PB173 – Ovladače jádra – Linux

III. Ladění jádra

Jiri Slaby

ITI, Fakulta informatiky

1. 10. 2013

Ladění jádra

LDD3 kap. 4 (zastaralá)

Jak zjistit chybu v jádře

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

Výpisy

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

Důležitost	Konzole	`/var/log/*`	`dmesg`
vysoká	ANO	ANO	ANO
normální	NE	ANO	ANO
ladicí	NE	NE	ANO
zapisuje	jádro	syslog*	jádro

Obvyklé nastavení

Úroveň ovlivňuje, co se kde zobrazí

Nastavení výpisů

- `dmesg` nastavit nelze
  - Obsahuje vždy všechno
- Nastavení syslogu závislé na systému

Nastavení konzole

- `dmesg -n číslo`
  - *Zprávy s vyšší úrovní se objeví na konzoli*
  - Hodnota 8 $\Rightarrow$ všechno dění
    - Např. při „hard-locku”
- Demo: `pb173/03` + `dmesg -n`

Úkol

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`

Úroveň výpisů

Volba úrovně

- Nechceme 10G `/var/log/messages`
  - Ladicí výpisy: `DEBUG`
- Nechceme spoustu hlášek na konzoli
  - Obvyklé chyby: `WARNING`, `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", ...
```

Cíl výpisů

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`

Úkol

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=@/,60000@10.0.2.2/`
5. Např. `echo h >/proc/sysrq-trigger`

Více informací v `Documentation/networking/netconsole.txt`

Nevýhody výpisů

- 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(...)
```

Mnoho binárních dat se předává pomocí speciálních souborů

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`
- `debugfs_create_dir`,
  `debugfs_create_file` (`file_operations`),
  `debugfs_create_symlink`,
  `debugfs_create_u{8,16,32,bool}`,
  `debugfs_remove`
- Lze si předat `data`, jsou pak v `inode->i_private` (v `open`)

Úkol

Vytvoření položek v debugfs

1. Vytvořit adresář
2. Vytvořit v něm soubor (u16)
   - Vrací např. 11320
3. Přeložit, vložit do systému
4. Přečíst soubor

SysRq

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

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

Příklady SysRq

Úkol

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`

strace

- Chci-li znát komunikaci jádro $\leftrightarrow$ program
- `man strace`
- Zachytává syscally
- 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 00000000000005
IP: [< ffffffffa0039042 >] my_init+0x12/0x30 [pb173]
PGD 3daff067 PUD 3c713067 PMD 0
Oops: 0002 [#1] SMP
last sysfs file : /sys/devices/system/...
CPU 0
Modules linked in: pb173(+) mperf fuse ...
Pid: 2506, comm: insmod Not tainted 2.6.36-rc4-16-default #1 /Bochs
RIP: 0010:[< ffffffffa0039042 >] [< ffffffffa0039042 >] my_init+0x12/0x30 [pb173]
RSP: 0018:ffff88003c741f28 EFLAGS: 00010292
RAX: 0000000000000017 RBX: ffffffffa0039180 RCX: 0000000000000e91
 ...
CR2: 0000000000000005 CR3: 000000003bb47000 CR4: 00000000000006f0
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 ...
```