

PB173 – Ovladače jádra – Linux

III. Ladění jádra

Jiri Slaby

Fakulta informatiky
Masarykova univerzita

4. 10. 2016

LDD3 kap. 4 (zastaralá)

Jak zjistit chybu v jádře

- Ladicími výpisy
- Výstupem přes soubor (obsah velkých bufferů apod.)
- Pád \Rightarrow analýza Oops
- Regrese \Rightarrow `git bisect`
- Automatické nástroje
- (Debugger – `kgdb` a další)
- ...

Sekce 1

Ladění jádra

- `printk` již známe
- Před formátovací řetězec se vkládá `KERN_*`
 - Řetězec ve formě `<cislo>` nebo `\001cislo`
 - Sděluje úroveň (důležitost) zprávy
 - Definované v `linux/kernel.h`
 - Zkratky: `pr_err`, `pr_info`, ...

Úroveň ovlivňuje, co se kde zobrazí

- dmesg nastavit nelze
 - Obsahuje vždy všechno
- Nastavení /var/log/* (syslogu) závislé na systému

Obvyklé nastavení

| Důležitost | Cíl | | |
|------------|-------|------------|---------|
| | dmesg | /var/log/* | Konzole |
| KERN_EMERG | ✓ | ✓ | ✓ |
| KERN_INFO | ✓ | ✓ | ✗ |
| KERN_DEBUG | ✓ | ✗ | ✗ |
| Zapisuje: | jádro | syslog* | jádro |

- Příkazem: `dmesg -n cislo`
- Zprávy s vyšší úrovní se objeví na konzoli
 - Vyšší úroveň = nižší číslo
- Hodnota 8 (debug) \Rightarrow všechno dění
 - Např. při „hard-locku“

Demo: `pb173/03 + dmesg -n`

Práce s úrovněmi

- 1 Přidejte si do kódu modulu `WARN_ON(1)`
- 2 Zjistěte s jakou úrovní se varování vypisuje
 - Pomocí postupného nastavování `dmesg -n`

Volba úrovně

- Nechceme 10G /var/log/messages
 - Ladicí výpisy: KERN_DEBUG
- Nechceme spoustu hlášek na konzoli
 - Obvyklé chyby: KERN_WARNING, KERN_ERR

Příklady

```
printk (KERN_EMERG "Thread overran stack, or stack corrupted\n");
printk (KERN_ALERT "Unhandled fault: %s (0x%03x) at 0x%08lx\n", ...
printk (KERN_CRIT "Invalid DMA channel for ata\n");
printk (KERN_ERR "button.c: Not enough memory\n");
printk (KERN_WARNING "Only using first memory bank\n");
printk (KERN_NOTICE "PCI: Starting initialization.\n");
printk (KERN_INFO "SMP mode deactivated.\n");
printk (KERN_DEBUG "cosa%d: microcode started\n", ...
```


Co vše může být konzole pro výpisy

- `tty0-tty*` (co když stroj umře?)
- Sériová konzole (`ttyS*`)
- Síťová konzole (`netconsole`)
- Všechno, co se zaregistruje pomocí `register_console`

Nastavení pomocí parametru jádra (modulu)

- `console=tty1`
- `console=ttyS0`
- `netconsole=...`

[Documentation/console/console.txt](#)

[Documentation/serial-console.txt](#)

Použití netconsole ve virtuálním stroji

- 1 Na hostitelovi: `nc -ul 0.0.0.0 60000`
- 2 `dmesg -n 8` (vypisovat vše)
- 3 `modinfo netconsole`
- 4 `modprobe netconsole netconsole=@/ens3,60000@10.0.2.2/`
- 5 Např. `echo h >/proc/sysrq-trigger`

Více informací v [Documentation/networking/netconsole.txt](#)

- Mnoho dat

- Znepřehlednění (typické pro ACPI, iwlwifi)
- Nestíhá se posílat na konzoli (ztráta informací)
- *Částečné řešení* (`linux/printk.h`):

```
if ( printk_ratelimit ()  
    printk (...) ;
```

- Binární data

- `dmesg` zobrazuje jen textová data
- *Částečné řešení* (`linux/kernel.h`):

```
print_hex_dump_bytes(...) // Vyzaduje DEBUG  
print_hex_dump(...)
```

Mnoho binárních dat se předává pomocí speciálních souborů

Sekce 2

debugfs

- Předávání ladicích dat přes soubor
- Většinou v `/sys/kernel/debug/`
 - Pokud ne: `mount -t debugfs none /sys/kernel/debug/`
- `Documentation/filesystems/debugfs.txt`
- `linux/debugfs.h`
- Vytvoření
 - Adresář: `debugfs_create_dir`
 - Symbolický odkaz: `debugfs_create_symlink`
 - Vystavit proměnnou: `debugfs_create_u{8,16,32,bool}`
 - Celé `file_operations`: `debugfs_create_file`
 - Lze si předat data, jsou pak v `open` v `inode->i_private`
- Smazání
 - 1 položka: `debugfs_remove`
 - Celý podstrom: `debugfs_remove_recursive`

Vytvoření položek v debugfs

- 1 Vytvořte adresář (`debugfs_create_dir`)
- 2 Vytvořte v něm soubor (`debugfs_create_u16`)
 - Vrací nějakou hodnotu (např. 11320)
- 3 Přeložte, vložte do systému (`make, insmod`)
- 4 Přečtěte soubor
 - `cat /sys/kernel/debug/<dir>/<file>`

Sekce 3

sysrq + strace

- System-request ([Documentation/sysrq.txt](#))
 - Klávesnicí: Alt-PrintScreen
 - Vzdáleně: /proc/sysrq-trigger
- Funguje (většinou) i po pádu systému
- Zapnutí: `sysctl kernel.sysrq=1`

Příklady SysRq

| | |
|---------|---------------------------|
| h | zkrácená nápověda |
| r | „reset klávesnice“ |
| p | výpis tras procesů na CPU |
| t | výpis tras všech procesů |
| w | výpis blokujících procesů |
| s-u-s-b | sync + unmount + reboot |

Sysrq

- 1 Vypište si nápovědu (`sysrq-h`)
- 2 Vypište si trasy procesů na CPU (`sysrq-p`)
- 3 Vypište si informace o paměti (`sysrq-m`)

Pozn.: v Qemu lze vyzkoušet v ovládací konzoli (Ctrl-Alt-2) příkazem `sendkey alt-sysrq-písmeno`

- Chci-li znát komunikaci jádro ↔ program
- `man strace`
- Zachytává syscalls
- Dekóduje a vypisuje jejich parametry na STDOUT

Demo: `strace -e open,read cat .../debug/...`

Analýza Oops

```
my_init(void) { char *ptr = (void *)5; *ptr = 5; }
```

BUG: unable to handle kernel NULL pointer dereference at 0000000000000005

IP: [`< ffffffff80039042 >`] my_init+0x12/0x30 [`pb173`]

PGD 3daff067 PUD 3c713067 PMD 0

...

Modules linked in: `pb173(+)` mperf fuse ...

Pid: 2506, comm: insmod Not tainted 2.6.36-rc4-16-**default** #1 /Bochs

RIP: 0010:[`< ffffffff80039042 >`] [`< ffffffff80039042 >`] my_init+0x12/0x30 [`pb173`]

RSP: 0018:ffff88003c741f28 EFLAGS: 00010292

RAX: 0000000000000017 RBX: ffffffff80039180 RCX: 00000000000000e91

...

CR2: 0000000000000005 CR3: 000000003bb47000 CR4: 000000000000006f0

Process insmod (pid: 2506, ...)

Stack: ...

Call Trace:

[`< ffffffff810002da >`] do_one_initcall+0x3a/0x170

[`< ffffffff8108f69a >`] sys_init_module+0xba/0x210

[`< ffffffff81002efb >`] system_call_fastpath+0x16/0x1b

[`<00007fc3386320ba>`] 0x7fc3386320ba

Code: ... e8 63 c8 41 e1 <c6> 04 25 05 00 00 ...