least 2.5.0
  - Cross toolchain [1]
  - Tools for building the kernel
  - tar
  - wget
  - editor
  - wget `https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.18.tar.xz`[2]
  - workshops.tar.gz (from the instructor)
  - extract linux-4.18.tar.xz and workshops tar.gz next to each other (expect linux-4.18 and workshops directories next to each other)

[1] arm-linux-gnueabi-gcc expected in $PATH

[2] tar xxf <filename>

Requirements 3/3

- On a Ubuntu 18.04 Desktop fresh install[3]
  - `sudo apt-get install qemu-system-arm make gcc bison flex pkg-config libncurses-dev gcc-arm-linux-gnueabi vim-gtk`
- On older Ubuntus you might also need
  - `sudo apt-get install bc libssl-dev`
- Translate these requirements to the distro you intend to use at workshops!

[3] I tried it in a virtual machine

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Rationale

- Ancient/bare-metal programming vs. modern programming
- More tasks than CPUs at a time (some vs. 1 or more)
- Scarce resources (cannot satisfy everyone at the same time)
- Devices

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

OS Purpose

- Manage processes (give everyone a chance to execute)
- Manage memory (code and data require memory)
- Manage devices (access peripherals)

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

OS major design decisions

- Microkernel vs monolithic (Tannenbaum vs Torvalds)
- How to use hardware capabilities
- How to interoperate with users
- Development model & project governance

Linux

- Monolithic, but with loadable modules
- Development discussed on mailing lists, patches
- Source code in git[4,5]
- ~800MB w/o version control
- Indexing, also on-line, e.g.: `https://elixir.bootlin.com`

[4] `git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git`
[5] `https://git-scm.com/`

Workshop 0[6]

- Find *container_of* definition in 4.18 and try understanding it
- If more than one found choose the one whose location looks most general
- `https://elixir.bootlin.com`, use 4.18

[6] Look for TODO.txt in W0 from the extracted tar.gz

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop 0 solution

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Linux

- There is no separate 'kernel process'[7]
- User process runs in CPU unprivileged mode
  - Until it needs operations which by OS design must happen in privileged mode
  - Syscalls (system calls) are gateways between these modes
  - A process, once in privileged mode, IS the 'running kernel'
  - Privileged code can actually reject servicing the syscall
- Multitasking
  - `schedule()`
  - Process can voluntarily yield the CPU while in kernel mode
  - Timer interrupt
- Self contained

[7] Kernel threads exist, though

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Hardware flavours

- Originally i386
- Many added, incl. arm and arm64
- Hardware discovery (ACPI, dtb[8], autodetection)
- Workshops
  - Emulated hardware - qemu[9]
  - ARM Versatile Express (qemu: vexpress-a9)

[8] https://www.devicetree.org/
[9] https://www.qemu.org/

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Kernel configuration

- Highly configurable
  - Architectures
  - Kernel subsystems (/kernel, /mm, /net, /drivers/*, /arch/* etc.)
  - Peripherals
  - Compile-in/Modularize/Leave out
- Dedicated tools complementing the Makefile

Kconfig	language for specifying kernel composition and dependencies
defconfig	sane set of choices for given architecture
.config	result of defconfig application

```
1 # sane set of settings
2 make ARCH=arm vexpress_defconfig
3
4 # customization if needed
5 make ARCH=arm menuconfig
```

Listing 1: Configuring the kernel

Kernel building

- (Cross-)toolchain[10] (ARCH=..., CROSS_COMPILE=...)
- Keep your CPU cores busy with -j!

```
# either
make -j8 ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-

# or
export CROSS_COMPILE=arm-linux-gnueabi-
make -j8 ARCH=arm
```

Listing 2: Compiling the kernel

- Make targets

```
make ARCH=arm help
```

Listing 3: Getting help on make targets

[10] arm-linux-gnueabi-gcc expected in $PATH

Kernel Rookie Guide Workshop

Andrzej Pietrasiewicz

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace

Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations

Userspace interface
Syscalls
Filesystems
Devices

Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop

What now?

Workshop 1[11]

- Patch the kernel with the provided patches
- Configure & cross-compile the kernel for vexpress
- Analyze the vexpress-run.sh script
- Run emulated machine in qemu with the script
  - ^A-X to exit qemu

[11] Look for TODO.txt in W1

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop 1 solution

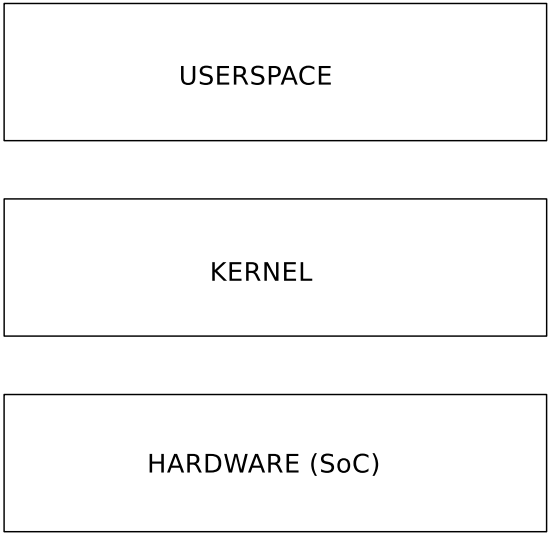
Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

System layers



Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Ramfs, tmpfs, rootfs, initramfs

ramfs	disk caches mounted as filesystem, no backing storage
tmpfs	ramfs += size limit, swapping
rootfs	special instance of ramfs/tmpfs, always present
initramfs	optional cpio archive[12], extracted into rootfs at boot time

- Initramfs handy for experimets with kernel
- We use external initramfs for workshops

[12] built into the kernel and/or provided by bootloader

Minimal userspace for workshops

- busybox[13] + minimal /etc packaged into initramfs
- Single binary containing 'applets' corresponding to standard tools
- Tool from kernel source tree understands both directories and list files

```
cat initramfs.list
file /busybox-armv7r ../initramfs/busybox-armv7r 0755 0 0
dir /sbin 0755 0 0
dir /proc 0755 0 0
dir /sys 0755 0 0
dir /bin 0755 0 0
dir /usr 0755 0 0
dir /usr/sbin 0755 0 0
dir /usr/bin 0755 0 0
dir /etc 0755 0 0
dir /etc/init.d 0755 0 0
file /etc/inittab ../initramfs/inittab 0755 0 0
file /etc/init.d/rcS ../initramfs/rcS 0755 0 0
```

Listing 4: Initramfs list - host paths relative to kernel source topdir!

[13] https://busybox.net/

Kernel Rookie Guide Workshop

Andrzej Pietrasiewicz

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace

Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations

Userspace interface
Syscalls
Filesystems
Devices

Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop

What now?

Workshop 2[14]

- Analyze generate_initramfs.sh
- Add a short text file to initramfs image using generate_initramfs.sh
- Run the virtual machine and verify new file's presence and contents

[14] TODO.txt in W2

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop 2 solution

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Source code placement

in-tree	inside kernel source directory
out-of-tree	outside kernel source directory

- Kernel build system happily handles both!

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Out-of-tree Makefile

- LDD3 [15] provides a nice Makefile (meant for compiling modules for currently running kernel)
- The below code creates hello.ko module (from hello.o and implicitly from hello.c)

```
# If KERNELRELEASE is defined, we've been invoked from the
# kernel build system and can use its language.
ifneq ($(KERNELRELEASE),)
    obj-m := hello.o
# Otherwise we were called directly from the command
# line; invoke the kernel build system.
else
    KERNELDIR ?= /lib/modules/$(shell uname -r)/build
    PWD  := $(shell pwd)
default:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif
```

Listing 5: Out-of-tree module makefile

[15] https://lwn.net/Kernel/LDD3/, ISBN: 0-596-00590-3

Interaction with modules

- Module dependencies
- Userspace tools
  - **insmod** dependencies must be provided
  - **modprobe** dependencies resolved
  - **rmmod** unload only this module
  - **modprobe -r** unload this module and unused dependencies
  - **lsmod** list loaded modules
  - **depmod** analyze module dependencies

```
1  # loading
2  insmod <filename>.ko
3  # or
4  modprobe <module name>
5
6  #unloading
7  rmmod <module name>
8  # or
9  modprobe -r <module name>
10
11 # list loaded
12 lsmod
```

Listing 6: Loading/unloading kernel modules

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Module boilerplate code

```c
#include <linux/init.h>
#include <linux/module.h>

static int hello_init(void)
{
    return 0;
}
module_init(hello_init);

static void hello_exit(void)
{
}
module_exit(hello_exit);

MODULE_LICENSE("GPL");
```

Listing 7: Linux kernel module boilerplate code

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Module parameters

- A value passed to module at insertion[16,17]
- Several types (int, string, etc)
- A way of associating the said value with a variable in the module

```
1 /* use this at file scope! */
2 module_param(<var_name>, int, <flags>) /* flags for sysfs access mode */
```

Listing 8: Module parameters (selection)

[16] or in the kernel command line if module compiled-in
[17] or through sysfs if parameter available there

Kernel diagnostic messages circular buffer

- Always available
- Never runs out of space[18]
- Some messages can be simultaneously output to the current system console[19]
- `dmesg` to output the diagnostic messages circular buffer contents

[18] But older messages can be overwritten

[19] Depends on message levels and current kernel settings

Message levels

- message levels KERN_*

EMERG (0)	system is unusable
ALERT (1)	action must be taken immediately
CRIT (2)	critical conditions
ERR (3)	error conditions
WARNING (4)	warning conditions
NOTICE (5)	normal but significant condition
INFO (6)	informational
DEBUG (7)	debug-level messages

```
1 cat /proc/sys/kernel/printk
2 4 4 1 7
```

Listing 9: Printk levels

Writing messages to the console

- Output methods

  printk printf equivalent

  pr_<level> printf equivalent (level: emerg, alert, crit etc.)[20]

- Remember about newlines!

```
1 /* default level */
2 printk("Hello, world!\n");
3
4 /* selected level */
5 printk(KERN_WARNING "Hello, warning!\n");
6
7 pr_warning("Hello, warning!\n");
```

Listing 10: Outputting messages from kernel

[20] for pr_debug see Documentation/admin-guide/dynamic-debug-howto.rst

Kernel Rookie Guide Workshop
Andrzej Pietrasiewicz
Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop 3[21]

- Write, compile and deploy (initramfs!) a module which does (almost) nothing
  - Module's init shall output a welcome string
  - module's exit shall output a goodbye string
- Load/unload the module in the virtual machine

[21] TODO.txt in W3

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop 3 solution

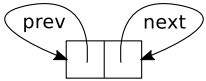
Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Linked lists in kernel

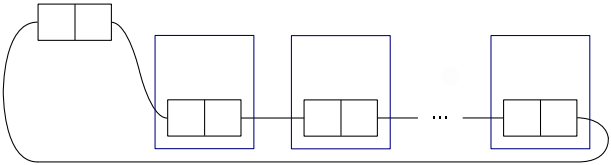


- Doubly-linked
- Circular

```
1 struct list_head {
2   struct list_head *next, *prev;
3 };
```

Listing 11: Linked list structure

- Embedded struct list_head
- *container_of*



Kernel Rookie Guide Workshop
Andrzej Pietrasiewicz
Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Linked lists API (selection)

```
/* declare list handle */
LIST_HEAD(my_list)

/* add/delete */
list_add(struct list_head *new, struct list_head *head)
list_add_tail(struct list_head *new, struct list_head *head)
list_del(struct list_head *entry)

/* query/get elements */
int list_empty(const struct list_head *head)
list_entry(ptr, type, member)
list_first_entry(head, type, member)
list_last_entry(head, type, member)

/* iterate */
list_for_each(pos, head)
list_for_each_safe(pos, next_tmp, head)
list_for_each_entry(pos, head, member)
list_for_each_entry_safe(pos, next_tmp, head, member)

/* see include/linux/list.h for more */
```

Listing 12: Linked lists API (selection)

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Small allocations

kzalloc allocate (several kB max)

kfree free

- *sizeof* is your friend
- verify that the return value is != NULL
- For greater amounts use other methods![22]

```
1 void *kzalloc(size_t size, gfp_t flags) /* flags almost always GFP_KERNEL */
2 kfree(const void *)
```

Listing 13: Small allocations API

[22] e.g. page allocator, slab caches, vmalloc

Kernel Rookie Guide Workshop
Andrzej Pietrasiewicz
Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop 4[23]

- Analyze krg.c
- Fill the blanks as instructed inline
- Compile and deploy the module
- Test your module
  - Try inserting and removing it several times
  - Try using the 'number' module parameter

[23] TODO.txt in W4

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop 4 solution

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Syscalls (1/2)

- Userspace programs don't 'link' against kernel
- Syscalls have numbers (e.g. on ARM exit:1, fork:2, read:3, write:4 etc)
- ABI - defines interaction between binary programs, including convention of passing parameters, receiving results and triggering kernel services[24]
  - CPU registers (e.g. on ARM r0, r1, ...)
  - stack

[24] Each ABI requires its matching toolchain

Syscalls (2/2)

- ARM Embedded ABI (here: 32-bit)[25]
  - syscall number in r7
  - syscall arguments in r0...r6
  - return value in r0
  - software interrupt #0 triggers syscall execution

```
1 # call syscall 'exit' with parameter 0
2 mov r7, #1
3 mov r0, #0
4 svc #0
```

Listing 14: Calling a syscall

- libc is your friend!
- strace is your friend, too!

[25] man syscall, other ARM ABIs exist, e.g. armhf

Workshop 5[27]

- Analyze, compile[26] and deploy "Hello, world" written in ARM assembly
- Run it in the virtual machine

[26] use the provided compile_hello.sh
[27] TODO.txt in W5

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop 5 solution

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Adding new syscalls

- **Don't**
- Unless you really have to[28]
- Or in a training

[28] Difficult if you want it upstream

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Architectural overview

- User
- Syscall interface
- VFS (Virtual Filesystem/Virtual Filesystem Switch)
- 'Real' filesystems (ext2, ext3, ext4....), pseudo filesystems[29]
- Block layer
- Device drivers

[29] all available ('real' and pseudo) listed in `/proc/filesystems`

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Pseudo filesystems

- Pseudo filesystems
  - sysfs
  - proc
  - debugfs
  - configfs
  - ...
- No Block layer/Device drivers
- Particular filesystem driver knows what it means to read/write a 'file'
- 'Everything is a file' philosophy

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Devices and drivers

- One of OS purposes: Manage devices
- Character devices, block devices
- entries in `/dev`

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

ioctl

- Swiss army knife of all syscalls[30]
- Called on device special file file descriptor
- 'Sub-syscall' number encoded in parameter (interpreted by driver)
- Optional pointer to user memory

```
1 int ioctl(int fd, unsigned long request, ...);
```

Listing 15: libc's ioctl() prototype

[30] man ioctl

Kernel Rookie Guide Workshop
Andrzej Pietrasiewicz
Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Memory access considerations

- process virtual memory
- kernel virtual memory
- accessing user memory from kernel might be non-trivial![31]
- `copy_from_user()`, `copy_to_user()`

[31] https://youtu.be/EWwfMM2AW9g

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

UP vs SMP?

- SMP: multiple threads of execution at the same time
- UP: single thread of execution at a time

BUT

- Even on UP effectively concurrent!

SO

- Synchronization/locking required in either case

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Common synchronization primitives 1/3

- Semaphore[32]
  - Can be incremented and decremented at will
  - Any thread of execution can do it
  - If current value 0, next to decrement will be put to sleep
  - Sleeping until someone increments (first come - first served)
  - Potentially long waiting time

```
1  /* definition */
2
3  /* initial count 1 */
4  DEFINE_SEMAPHORE(name)
5
6  /* any initial count */
7  struct semaphore my_semaphore = __SEMAPHORE_INITIALIZER(my_semaphore, initial_count)
8  void sema_init(struct semaphore *sem, int val)
9
10 /* increment/decrement */
11 void up(struct semaphore *sem)
12 int down_interruptible(struct semaphore *sem)
```

Listing 16: Semaphore API (selection)

[32] `#include <linux/semaphore.h>`

Kernel Rookie Guide Workshop
Andrzej Pietrasiewicz
Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Common synchronization primitives 2/3

- Mutex[33]
  - Only two states: available and busy ('a binary semaphore')
  - Can only be released by the one who acquired
  - If busy, next to acquire will be put to sleep
  - Sleeping until someone releases (first come - first served)
  - Avoid long waiting time

```
1  /* definition */
2  DEFINE_MUTEX(mutexname)
3  mutex_init(mutex)
4
5  /* acquire/release */
6  void mutex_lock(struct mutex *lock)
7  void mutex_unlock(struct mutex *lock)
8
9  /* acquire or return, without waiting */
10 int mutex_trylock(struct mutex *lock)
```

Listing 17: Mutex API (selection)

[33] `#include <linux/mutex.h>`

Kernel Rookie Guide Workshop
Andrzej Pietrasiewicz
Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Common synchronization primitives 3/3

- Spinlock[34]
  - Busy waiting (can be good!)
  - While contended, no guarantee of 'fair' acquiring across all architectures
  - Sometimes turning off interrupts on local CPU required
  - Must be used if sleeping prohibited in a context
  - Supposed to be kept for very short time

```
1  /* initialize */
2  DEFINE_SPINLOCK(lock)
3
4  /* acquire/release */
5  void spin_lock(spinlock_t *lock)
6  void spin_unlock(spinlock_t *lock)
7
8  spin_lock_irqsave(lock, flags)
9  void spin_unlock_irqrestore(spinlock_t *lock, unsigned long flags)
10
11 /* acquire or return, without waiting */
12 int spin_trylock(spinlock_t *lock)
```

Listing 18: Spinlock API (selection)

[34] `#include <linux/spinlock.h>`

Kernel Rookie Guide Workshop

Andrzej Pietrasiewicz

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace

Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations

Userspace interface
Syscalls
Filesystems
Devices

Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop

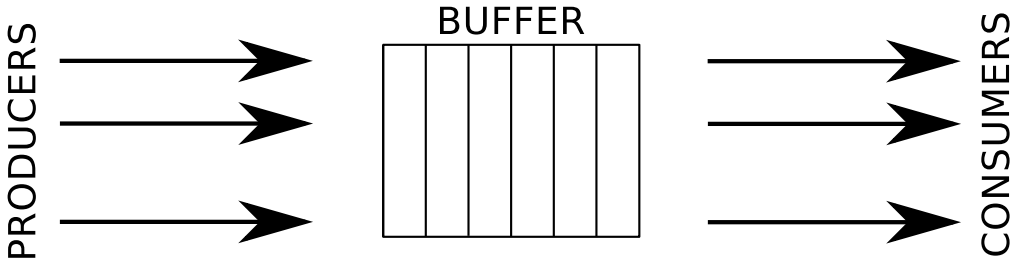
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Producer and consumer (aka Bounded buffer) problem




```
1 down(empty_count) /* at init: buffer size */
2   lock(buffer)
3     put_into(buffer)
4   unlock(buffer)
5 up(fill_count)
```

Listing 19: Producer pseudo code

```
1 down(fill_count) /* at init: 0 */
2   lock(buffer)
3     get_from(buffer)
4   unlock(buffer)
5 up(empty_count)
```

Listing 20: Consumer pseudo code

Workshop 6[35]

- Analyze the userspace program (syscall.c) and discover what it does
- Analyze `arch/arm/kernel/sys_arm.c` in kernel sources and fill the blanks as instructed inline
- Rebuild the kernel
- Try running ./syscall in the virtual machine
  - ./syscall -p <number>to put element into buffer
  - ./syscall -g to get element from buffer
  - You might need to put either into background with & (e.g. while getting from empty buffer)

[35] TODO.txt in W6

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop 6 solution

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Workshop plan

- Meeting the Linux kernel
  - OS purpose and Linux
  - Hardware, Kernel & toolchain
  - (Early) userspace
- Writing basic modules
  - In-tree vs out-of-tree
  - Module boilerplate code
  - Linked lists
  - Small allocations
- Userspace interface
  - Syscalls
  - Filesystems
  - Devices
- Synchronization and locking basics
  - Concurrency
  - Synchronization primitives
  - Final workshop
- What now?

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Further reading/coding

- symbol exporting (EXPORT_SYMBOL*)
- more allocators and memory management, including DMA
- more locking (atomics, rw locks, RCU, memory barriers...)
- character devices
- driver model
- frameworks, subsystems
- Device Tree
- git & patches preparation
- mailing lists

Thank you for attention!
andrzej.p@samsung.com

SAMSUNG

Meeting the Linux kernel
OS purpose and Linux
Hardware, Kernel & toolchain
(Early) userspace
Writing basic modules
In-tree vs out-of-tree
Module boilerplate code
Linked lists
Small allocations
Userspace interface
Syscalls
Filesystems
Devices
Synchronization and locking basics
Concurrency
Synchronization primitives
Final workshop
What now?

Appendix

```
#!/bin/bash

LINUX_SOURCE=../linux-4.18
INITRAMFS_PATH=${PWD}/initramfs
MODULES_PATH=${PWD}/..
INITRAMFS_LIST_PATH=${PWD}/initramfs

cd ${LINUX_SOURCE}
scripts/gen_initramfs_list.sh -o ${INITRAMFS_PATH}/initramfs.cpio \
  ${INITRAMFS_LIST_PATH}/initramfs.list
```

Listing 21: generate_initramfs.sh

```
#!/bin/bash

LINUX_SOURCE=../linux-4.18
INITRAMFS_PATH=initramfs
QEMU_PREFIX=

${QEMU_PREFIX}qemu-system-arm \
  -nographic \
  -M vexpress-a9 \
  -kernel ${LINUX_SOURCE}/arch/arm/boot/zImage \
  -append "root=/dev/ram␣rdinit=/busybox-armv7r␣init␣console=ttyAMA0,115200n8␣earlyprintk" \
  -dtb ${LINUX_SOURCE}/arch/arm/boot/dts/vexpress-v2p-ca9.dtb \
  -initrd ${INITRAMFS_PATH}/initramfs.cpio
```

Listing 22: vexpress_run.sh

```
file /busybox-armv7r ../workshops/initramfs/busybox-armv7r 0755 0 0
dir /sbin 0755 0 0
dir /proc 0755 0 0
dir /sys 0755 0 0
dir /bin 0755 0 0
dir /usr 0755 0 0
dir /usr/sbin 0755 0 0
dir /usr/bin 0755 0 0
dir /etc 0755 0 0
dir /etc/init.d 0755 0 0
file /etc/inittab ../workshops/initramfs/inittab 0755 0 0
file /etc/init.d/rcS ../workshops/initramfs/rcS 0755 0 0
file /syscall ../workshops/initramfs/syscall 0755 0 0
file /strace ../workshops/initramfs/strace 0755 0 0
```

Listing 23: initramfs.list

```
::sysinit:/etc/init.d/rcS
::askfirst:-/bin/sh
::ctrlaltdel:/sbin/reboot
::shutdown:/sbin/swapoff -a
::shutdown:/sbin/umount -a -r
::restart:/sbin/init
```

Listing 24: inittab

```
#!/busybox-armv7r sh

/busybox-armv7r --install

mount -t proc none /proc
mount -t sysfs none /sys

mdev -s
echo /sbin/mdev > /proc/sys/kernel/hotplug
```

Listing 25: rcS

```
From 3aae8f646785afab58f2195111d6280029af7ac0 Mon Sep 17 00:00:00 2001
From: Andrzej Pietrasiewicz <andrzej.p@samsung.com>
Date: Fri, 17 Aug 2018 13:58:36 +0200
Subject: [PATCH] kbuild: make sorting initramfs contents independent of locale

Some LANG values (e.g. pl_PL.UTF-8) cause the sort command to output
files before their parent directories, which makes them inaccessible for
the kernel. In other words, when the kernel populates the rootfs, it is
unable to create files whose parent directories have not been yet created.

This patch makes sorting use the default (LANG=C) locale, which results in
correctly laid out initramfs images (parent directories before files).

Signed-off-by: Andrzej Pietrasiewicz <andrzej.p@samsung.com>
---
 scripts/gen_initramfs_list.sh | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

diff --git a/scripts/gen_initramfs_list.sh b/scripts/gen_initramfs_list.sh
index 10e528b..0aad760 100755
--- a/scripts/gen_initramfs_list.sh
+++ b/scripts/gen_initramfs_list.sh
@@ -174,7 +174,7 @@ dir_filelist() {
   ${dep_list}header "$1"

   srcdir=$(echo "$1" | sed -e 's://*:/:g')
-  dirlist=$(find "${srcdir}" -printf "%p %m %U %G\n" | sort)
+  dirlist=$(find "${srcdir}" -printf "%p %m %U %G\n" | LANG=C sort)

   # If $dirlist is only one line, then the directory is empty
   if [  "$(echo "${dirlist}" | wc -l)" -gt 1 ]; then
--
2.7.4
```

Listing 26: kbuild patch

```
From 21cf89b5a128b6a3390efd6c160f67edfde20fa9 Mon Sep 17 00:00:00 2001
From: Bartlomiej Zolnierkiewicz <b.zolnierkie@samsung.com>
Date: Fri, 31 Aug 2018 11:53:36 +0200
Subject: [PATCH] Revert "irqdomain: Don't set type when mapping an IRQ"

This reverts commit 1e2a7d78499ec8859d2b469051b7b80bad3b08aa.

This makes ARM QEMU vexpress platform work again.

Signed-off-by: Bartlomiej Zolnierkiewicz <b.zolnierkie@samsung.com>
---
 include/linux/irqdomain.h |  3 ---
 kernel/irq/irqdomain.c    | 23 +++++------------------
 2 files changed, 5 insertions(+), 21 deletions(-)

diff --git a/include/linux/irqdomain.h b/include/linux/irqdomain.h
index dccfa65..e93742d 100644
--- a/include/linux/irqdomain.h
+++ b/include/linux/irqdomain.h
@@ -526,9 +526,6 @@ static inline int irq_domain_alloc_irqs(struct irq_domain *domain,
  return -1;
 }

-static inline void irq_domain_free_irqs(unsigned int virq,
-         unsigned int nr_irqs) { }
-
 static inline bool irq_domain_is_hierarchy(struct irq_domain *domain)
 {
  return false;
```

Listing 27: irqdomain patch 1/3

```
diff --git a/kernel/irq/irqdomain.c b/kernel/irq/irqdomain.c
index 5d9fc01b..75f175b 100644
--- a/kernel/irq/irqdomain.c
+++ b/kernel/irq/irqdomain.c
@@ -744,7 +744,6 @@ static void of_phandle_args_to_fwspec(struct of_phandle_args *irq_data,
 unsigned int irq_create_fwspec_mapping(struct irq_fwspec *fwspec)
 {
   struct irq_domain *domain;
-  struct irq_data *irq_data;
   irq_hw_number_t hwirq;
   unsigned int type = IRQ_TYPE_NONE;
   int virq;
@@ -792,11 +791,7 @@ unsigned int irq_create_fwspec_mapping(struct irq_fwspec *fwspec)
      * it now and return the interrupt number.
      */
     if (irq_get_trigger_type(virq) == IRQ_TYPE_NONE) {
-      irq_data = irq_get_irq_data(virq);
-      if (!irq_data)
-        return 0;
-
-      irqd_set_trigger_type(irq_data, type);
+      irq_set_irq_type(virq, type);
       return virq;
     }

@@ -816,18 +811,10 @@ unsigned int irq_create_fwspec_mapping(struct irq_fwspec *fwspec)
       return virq;
   }

-  irq_data = irq_get_irq_data(virq);
-  if (!irq_data) {
-    if (irq_domain_is_hierarchy(domain))
-      irq_domain_free_irqs(virq, 1);
```

Listing 28: irqdomain patch 2/3

```
-   else
-     irq_dispose_mapping(virq);
-   return 0;
- }
-
- /* Store trigger type */
- irqd_set_trigger_type(irq_data, type);
-
+ /* Set type if specified and different than the current one */
+ if (type != IRQ_TYPE_NONE &&
+     type != irq_get_trigger_type(virq))
+   irq_set_irq_type(virq, type);
  return virq;
 }
 EXPORT_SYMBOL_GPL(irq_create_fwspec_mapping);
--
1.9.1
```

Listing 29: irqdomain patch 3/3

```
From cb1d564a98486d331f9fb164e8d266d9aedd0cd8 Mon Sep 17 00:00:00 2001
From: Andrzej Pietrasiewicz <andrzej.p@samsung.com>
Date: Fri, 5 Oct 2018 10:08:49 +0200
Subject: [PATCH] krg locking workshop

Signed-off-by: Andrzej Pietrasiewicz <andrzej.p@samsung.com>
---
 arch/arm/kernel/sys_arm.c  | 116 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 arch/arm/tools/syscall.tbl |   2 +
 2 files changed, 118 insertions(+)

diff --git a/arch/arm/kernel/sys_arm.c b/arch/arm/kernel/sys_arm.c
index bdf7514..072cc11 100644
--- a/arch/arm/kernel/sys_arm.c
+++ b/arch/arm/kernel/sys_arm.c
@@ -27,6 +27,9 @@
 #include <linux/ipc.h>
 #include <linux/uaccess.h>
 #include <linux/slab.h>
+#include <linux/semaphore.h>
+#include <linux/mutex.h>
+#include <linux/list.h>

 /*
  * Since loff_t is a 64 bit type we avoid a lot of ABI hassle
@@ -37,3 +40,116 @@ asmlinkage long sys_arm_fadvise64_64(int fd, int advice,
 {
  return ksys_fadvise64_64(fd, offset, len, advice);
 }
+
+/****************** Kernel Rookie Guide OSSEU'18 Edinburgh ******************/
+#define KRG_BUF_SIZE     13
+
```

Listing 30: krg workshop patch 1/5

```
+struct krg_job {
+ struct list_head  entry;
+ int     id;
+};
+
+static struct semaphore krg_empty_count = __SEMAPHORE_INITIALIZER(krg_empty_count, KRG_BUF_SIZE);
+static struct semaphore krg_fill_count = __SEMAPHORE_INITIALIZER(krg_fill_count, 0);
+
+static LIST_HEAD(krg_jobs);
+static DEFINE_MUTEX(krg_mutex);
+
+SYSCALL_DEFINE1(krg_put, int, id)
+{
+ struct krg_job *job;
+
+ job = NULL;
+ /*
+  * TODO: allocate struct krg_job
+  * use kzalloc()
+  */
+ if (!job)
+   return PTR_ERR(job);
+
+ /* TODO: set job id */
+
+ /*
+  * TODO: decrement empty count, use
+  *
+  * if (down_interruptible()) {
+  *  TODO: avoid memory leak
+  *  use kfree()
+  *
+  *  // upper layers can handle retrying
```

Listing 31: krg workshop patch 2/5

```
+  *  return -ERESTARTSYS;
+  * }
+  */
+
+ /*
+  * TODO: ensure only one process accesses the buffer
+  * use mutex_lock()
+  */
+
+ printk("> put:%d\n", job->id);
+
+ /*
+  * TODO: add job to the list
+  * use list_add_tail()
+  */
+
+ /*
+  * TODO: let other processes access the buffer
+  * use mutex_unlock()
+  */
+
+ /*
+  * TODO: increment fill count
+  * use up()
+  */
+
+ return 0;
+}
+
+SYSCALL_DEFINE0(krg_get)
+{
+ struct krg_job *job;
+ int ret;
```

Listing 32: krg workshop patch 3/5

```
+
+ /*
+  * TODO: decrement fill count, use
+  * if (down_interruptible())
+  *  return -ERESTARTSYS;
+  */
+
+ /*
+  * TODO: ensure only one process accesses the buffer
+  * use mutex_lock()
+  */
+
+ /*
+  * TODO: let 'job' point to the first element of the buffer
+  * use job = list_first_entry()
+  */
+
+ printk("<␣get:%d\n", job->id);
+
+ ret = job->id;
+
+ /*
+  * TODO: remove the element from the buffer
+  * use list_del()
+  */
+
+ /*
+  * TODO: avoid memory leak
+  * use kfree()
+  */
+
+ /*
+  * TODO: let other processes access the buffer
```

Listing 33: krg workshop patch 4/5

```
+   * use mutex_unlock()
+   */
+
+ /*
+   * TODO: increment empty count
+   * use up()
+   */
+
+ return ret;
+}
diff --git a/arch/arm/tools/syscall.tbl b/arch/arm/tools/syscall.tbl
index fbc74b5..febf960 100644
--- a/arch/arm/tools/syscall.tbl
+++ b/arch/arm/tools/syscall.tbl
@@ -413,3 +413,5 @@
 396   common   pkey_free    sys_pkey_free
 397   common   statx        sys_statx
 398   common   rseq         sys_rseq
+399   eabi   krg_put      sys_krg_put
+400   eabi   krg_get      sys_krg_get
--
2.7.4
```

Listing 34: krg workshop patch 5/5

```
1. Find container of definition in 4.18 and try understanding it

2. If more than one found choose the one whose location looks most general

3. https://elixir.bootlin.com, use 4.18
```

Listing 35: W0/TODO.txt

```
1. Patch the kernel with the provided patches

In kernel source directory:

patch -p1 < PATH_TO_WORKSHOPS/kernel-patches/0001-kbuild....
patch -p1 < PATH_TO_WORKSHOPS/kernel-patches/0001-Revert....
patch -p1 < PATH_TO_WORKSHOPS/kernel-patches/0001-krg....

2. Configure & cross-compile the kernel for vexpress

In kernel source directory:

make ARCH=arm vexpress_defconfig
make -j8 ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-

3. Analyze the vexpress-run.sh script

4. Run emulated machine in qemu with the script
  ^A-X to exit qemu
```

Listing 36: W1/TODO.txt

```
1. Add a short text file to initramfs image

Use initramfs.list - pay special attention to host paths which are relative to
kernel source toplevel directory!

2. Use generate initramfs.sh to build initramfs

3. Run the virtual machine and verify new file's presence and contents
```

Listing 37: W2/TODO.txt

```
1. Write, compile and deploy (initramfs!) a module which does (almost) nothing

   Module's init shall output a welcome string
   Module's exit shall output a goodbye string

Use the W3 directory and a Makefile there.

Remember about ARCH=arm and CROSS_COMPILE=arm-linux-gnueabi-

2. Load/unload the module in the virtual machine
```

Listing 38: W3/TODO.txt

```
ifneq ($(KERNELRELEASE),)

obj-m := hello.o

else

# modify appropriately
LINUX_SOURCE := ../../linux-4.18

PWD := $(shell pwd)

default:
	$(MAKE) -C $(LINUX_SOURCE) SUBDIRS=$(PWD) modules

clean:
	$(MAKE) -C $(LINUX_SOURCE) SUBDIRS=$(PWD) clean

endif
```

Listing 39: W3/Makefile

```
1. Analyze krg.c

2. Fill the blanks as instructed inline

3. Compile and deploy the module

Remember about ARCH=arm and CROSS_COMPILE=arm-linux-gnueabi-

4. Test your module
   Try inserting and removing it several times
   Try using the ‘number‘ module parameter
```

Listing 40: W4/TODO.txt

```
ifneq (${KERNELRELEASE},)

obj-m := krg.o

else

# modify appropriately
LINUX_SOURCE := ../../linux-4.18

PWD := $(shell pwd)

default:
  ${MAKE} -C ${LINUX_SOURCE} SUBDIRS=${PWD} modules

clean:
  ${MAKE} -C ${LINUX_SOURCE} SUBDIRS=${PWD} clean

endif
```

Listing 41: W4/Makefile

```c
/****************** Kernel Rookie Guide OSSEU'18 Edinburgh ******************
 *
 * (C) Andrzej Pietrasiewicz <andrzej.p@samsung.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 *
 */
#include <linux/module.h>
#include <linux/list.h>
#include <linux/slab.h>

struct data {
  int     id;
  struct list_head  entry;
};

static LIST_HEAD(collection);

/*
 * The module allocates `number` instances (default: 3) of struct data
 * and adds them to `collection`.
 */
static int number = 3;
module_param(number, int, 0);

int krg_init(void)
{
  struct data *data;
  int i;

  /* we want our list to have some length, no more than 1000 */
```

Listing 42: W4/krg.c 1/4

```c
  if (number <= 0 || number > 1000)
    return -EINVAL;

  /* TODO: ensure appropriate `number` of iterations */
  for (i = 0; i < 0; ++i) {
    /*
     * TODO: allocate a struct data
     * use kzalloc()
     */
    data = NULL;
    if (!data)
      goto no_memory;

    /* TODO: set the id member of struct data to i */

    /*
     * TODO: add struct data after the last element of collection
     * use list_add_tail()
     */

    printk("%s added:%d@%px\n", __func__, data->id, data);
  }

  return 0;

no_memory:
  while (i--) {
    /*
     * TODO: let `data` point to last element of collection
     * use list_last_entry()
     */

    /*
```

Listing 43: W4/krg.c 2/4

```
     * TODO: remove the element from the collection
     * use list_del()
     */

    /*
     * TODO: free the memory occupied by the removed element
     * use kfree()
     */
  }
  return -ENOMEM;
}
module_init(krg_init);

void krg_exit(void)
{
  struct data *data;

  while (number--) {
    /*
     * TODO: let 'data' point to last element of collection
     * use list_last_entry()
     */

    /*
     * TODO: remove the element from the collection
     * use list_del()
     */

    printk("%s removed:%d@%px\n", __func__, data->id, data);

    /*
     * TODO: free the memory occupied by the removed element
     * use kfree()
```

Listing 44: W4/krg.c 3/4

```
     */
  }
}
module_exit(krg_exit);

MODULE_LICENSE("GPL");
```

Listing 45: W4/krg.c 4/4

```
1. Analyze, compile and deploy ‘Hello, world‘ written in ARM assembly

See hello.S

use the provided compile_hello.sh

__NR_write
__NR_exit

symbolic numbers of write and exit syscalls

necessary ARM assembly:

mov rX, <immediate> - store <immediate> value in register rX
ldr rX, <address> - store <address> value in register rX
svc <immediate>   - trigger software interrupt number <immediate>

2. Run it in the virtual machine
```

Listing 46: W5/TODO.txt

```
#!/bin/bash

LINUX_SOURCE=../../linux-4.18
#
# IMPORTANT!
#
# run these commands once before running this script:
#
# cd ${LINUX_SOURCE}
# make ARCH=arm headers_install
#
arm-linux-gnueabi-gcc hello.S -c -static -I${LINUX_SOURCE}/usr/include
arm-linux-gnueabi-ld hello.o -o hello
```

Listing 47: W5/compile_hello.sh

```
#include "linux/unistd.h"
.data
hello_string:
      .ascii "Hello World\n"
.text
.globl _start
_start:
      mov r7, #__NR_write
      mov r0, #1
      ldr r1,=hello_string
      mov r2, #12
      svc #0

      mov r7, #__NR_exit
      mov r0, #0
      svc #0
```

Listing 48: W5/hello.S

```
1. Analyze the userspace program (syscall.c) and discover what it does

2. Analyze arch/arm/kernel/sys_arm.c in kernel sources and fill the blanks
as instructed inline

3. Rebuild the kernel

Remember about ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-

4. Try running ./syscall in the virtual machine

  ./syscall -p <number> to put element into buffer
  ./syscall -g to get element from buffer
  You might need to put either into background with &
  (e.g. while getting from empty buffer)

When trying to get from empty buffer the process will sleep waiting
on the semaphore. If you don't put the process into background you
have no other shell available to put the element into buffer.
Converse situation happens when trying to put into full buffer.
```

Listing 49: W6/TODO.txt

```c
#include <ctype.h>
#include <errno.h>
#include <getopt.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "linux/unistd.h"

int main(int argc, char *argv[])
{
  int value;

  while ((value = getopt(argc, argv, "p:g")) != -1)
    switch (value) {
    case 'p':
      if (!isdigit(optarg[0])) {
        fprintf(stderr, "Cannot␣parse␣put␣argument!\n");
        exit(EXIT_FAILURE);
      }
      value = atoi(optarg);
      return (syscall(__NR_krg_put, value) < 0) ? errno : 0;

    case 'g':
      value = syscall(__NR_krg_get);
      return (value < 0) ? errno : 0;

    default:
      exit(EXIT_FAILURE);
    }

  exit(EXIT_SUCCESS);
}
```

Listing 50: W6/syscall.c