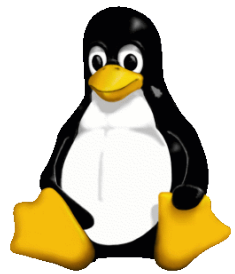
Depurando o Kernel de forma interativa com KGDB

Edjunior Machado
Breno Leitão

FISL 11, 2010



Agenda

Você está corrompido

Linus Torvalds disse:

- "I don't like debuggers. Never have, probably never will"
- "I happen to believe that not having a kernel debugger forces people to think about their problem on a different level than with a debugger."
- "Without a debugger, you tend to think about problems another way."

Como solucionar um problema no kernel

- Lendo o código
- O rei dos depuradores `printk()`[1] ou `trace_printk()`[2]
- Ftrace, LLTng e Systemtap
- **xmon** para PowerPC

E se:

- O problema for no boot
- Se você não sabe onde o problema está ?
- Interativo/Step-by-step
- Comandos on-demand

[1] muitos mili segundos
[2] décimos de microsegundos

```
0:mon> e
cpu 0x0: Vector: 700 (Program Check) at [c0000000035af8b0]
   pc: c000000000496dd0: .skb_pull+0x30/0x50
   lr: c0000000004c55ac: .eth_type_trans+0x3c/0x160
 current = 0xc00000039325570
 paca    = 0xc000000000f62500
   pid   = 15772, comm = ftp
kernel BUG at include/linux/skbuff.h:1132!
0:mon> t
[link register   ] c000004c55ac .eth_type_trans+0x3c/0x160
[c00035afb30] c00000e3c6b0 check_legacy_serial_console+0x0/0x10
[c00035afbc0] d000014049e8 .ixgbe_clean_rx_irq+0x578/0x940 [ixgbe]
[c00035afcf0] d00001405dd4 .ixgbe_clean_rxtx_many+0x124/0x280 [ixgbe]
[c00035afdc0] c000004a7788 .net_rx_action+0x168/0x2e0
...
[c00000003e4a7e30] c00000000000852c syscall_exit+0x0/0x40
--- Exception: c01 (System Call) at 0000008078afbb64
```

Depurando step-by-step

- Surge uma implementação na LinSysSoft no kernel 2.4
- Em 2006 a Wind River assume o código e cria o KGDB light
- Isto é, depuração estritamente via console serial (`kgdboc`)
- Depuração via Ethernet é temporariamente cancelada (`kgdboe`)
- Ingo Molnar adere ao projeto e o KGDB light é aceito *upstream* no kernel 2.6.26 (2008)

"This is a slimmed-down and cleaned up version of KGDB that i've created out of the original patches that we submitted two weeks ago. I went over the kgdb patches with Thomas and we cut out everything that we did not like, and cleaned up the result. KGDB is still just as functional as it was before (i tested it on 32-bit and 64-bit x86) - and any desired extra capability or complexity should be added as a delta improvement, not in this initial merge." --Ingo Molnar

O que é o KGDB

- Extensão que provê mecanismo para depurar o kernel usando GDB
- São necessárias 2 máquinas (uma estará parada no *breakpoint*)
- Atualmente, utiliza a porta serial para comunição
- Multi-arquitetura (x86, x86-64, ARM, Blackfin, MIPS, SH, PowerPC e SPARC)
- Pode ser utilizada em máquinas virtuais

Configurações

- Habilitar suporte ao KGDB no kernel
- Habilitar compilação com *debug info* para informações sobre código fonte
- Desabilitar *write protect* kernel para possibilitar inserção de *breakpoints*

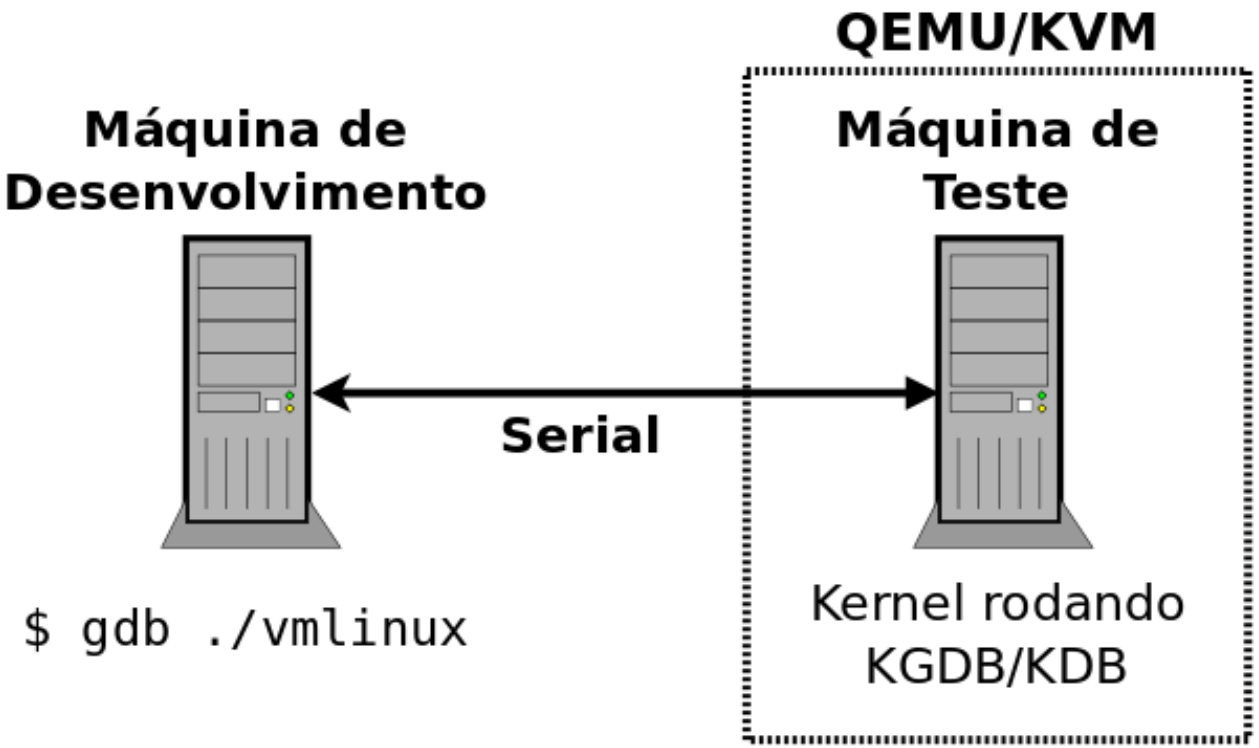
$ make menuconfig

```
General setup  --->
    [*] Prompt for development and/or incomplete code/drivers
Kernel hacking  --->
    -*- Magic SysRq key
    [*] Compile the kernel with debug info
    [*] KGDB: kernel debugger  --->
        --- KGDB: kernel debugger
        <*>   KGDB: use kgdb over the serial console
        [ ]   KGDB: internal test suite
        [ ]   KGDB: Allow debugging with traps in notifiers
        [*]   KGDB_KDB: include kdb frontend for kgdb
        [*]     KGDB_KDB: keyboard as input device
    [ ] Write protect kernel read-only data structures
```

Parâmetros

- **kgdboc=kbd,kms,<serial>,<baudrate>**
  - **kbd**: (e não kdb) suporte ao Kernel Debugger
  - **kms**: integração ao Kernel Mode Setting, para acessar o console toda vez que entrar em modo depuração
  - **<serial>,<baudrate>**: configuração da interface serial
  - Parâmetros também configuraveis através de `/sys/module/kgdboc/parameters/kgdboc`
- **kgdbwait**
  - Kernel é bloqueado na inicialização aguardando conexão do GDB
  - Suporte ao KGDB deve estar compilado *built-in*
- **kgdbcon**
  - Mensagens do kernel são replicadas no GDB enquanto este estiver conectado

Setup


QEMU/KVM
Máquina de Desenvolvimento
Máquina de Teste
Serial
$ gdb ./vmlinux
Kernel rodando KGDB/KDB

Exemplo usando KGDB - *Breakpoints*

- Habilitar KGDB via ttyS0 na máquina de testes

```
# echo ttyS0 > /sys/module/kgdboc/parameters/kgdboc
# echo g > /proc/sysrq-trigger
```

- Na máquina de desenvolvimento, conectar GDB (verificar SERIAL utilizada pelo KVM/QEMU) e incluir um breakpoint na função `icmp_reply`

```
$ gdb ./vmlinux
(gdb) target remote /dev/pts/<SERIAL>
(gdb) break icmp_reply
(gdb) cont
```

- Envie um *ping* para a máquina de testes a partir da máquina de desenvolvimento

```
$ ping 192.168.122.218 -c 1
```

Exemplo usando KGDB - Depurando módulos

- Na máquina de testes, carregar o módulo de rede *dummy*

```
# modprobe dummy
# ifconfig dummy0
# cat /sys/module/dummy/sections/.text
# cat /sys/module/dummy/sections/.data
# echo g > /proc/sysrq-trigger
```

- Na máquina de desenvolvimento, adicionar ao GDB os endereços das seções ELF do módulo carregado e um *breakpoint* para a função `dummy_set_address`

```
$ gdb ./vmlinux
(gdb) target remote /dev/pts/<SERIAL>
(gdb) add-symbol-file drivers/net/dummy.o <ENDEREÇO .text> -s .data <ENDEREÇO .data>
(gdb) break dummy_set_address
(gdb) cont
```

- Na máquina de testes, configurar o endereço MAC da interface `dummy0`

```
# ifconfig dummy0 hw ether 00:00:00:00:00:01
```

- Na máquina de desenvolvimento, redefinir o MTU na estrutura `dev`

```
(gdb) set dev->mtu = 1499
(gdb) cont
```

O que é o KDB

- Kernel DeBugger é um depurador *built-in* mantido pela SGI como *patch* externo desde Linux 2.2
- Não necessita de uma segunda máquina para depuração
- Utiliza apenas teclado + console texto VGA
- Não é um depurador em nível de código
- Útil para verificar `dmesg`, `lsmod`, `ps`, *stack trace*, estado dos registradores
- Também possui suporte a *breakpoints* e modificações na memória
- Acessível através do KGDB (comando `monitor`)

Exemplo usando KDB - *Breakpoints*

- A partir do console da máquina de testes, execute

```
# echo kbd,ttyS0 > /sys/module/kgdboc/parameters/kgdboc
# echo g > /proc/sysrq-trigger
```

- O sistema é direcionado para o prompt do KDB. Então inserimos um *breakpoint* na *syscall* `sys_mkdir`

```
[0]kdb> bp sys_mkdir
[0]kdb> go
```

- De volta ao prompt do sistema, execute um comando mkdir

```
# mkdir dirteste
```

Outros Exemplos

- *Breakpoint* em outras *syscalls* como `sys_sync` (utilizada pelo `sync`)
- Utilizando `monitor` do KGDB para matar processos via KDB
- Carregar um módulo exemplo com algum *bug* providencial

Futuro

- Conclusão do *merge* KGDB + KDB (2.6.35)
- Integração com KMS (Kernel Mode Setting)
- kgdbou (KGDB over USB)
  - Estabilizar suporte USB Console
- kgdboe v2 (KGDB over Ethernet)
  - Melhorias nos drivers de rede (problemas com preempção + *locks*)
  - Ou hardware com fila dedicada (ex.: e1000e)

Referências

- **KGDB Wiki**
  http://kgdb.wiki.kernel.org
- **Using kgdb, kdb and the kernel debugger internals**
  http://kernel.org/pub/linux/kernel/people/jwessel/kdb/
- **IBM developerWorks - Mastering Linux debugging techniques**
  http://www.ibm.com/developerworks/linux/library/l-debug/
- **KGDB Mailing List**
  https://lists.sourceforge.net/lists/listinfo/kgdb-bugreport

Obrigado!

Dúvidas?

Edjunior Machado
edjunior@gmail.com / emachado@linux.vnet.ibm.com
Breno Leitão
breno.leitao@gmail.com / leitao@linux.vnet.ibm.com