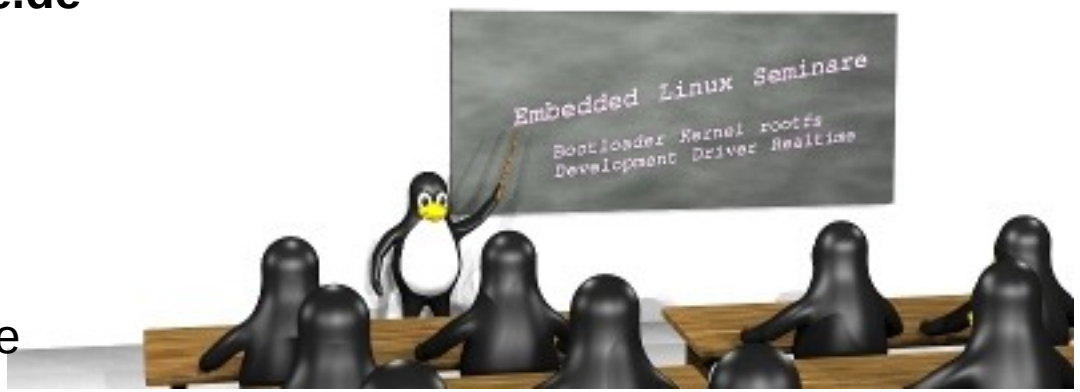
Embedded-Linux-Seminare

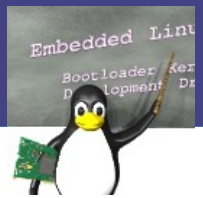
Boot Prozess

http://www.embedded-linux-seminare.de

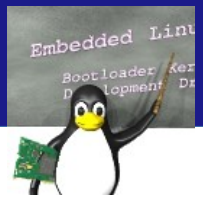
Diplom-Physiker Peter Börner
Spandauer Weg 4
37085 Göttingen

Tel.: 0551-7703465
Mail: info@embedded-linux-seminare.de





Kopier-Rechte



Boot Prozess

Bootloader

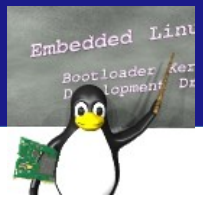
- Wird durch die Hardware von einer definierten Stelle im ROM/Flash ausgeführt
- Initialisiert die Unterstützung für das Gerät auf dem das Kernel Image gespeichert ist (lokales Medium, Netzwerk, USB-Stick, CD-ROM, …)
- Lädt den Kernel in den Arbeitsspeicher
- Startet den Kernel mit spezifizierten Kommandozeilen Parametern

Kernel

- Dekomprimiert sich selber
- Initialisiert den zentralen Kernel und statisch gelinkte Treiber
- Mountet das Root Filesystem
- Startet das erste Userland Programm (durch den *init* Kernel Parameter spezifiziert)

Erstes Userland Programm

- Konfiguriert Userspace und startet System Dienste



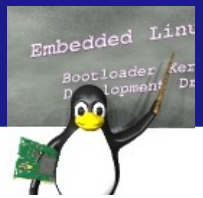
Boot Prozess

Bootloader
- Wird durch die Hardware von einer definierten Stelle im ROM/Flash ausgeführt
- Initialisiert die Unterstützung für das Gerät auf dem das Kernel Image gespeichert ist (lokales Medium, Netzwerk, USB-Stick, CD-ROM, …)
- Lädt Initramfs
- Lädt den Kernel in den Arbeitsspeicher
- Startet den Kernel mit spezifizierten Kommandozeilen Parametern

Initramfs
- Root Filesystem
- Wird im Arbeitsspeicher instaliert
- Enthält z.B. nachladbare Module für den Kernel (z.B. Zugriff auf das Medium auf dem der Kernel gespeichert ist)
- Wird später durch „echtes“ Root Filesystem ersetzt

Kernel
- Dekomprimiert sich selber
- Initialisiert den zentralen Kernel und statisch gelinkte Treiber
- Mountet das Root Filesystem
- Startet das erste Userland Programm (durch den *init* Kernel Parameter spezifiziert)

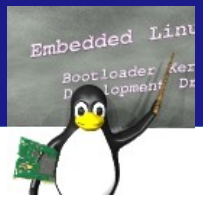
Erstes Userland Programm
- Konfiguriert Userspace und startet System Dienste



Kernel Command Line Parameter

Dem Linux Kernel können zur Bootzeit Parameter übergeben werden

- Kernel Command Line Argumente sind Teil der Bootloader Konfiguration
- Sie werden vom Bootloader an eine Stelle im Arbeitsspeicher kopiert, an denen der Kernel sie erwartet
- Nützlich um das Verhalten des Kernels zu verändern ohne ihn neu erstellen zu müssen
- Nützlich um Kernel- und Treiber-Initialisierung ohne Userspace Skripte umzusetzen.



Kernel Command Line Beispiel

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

Root Filesystem (Ramdisk)

Root Filesystem Mount Modus

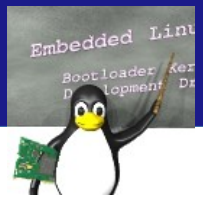
Erstes Userspace Programm

Konsole (seriell)

Konsole (Framebuffer)

Verschiedene Parameter ...

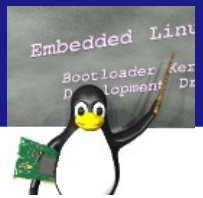
Zahlreiche Parameter sind beschrieben in
`Documentation/kernel-parameters.txt`



Root Filesystem über NFS

Das Root Filesystem kann auch über das Netzwerk von einem Entwicklungsrechner gemountet werden. Dies ist für Entwicklungen im Embedded Bereich gebräuchlich:

- Erlaubt es Dateien im Root Filesystem auszutauschen ohne das System neu starten zu müssen.
- Das Root Filesystem kann wesentlich größer sein als es die lokalen Speichermedien zulassen.



NFS Boot Setup (1)

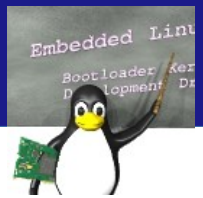
- Auf dem Host (NFS Server)
  - In `/etc/exports` muss das zu exportierende Verzeichnis eingetragen werden:

```
/home/rootfs 192.168.1.111(rw,no_root_squash,no_subtree_check)
```

- Exportieren durch Neustart des Servers
  `/etc/init.d/nfs-kernel-server restart`
  oder durch Aufruf von
  `exportfs -a`

client address

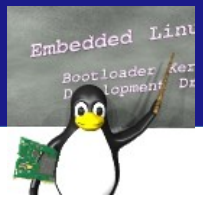
NFS server options



NFS Boot Setup (2)

- Auf dem Target (NFS Client)
  - Konfigurations Optionen des Kernesl setzen
    - `CONFIG_NFS_FS=y`
    - `CONFIG_ROOT_NFS=y`
- Command Line Parameter:

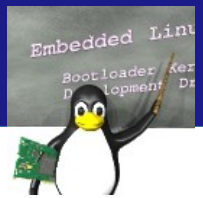
```
root=/dev/nfs \
ip=192.168.1.111 \
nfsroot=192.168.1.110:/home/nfsroot
```



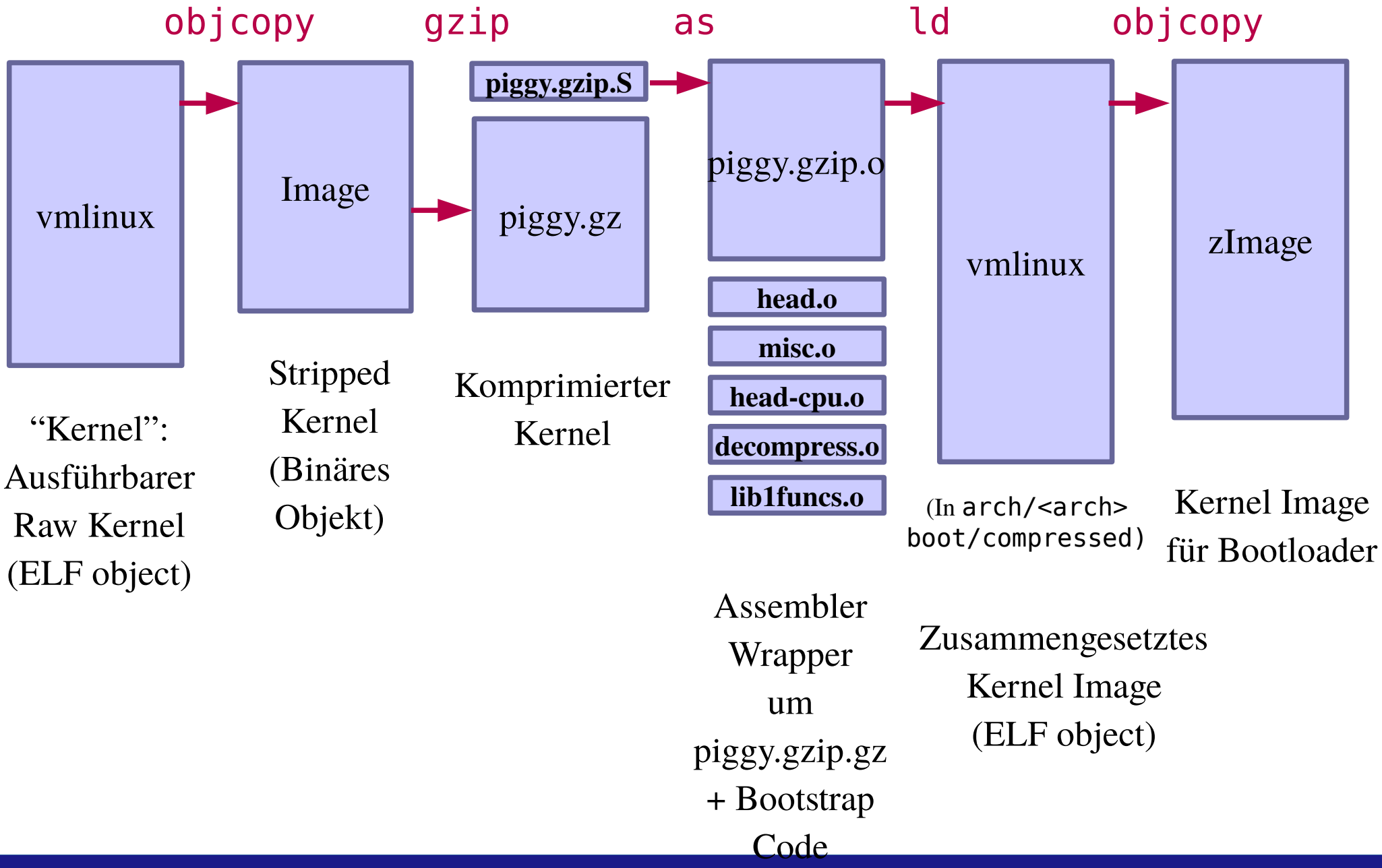
Kernel Bootstrap (1)

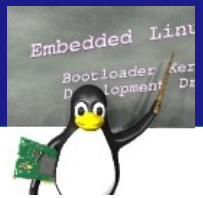
Wie der Kernel sich selbst auspackt ist beim Bauvorgang zu erkennen
Beispiel auf ARM (PXA Cpu) in Linux 2.6.36:

```
...
LD      vmlinux
SYSMAP  System.map
SYSMAP  .tmp_System.map
OBJCOPY arch/arm/boot/Image
Kernel: arch/arm/boot/Image is ready
AS      arch/arm/boot/compressed/head.o
GZIP    arch/arm/boot/compressed/piggy.gzip
AS      arch/arm/boot/compressed/piggy.gzip.o
CC      arch/arm/boot/compressed/misc.o
CC      arch/arm/boot/compressed/decompress.o
AS      arch/arm/boot/compressed/head-xscale.o
SHIPPED arch/arm/boot/compressed/lib1funcs.S
AS      arch/arm/boot/compressed/lib1funcs.o
LD      arch/arm/boot/compressed/vmlinux
OBJCOPY arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
...
```



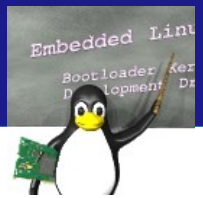
Kernel Bootstrap (2)





Bootstrap Code

- `head.o`:
  Architektur spezifischer Initialisierungs Code.
  Dies wird vom Bootloader ausgeführt
- `head-cpu.o` (im Beispiel `head-xscale.o`):
  CPU spezifischer Initialisierungs Code
- `decompress.o`, `misc.o:`
  Decompressions Code
- `lib1funcs.o`:
  Optimierte ARM Routinen (nur ARM)

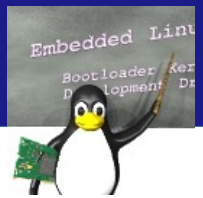


Bootstrap Code Aufgaben

Hauptarbeit liegt bei `head.o`:

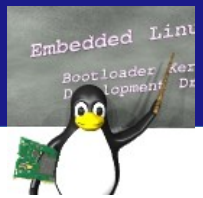
- Architektur, Prozessor und Machinen Typ überprüfen.
- MMU, Page Table Einträge konfigurieren und virtuellen Speicher enablen.
- Ruft die `start_kernel` Funktion in `init/main.c` auf.

  Gleicher Code für alle Architekturen.



Wichtigste Aktionen von start_kernel

- Ruft `setup_arch(&command_line)` auf.
  (Funktion istt definiert in `arch/<arch>/kernel/setup.c`),
  Kopiert die Command Line von dort wo der Bootloader sie hinterlassen hat.
  - Auf `arm` ruft diese Funktion `setup_processor` (zeigt CPU Information) und `setup_machine`(sucht Maschine aus Liste unterstützter Maschinen) auf.
- Initialisiert die Konsole so früh wie möglich (wegen Fehlermeldungen)
- Initialisiert zahlreiche Subsysteme
- Ruft gegebenenfalls `rest_init` auf.

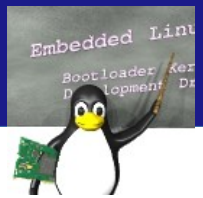


rest_init: Starten des init Prozesses

Startet einen neuen Kernel Thread, der später zum init Prozess wird

```
static noinline void __init_refok rest_init(void)
        __releases(kernel_lock)
{
        int pid;

        rcu_scheduler_starting();
        /*
         * We need to spawn init first so that it obtains pid 1, however
         * the init task will end up wanting to create kthreads, which, if
         * we schedule it before we create kthreadd, will OOPS.
         */
        kernel_thread(kernel_init, NULL, CLONE_FS | CLONE_SIGHAND);
        numa_default_policy();
        pid = kernel_thread(kthreadd, NULL, CLONE_FS | CLONE_FILES);
        rcu_read_lock();
        kthreadd_task = find_task_by_pid_ns(pid, &init_pid_ns);
        rcu_read_unlock();
        complete(&kthreadd_done);

        /*
         * The boot idle thread must execute schedule()
         * at least once to get things moving:
         */
        init_idle_bootup_task(current);
        preempt_enable_no_resched();
        schedule();
        preempt_disable();

        /* Call into cpu_idle with preempt disabled */
        cpu_idle();
}
```

Quelle: Linux 2.6.36



kernel_init

`kernel_init` erledigt zwei Aufgaben:

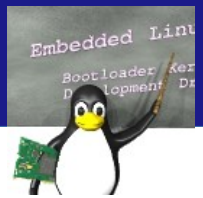
- Ruft `do_basic_setup` auf.

  Starte Geräte Initialisierung:

  (Linux 2.6.36 Code Ausschnitt):

```
static void __init do_basic_setup(void)
{
        cpuset_init_smp();
        usermodehelper_init();
        init_tmpfs();
        driver_init();
        init_irq_proc();
        do_ctors();
        do_initcalls();
}
```

- Ruft `init_post` auf

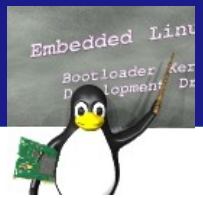


do_initcalls

Ruft hinzufügbare Einsprungpunkte auf, die mit den Makros (s.u.) registriert worden sind.
Vorteil: Der generische Code muss nichts darüber wissen.

```
/*
 * A "pure" initcall has no dependencies on anything else, and purely
 * initializes variables that couldn't be statically initialized.
 *
 * This only exists for built-in code, not for modules.
 */
#define pure_initcall(fn)               __define_initcall("0",fn,1)

#define core_initcall(fn)               __define_initcall("1",fn,1)
#define core_initcall_sync(fn)          __define_initcall("1s",fn,1s)
#define postcore_initcall(fn)           __define_initcall("2",fn,2)
#define postcore_initcall_sync(fn)      __define_initcall("2s",fn,2s)
#define arch_initcall(fn)               __define_initcall("3",fn,3)
#define arch_initcall_sync(fn)          __define_initcall("3s",fn,3s)
#define subsys_initcall(fn)             __define_initcall("4",fn,4)
#define subsys_initcall_sync(fn)        __define_initcall("4s",fn,4s)
#define fs_initcall(fn)                 __define_initcall("5",fn,5)
#define fs_initcall_sync(fn)            __define_initcall("5s",fn,5s)
#define rootfs_initcall(fn)             __define_initcall("rootfs",fn,rootfs)
#define device_initcall(fn)             __define_initcall("6",fn,6)
#define device_initcall_sync(fn)        __define_initcall("6s",fn,6s)
#define late_initcall(fn)               __define_initcall("7",fn,7)
#define late_initcall_sync(fn)          __define_initcall("7s",fn,7s)
```

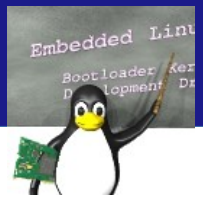
Definiert in `include/linux/init.h`



initcall Beispiel

`arch/arm/mach-pxa/lpd270.c` (Linux 2.6.36)

```
static int __init lpd270_irq_device_init(void)
{
        int ret = -ENODEV;
        if (machine_is_logicpd_pxa270()) {
                ret = sysdev_class_register(&lpd270_irq_sysclass);
                if (ret == 0)
                        ret = sysdev_register(&lpd270_irq_device);
        }
        return ret;
}

device_initcall(lpd270_irq_device_init);
```



init_post

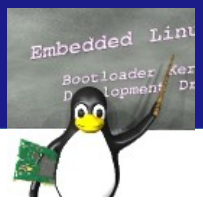
Letzter Schritt beim Linux Bootprozess

- Versucht als erstes eine Konsole zu öffnen
- Versucht dann den init Prozess zu starten durch Umsetzen des Kernel Threads in Userspace Prozess

init_post Code

```
static noinline int init_post(void)
        __releases(kernel_lock)
{
        /* need to finish all async __init code before freeing the memory */
        async_synchronize_full();
        free_initmem();
        mark_rodata_ro();
        system_state = SYSTEM_RUNNING;
        numa_default_policy();


        current->signal->flags |= SIGNAL_UNKILLABLE;

        if (ramdisk_execute_command) {
                run_init_process(ramdisk_execute_command);
                printk(KERN_WARNING "Failed to execute %s\n",
                                ramdisk_execute_command);
        }

        /*
         * We try each of these until one succeeds.
         *
         * The Bourne shell can be used instead of init if we are
         * trying to recover a really broken machine.
         */
        if (execute_command) {
                run_init_process(execute_command);
                printk(KERN_WARNING "Failed to execute %s.  Attempting "
                                        "defaults...\n", execute_command);
        }
        run_init_process("/sbin/init");
        run_init_process("/etc/init");
        run_init_process("/bin/init");
        run_init_process("/bin/sh");

        panic("No init found.  Try passing init= option to kernel. "
              "See Linux Documentation/init.txt for guidance.");
}
```

Quelle:
`init/main.c`
in Linux 2.6.36



Kernel Initialisierungs Graph