Linux Kernel Inside

aneb kterak sobě driver napsati

Martin Mareš, mj@ucw.cz

1. Úvodem

- Motto: *Q: "How to start hacking Linux?" A: "vi /boot/vmlinuz"*
- Základním cílem této přednášky je přiblížit potenciálním autorům driverů (pod heslem „Národ sobě"), jak vypadá Linuxové jádro uvnitř a jakými pravidly je vhodné se při psaní „jaderného[1]" kódu řídit.
- Tento text odpovídá Linuxu 2.1 (a doufejme, že také 2.2), předchozí verze se v mnohých detailech liší (např. co se týče user-accessu a SMP).
- Kernel funguje jako interface: na jedné straně jsou uživatelské procesy komunikující prostřednictvím systémových volání, na druhé straně hardware ovládaný pomocí I/O operací.
- Podpora více architektur (32 i 64-bitových), symetrický multiprocessing (SMP).
- Rozdělení kódu: generický kód, architektury, subarchitektury, skupiny driverů.
- Standardní model vývoje: Všichni píší patche, Linus rozhoduje, co se do jádra zařadí a co (což je častější) nezařadí. Výhody: Mnoho lidí patche čte ⇒ často jsou chyby odhaleny ještě před tím, než se projeví. Nevýhody: Mírná chaotičnost návrhu[2].
- Standardní model dokumentace[3], adresář `Documentation/`, Kernel Hacker's Guide.
- Komunikace mezi developery kernelu: `linux-kernel@vger.rutgers.edu`
- Kdo je kdo: `CREDITS`, `MAINTAINERS`.

Jazyk:

- C, žádné C++ (s možnou výjimkou komentářů `//`)
- Extenze GCC (zejména co se týče inline assembleru a inicializace struktur)
- Nejsou k dispozici standardní knihovny, ba ani standardní includy.
- Hierarchie kernelových includů: `<linux/...>`, `<asm/...>`, `<net/...>`.
- Konfigurace jádra: `<linux/config.h>`.
- Základní kernelové funkce:
  operace se stringy a bloky paměti (`<linux/string.h>`),
  printk, *sprintf*, *panic* (`<linux/kernel.h>`)
- Není možno používat typy *float* a *double*.
- Neexistuje *errno*, chybové kódy se vrací jako záporná čísla.
- Možnost označení částí programu a dat jako inicializačních (po bootu uvolněno): `__init`, `__initdata`. Zatím bohužel u modulů ignorováno.

Organizace paměti:

- Adresy fyzické, sběrnicové, virtuální a uživatelské.
- K jiným než virtuálním adresám není možno přistupovat přímo.
- Funkce *phys_to_virt*, *virt_to_phys*, *virt_to_bus*, *bus_to_virt*.

2. Volání jádra

- Jak se vůbec jádro dostane ke slovu?[4]
- System calls (funkce *sys_xxx*, případně *do_xxx*).
- Výjimky (page fault, TLB miss, illegal instruction apod.).
- Přerušení (od periferií či IPI).

Přístup k uživatelským datům:

- `<asm/uaccess.h>`
- Nelze provádět z interruptu, jelikož může blokovat.
- Kernelové kontra uživatelské pointery, *set_fs*, *get_fs*, `USER_DS`, `KERNEL_DS`.
- *get_user(var,ptr)*, *put_user(val,ptr)*, *copy_from_user*, *copy_to_user*
- *strcpy_from_user*, *strlen_user*, *clear_user*
- Vícenásobné přístupy: *access_ok*, *__get_user* a spol.

[1] kernelového, nikoliv nukleárního

[2] Světem obchází přízrak a jmenuje se Binární Kompatibilita.

[3] Známý též pod heslem „Use the source, Luke!"

[4] Možná by byla korektnější otázka „Jak se vůbec někdo jiný než jádro dostane ke slovu?", ale to už příliš zavání problémy typu „slepice nebo vejce?"

3. Periferie

- `<asm/io.h>`

I/O porty:

- *outb(val,port)*, *outw*, *outl*
- Zpomalené verze: *outb_p*, *outw_p*, *outl_p*
- Blokový přenos: *outsb*, *outsw*, *outsl*, *insb*, *insw*, *insl*
- Rezervování portů: `<linux/ioport.h>`, *request_region*, *release_region*, *check_region*

Paměťově mapované periferie:

- Nejprve je nutno zařízení namapovat do kernelového adresního prostoru: *ioremap(start,len)*, *ioremap_nocache*
- Přístup k namapovaným oblastem: *readb(addr)*, *readw*, *readl*, *writeb(b,addr)*, *writew*, *writel*
- Blokové přenosy: *memcpy_fromio*, *memcpy_toio*, *memset_io*
- Zatím žádné rezervování (2.3?)
- Synchronizace: *barrier*

Přerušení:

- *request_irq(irq,handler,flags,name,data)*, *free_irq(irq)*
- Globální zákaz přerušení: *cli*, *sti*, *save_flags*, *restore_flags* (flags jsou *long int*!).
- Lokálně pro jeden procesor: *__cli*, *__sti*, *__save_flags*, *__restore_flags*
- Pro konkrétní přerušení: *disable_irq*, *enable_irq*
- IRQ probing: *probe_irq_on*, *probe_irq_off*
- Pozor, číslo interruptu je *unsigned int.*

Bottom Halves:

- Časově náročnější, leč nekritickou obsluhu periferií je vhodné provádět ne přímo v interrupt handleru, nýbrž až později. Od toho tu jsou BH.
- `<linux/interrupt.h>`: *xxx_BH*, *init_bh(nr,routine)*, *remove_bh*, *mark_bh*
- Typické použití: Interrupt handler uloží přišedší data do bufferu a zavolá *mark_bh.* V obsluze BH se poté data zpracovávají.
- Locking: *start_bh_atomic*, *end_bh_atomic*, *disable_bh*, *enable_bh*
- Často jsou vhodnějším[5] mechanismem dynamické BH (jinak též *task lists*, viz `<linux/tqueue.h>`): stačí vytvořit strukturu *tq_struct*, zavolat *queue_task(&tq,queue)* a případně označit BH handler, který se o příslušnou frontu stará. Běžně se používají fronty *tq_immediate* (`IMMEDIATE_BH`), *tq_scheduler* (spustí se při vstupu do scheduleru) a *tq_timer* (spuštěno při následujícím tiku časovače; používáno pro „autobuffering" u tty vrstvy).

Identifikace zařízení:

- ISA: Auto-probing
- PCI: `<linux/pci.h>`, *struct pci_dev*, *pci_find_device*, *pci_read_config_byte* apod.

4. Scheduling

- Je-li proces uvnitř jádra (to jest zavolal-li syscall či vyvolal exception), běží, dokud se sám nevzdá procesoru voláním funkce *schedule* (plus: eliminuje spoustu problémů se zamykáním; minus: proces, který se „zaběhnul" v jádru, zablokuje systém[6]).
- Proces může být kdykoliv přerušen interruptem či bottom half handlerem (následkem interruptu), pokud to není explicitně zakázáno.
- Na ostatních procesorech mohou běžet libovolné procesy, ale i interrupty[7].
- Na každém procesoru vždy běží alespoň *idle task*[8].

Procesy:

- Každý proces je representován strukturou *task_struct*, na aktuální proces na lokálním procesoru ukazuje pointer *current.*
- Každý proces má svůj kernelový zásobník[9], který je vlastně součástí *task_struct*, takže *current* je možno zjistit ze stack pointeru.

---

[5] Hlavní výhodou je, že jejich počet není nijak limitován, na rozdíl od BH klasických, kterých může být max. 32.

[6] To není příliš na závadu, jelikož se to může stát pouze v případě chyby v jádru, která ovšem může způsobit daleko podivuhodnější věci...

[7] Ovšem naštěstí nemůže být jeden proces či interrupt spuštěn na dvou procesorech současně.

[8] Idle task nikdy nespí a timer stále bdí...

[9] Dalo by se to vyřešit daleko lépe, ale zdá se, že firma Intel pochopila užitečnost interrupt stacků, a tak se tvrdošíjně brání je implementovat.

- Linuxový přístup ke threadům: *clone*, sdílení adresního prostoru, tabulek signal-handlerů či file-descriptorů.
- Kernelové thready: běží v adresním prostoru jádra.
- Stavy procesu: running, interruptible sleep, uninterruptible sleep, zombie, stopped.

Přímé volání scheduleru:

- Stačí zavolat *schedule* a předtím nastavit *current→state* na požadovaný stav procesu.
- Spící proces lze probudit pomocí *wake_up_process*.

Fronty procesů:

- `<linux/wait.h>`: *struct wait_queue*
- *init_wait_queue(&q)*, *sleep_on(&q)*, *interruptible_sleep_on(&q)*, *wake_up(&q)*, *wake_up_interruptible(&q)*

Semafory:

- `<asm/semaphore.h>`
- Vhodné k ochraně kritických sekcí proti ostatním procesům, nikoliv však proti interruptům či BH.
- `struct semaphore sem = MUTEX` (inicializace na 1), případně `MUTEX_LOCKED` (na 0).
- *up*, *down*

Timery:

- `<linux/timer.h>`
- Interrupt od časovače přichází `HZ` krát za sekundu, po každé se spustí `TIMER_BH`, zvýší se proměnná *jiffies* a zpracují se timery.
- Statické timery (`TIMER_xxx`) – historické, nepoužívat v novém kódu.
- Dynamické timery: struktura *timer_list*. Stačí vyplnit položky *expires* (v jiffies), *function* a *data*, zavolat *init_timer(&t)* a *add_timer(&t)*. Poté je možno zrušit voláním *del_timer* či modifikovat pomocí *mod_timer*.
- V obsluze timeru je možno volat *add_timer* pro znovuspuštění timeru.

Spinlocky:

- `<asm/spinlock.h>`, typ *spinlock_t*
- Používají se pro zamykání *krátkých* kritických sekcí vůči ostatním procesorům, na non-SMP kernelu nedělají nic.
- `spinlock_t lock = SPIN_LOCK_UNLOCKED`
- *spin_lock*, *spin_unlock*, *spin_try_lock*
- IRQ-safe verze: *spin_lock_irq*, *spin_unlock_irq*, *spin_lock_irqsave*, *spin_unlock_irqrestore* (na non-SMP jádrech ekvivalentní s *cli*, resp. *sti*.
- Read-write verze (*rwlock_t*): *read_lock*, *read_unlock*, *write_lock*, *write_unlock* a IRQ-safe verze.

Atomické operace:

- `<asm/atomic.h>`, typ *atomic_t*
- Proměnné atomické vůči interruptům i ostatním procesorům. Velikostí ekvivalentní s *int*em.
- *atomic_read*, *atomic_set*, *atomic_add*, *atomic_sub*, *atomic_inc*, *atomic_dec*, *atomic_dec_and_test*
- *xchg*, *tas* (obojí v `<asm/system.h>`, definovány pro *long int*)

5. Správa paměti

- Alokace po stránkách: funkce *get_free_page(flags)*, *__get_free_page* (bez nulování), *free_page*.
- Flagy: `GFP_KERNEL`, `GFP_USER` (s nižší prioritou), `GFP_ATOMIC` (nemůže blokovat, lze použít z obsluhy přerušení či BH).
- Vícestránkové oblasti: *__get_free_pages(flags, order)*, *free_pages*.
- Malé bloky: *kmalloc(size,flags)*, *kfree(ptr)*.
- Slab-alokátor (mnoho bloků téže velikosti).

6. Portabilita

- Celý Linux je multiplatformní OS ⇒ kód je nutné psát tak, aby fungoval na všech platformách. „Funguje na mém počítači" není postačující kriterium správnosti.[10]

Datové typy:

- Jediné, na co je možne u běžných typů spoléhat, je, že *int* má alespoň 32 bitů, *char* alespoň 8 a že pointer je možno bez úhony přetypovat na *long int* a zpět.
- Proto existují předdefinované kernelové typy *u8*, *s8*, *u16*, *s16*, *u32*, *s32*, *u64*, *s64* fungující na všech platformách stejně.
- Častým problémem je *nerespektování alignmentu* – mnoho procesorů vyžaduje, aby při k-bitovém přístupu byla data zarovnána na násobek k bitů atd.

---

[10] S jednou občasnou výjimkou, a tou je Linus Torvalds osobně.

- Vše, co je modifikováno asynchronně interruptem, musí být *volatile*.
- Vše, co je modifikováno asynchronně jiným procesorem, musí být chráněno lockem nebo být atomické.

Endianita:

- Největším zdrojem problémů s portabilitou je ovšem pořadí ukládání vícebytových čísel, tak řečená *endianita* procesoru[11].
- Linux proto disponuje mnoha funkcemi na konverzi čísel mezi těmito formáty – viz `<asm/byteorder.h>`.
- Je definováno `__LITTLE_ENDIAN` či `__BIG_ENDIAN`, případně `__PDP_ENDIAN`.
- *le16_to_cpu* a *cpu_to_le16*, analogicky pro 32 a 64 bitů.
- Odkaz přes pointer[12]: *le16_to_cpup* (z paměti) a *le16_to_cpus* (do paměti).
- *swab16*, *swab32* a *swab64* – explicitní prohazování bytů.
- Nulu není potřeba konvertovat.

7. Drivery

- Konečně se dostáváme k zlatému hřebu, kterým je slibované psaní driverů.
- Statické kontra modulární drivery; symboly exportované modulům (`EXPORT_SYMBOL`, `/proc/ksyms`), modversions (omyl historie?).
- Kolik typů zařízení, tolik interfaců.
- Přidělování čísel zařízení.
- Generické vrstvy: parport, SCSI midlayer, console…

Inicializace:

- Pokud je driver zakompilován do jádra, je volání jeho inicializační funkce uvedeno na některém z init call listů (viz *device_setup* v `drivers/block/genhd.c`).
- Pokud je driver instalován jako modul, volá se jeho funkce *init_module*, při deinstalaci modulu pak funkce *cleanup_module*. Modul se musí starat o svůj *use count* (makro `MOD_INC_USE_COUNT` v `<linux/modules.h>`).
- Detekce hardwaru, rezervování portů, inicializace hardwaru.
- Zaregistrování struktury popisující driver. Pro znaková zařízení to je struktura *file_operations* obsahující pointery na funkce provádějící jednotlivé základní operace (*open*, *flush*, *release*, *read*, *write*, *llseek*, *readdir*, *poll*, *ioctl*, *mmap*, *fsync*, *fasync*, …).
- Možnosti konfigurace driveru: příkazová řádka jádra (viz `init/main.c`), parametry modulů.

Ladění driverů:

- Základní metodou ladění stále zůstávají ladící tisky.[13]
- Post-mortem analýza – hláška Oops!
- Zkoumání zkompilovaného kernelu (`vmlinux`) debuggerem, případně sledování běžícího jádra skrz soubor `/proc/kcore`.
- `ktrace`, remote GDB
- Profiling: `CONFIG_PROFILE`, `kprof`
- Magic SysRq keys dokáží být skvělými pomocníky.

Výhled do budoucnosti – Driver Descriptions:

- Výčet podporovaného hardwaru (jednodušší a přehlednější inicializace jádra, možnost automatické instalace driverů u hot-pluggable zařízení – PCMCIA, CARDBUS)
- Pokus o unifikaci předávání parametrů
- Jednotný pohled na všechny sběrnice (samozřejmě v mezích možností – „hon na zařízení“ [ISA] kontra „daňové přiznání“ [PCI, SBUS a prakticky všechny ostatní rozumné sběrnice)
- ProcFS interface ke sběrnicím i driverům.

Happy Hacking

[11] Podle Gulliverových cest – Liliputáni se dělili na dvě skupiny, které se přely o to, na kterém konci se mají naklepávat vajíčka (Little Endians a Big Endians). Počítačová terminologie toto přejímá a s hravostí sobě vlastní přidává výraz Middle Endians pro neobvyklá pořadí typu 2143. Jiným známým termínem pro tentýž jev je *byte sex*.

[12] Některé architektury mají k dispozici přístup k paměti zahrnující konverzi endianity.

[13] Opravdový kernelový guru ovšem používá tradiční metodu „kouknu a vidím“.