Odpluskwianie jądra Linuksa

Michał Węgorek Bartosz Dąbrowski Adam Kotwasiński

26 listopada 2007

Debuggowanie jądra-problemy

1. Aby sprawdzić kawałek kodu jądra potrzebne jest jego skompilowanie.
2. Ciężko powtórzyć błąd, np. może on się ujawniać przy rzadkim przeplocie.
3. Potrzebną dokumentację ciężko znaleźć, jeśli się uda to jest niedokładna bądź niekompletna.
4. Jak debugować coś na czym aktualnie pracujemy? Rozwiązanie: platformy do wirtualizacji.

Debuggowanie jądra-problemy

1. Aby sprawdzić kawałek kodu jądra potrzebne jest jego skompilowanie.
2. Ciężko powtórzyć błąd, np. może on się ujawniać przy rzadkim przeplocie.
3. Potrzebną dokumentację ciężko znaleźć, jeśli się uda to jest niedokładna bądź niekompletna.
4. Jak debugować coś na czym aktualnie pracujemy? Rozwiązanie: platformy do wirtualizacji.

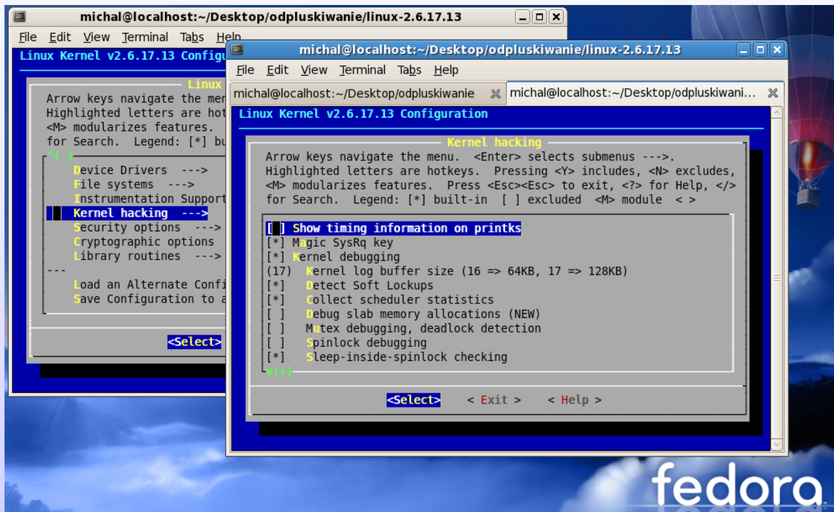
Debuggowanie jądra-problemy

1. Aby sprawdzić kawałek kodu jądra potrzebne jest jego skompilowanie.
2. Ciężko powtórzyć błąd, np. może on się ujawniać przy rzadkim przeplocie.
3. Potrzebną dokumentację ciężko znaleźć, jeśli się uda to jest niedokładna bądź niekompletna.
4. Jak debugować coś na czym aktualnie pracujemy? Rozwiązanie: platformy do wirtualizacji.

Debuggowanie jądra-problemy

1. Aby sprawdzić kawałek kodu jądra potrzebne jest jego skompilowanie.
2. Ciężko powtórzyć błąd, np. może on się ujawniać przy rzadkim przeplocie.
3. Potrzebną dokumentację ciężko znaleźć, jeśli się uda to jest niedokładna bądź niekompletna.
4. Jak debugować coś na czym aktualnie pracujemy? Rozwiązanie: platformy do wirtualizacji.

Kompilacja jądra-opcje wspomagające debugowanie



Kompilacja jądra-opcje wspomagające debugowanie

Te opcje są przydatne podczas debugowania jądra (plik .config)

- *CONFIG_PREEMPT*=y
- *CONFIG_DEBUG_KERNEL*=y
- *CONFIG_KALLSYMS*=y
- *CONFIG_SPINLOCK_SLEEP*=y
- *CONFIG_MAGIC_SYSRQ*=y

Kompilacja jądra-opcje wspomagające debugowanie

Jeżeli

- ustawiłeś *CONFIG_MAGIC_SYSRQ*=y
- lub wpiszesz `echo 1 > /proc/sys/kernel/sysrq`

to możesz używać SysRq Key → 'Alt+PrintScreen':

1. SysRq+b Restartuje komputer
2. SysRq+e Wysyła SIGTERM do wszystkich zadań (poza init!!!)
3. SysRq+h Pomoc
4. SysRq+l wysyła SIGKILL do wszystkich zadań (poza init!!!)
5. SysRq+k Zabija wszystkie zadania uruchomione z tej konsoli
6. SysRq+l wysyła SIGTKILL do wszystkich zadań (poza init!!!)
7. SysRq+m Obraz pamięci procesu zapisywany w sytuacji awaryjnej na dysku i wyświetla go w konsoli.
8. SysRq+o zatrzymuje system i wyłącza go
9. SysRq+p Wyświetla rejestry procesora w konsoli
10. SysRq+r Zmienia klawiaturę z RAW na XLATE
11. SysRq+s Zapisuje brudne bufory na twardym dysku
12. SysRq+t Pokazuje informacje o obecnym zadaniu w konsoli
13. SysRq+u Odmontowanie wszystkich plików systemowych i ponowne zamontowanie w trybie tylko do odczytu

Funkcja printk()

- printk() ⟷ printf()
- printk() używa poziomów logowania (ang. loglevels) → powiadamianie o ważności wiadomości.

1. KERN_EMERG ← najważniejsze, sytuacje awaryjne
2. KERN_ALERT ← błędy alarmujące
3. KERN_CRIT ← błędy krytyczne
4. KERN_ERR ← błędy systemu
5. KERN_WARNING ← ostrzeżenia
6. KERN_NOTICE ← notatki
7. KERN_INFO ← informacje
8. KERN_DEBUG ← najmniej ważne, standardowo nie wyświetlane

```
printk(KERN_DEBUG "This is a debug message only\n");
```

Uzycie printk()

```
printk(KERN_DEBUG "This is a debug message only\n");
```

- Zależnie od *loglevel* komunikaty printk() trafiają w odpowiednie miejsca, w tym na konsole, ale...
  1. Jeżeli w systemie działa klogd i syslogd → wszystkie w /var/log/messages
  2. Jesli wyłączymy klogd → wszystkie w /proc/kmsg (dmesg)

Poziomy printk()

- Sprawdźmy jakie mamy ustawienia printk()

```
[michal@localhost ~]$ cat /proc/sys/kernel/printk
6       4       1       7
```

1. 6=*console_loglevel*, na konsoli się wypiszą tylko komunikaty z *KERN_DEBUG* (7)
2. 4=*default_loglevel*, taki priorytet dostaną komunikaty, którym nie przydzielilismy żadnego z loglevel'i
3. najmniejsza wartość *console_loglevel*
4. 7=*default_console_loglevel*, wartość domyślna dla konsoli

- Możemy sami ustawić powyższe parametry:

```
[michal@localhost ~]$ echo "5 4 1 7" > /proc/sys/kernel/printk
```

Błąd oops

- **Oops** → raport buga zgłaszany przez jądro.
- Gdy pojawia się **oops** jądro wyświetli:
  1. opis buga
  2. licznik oops'ów
  3. aktualnie wykonywana instukcja
  4. zawartość rejestrów
  5. stos
  6. *backtrace*, czyli ślad wykonania
- Błąd **oops** nie musi oznaczać uszkodzenia systemu, a czasem system potrafi sam rozwiązać problem. Gdy system nie potrafi poradzić sobie z problemem, wyświetli komunikat kernel panic i przestanie działać.
- *Backtrace* domyślnie będzie zawierał adresy wywołanych funkcji. Jeśli wkompilowałeś w jądro opcję `CONFIG_KALLSYMS`, błąd **oops** zostanie przetłumaczony i wyświetli nazwy funkcji, które go spowodowały.

Oops-przykład z życia

```
BUG: unable to handle kernel paging request at virtual address 6b6b6c07
 printing eip:
c0138722
*pde = 00000000
Oops: 0002 [#1]
4K_STACKS PREEMPT SMP
last sysfs file: /devices/pci0000:00/0000:00:1d.7/uevent
Modules linked in: snd_timer snd soundcore snd_page_alloc intel_agp agpgart
ide_cd cdrom ipv6 w83627hf hwmon_vid hwmon i2c_isa i2c_i801 skge af_packet
ip_conntrack_netbios_ns ipt_REJECT
CPU:     0
EIP:     0060:[<c0138722>]     Not tainted VLI
EFLAGS: 00010046     (2.6.18-rc2-mm1 #78)
EIP is at __lock_acquire+0x362/0xaea
eax: 00000000    ebx: 6b6b6b6b    ecx: c0360358   edx: 00000000
esi: 00000000    edi: 00000000    ebp: f544ddf4   esp: f544ddc0
ds: 007b    es: 007b    ss: 0068
Process udevd (pid: 1353, ti=f544d000 task=f6fce8f0 task.ti=f544d000)
Stack: 00000000 00000000 00000000 c7749ea4 f6fce8f0 c0138e74 000001e8 00000000
         00000000 f6653fa4 00000246 00000000 00000000 f544de1c c0139214 00000000
         00000002 00000000 c014fe3a c7749ea4 c7749e90 f6fce8f0 f5b19b04 f544de34
Call Trace:
[<c0139214>] lock_acquire+0x71/0x91
[<c02f2bfb>] _spin_lock+0x23/0x32
[<c014fe3a>] __delayacct_blkio_ticks+0x16/0x67
[<c01a4f76>] do_task_stat+0x3df/0x6c1
[<c01a5265>] proc_tgid_stat+0xd/0xf
[<c01a29dd>] proc_info_read+0x50/0xb3
[<c0171cbb>] vfs_read+0xcb/0x177
[<c017217c>] sys_read+0x3b/0x71
[<c0103119>] sysenter_past_esp+0x56/0x8d
DWARF2 unwinder stuck at sysenter_past_esp+0x56/0x8d
Leftover inexact backtrace:
[<c0104318>] show_stack_log_lvl+0x8c/0x97
[<c010447f>] show_registers+0x15c/0x1ed
Code: 68 4b 75 2f c0 68 d5 04 00 00 68 b9 75 31 c0 68 e3 06 31 c0 e8 ce 7e fe ff
e8 87 c2 fc ff 83 c4 10 eb 08 85 db 0f 84 6b 07 00 00 <f0> ff 83 9c 00 00 00 8b
EIP: [<c0138722>] __lock_acquire+0x362/0xaea SS:ESP 0068:f544ddc0
```

Oops-przykład z życia

Opis bug'a oops'a

BUG: unable to handle kernel paging request at virtual address 6b6b6c07

```
 printing eip:
c0138722
*pde = 00000000
Oops: 0002 [#1]
4K_STACKS PREEMPT SMP
last sysfs file: /devices/pci0000:00/0000:00:1d.7/uevent
Modules linked in: snd_timer snd soundcore snd_page_alloc intel_agp agpgart
ide_cd cdrom ipv6 w83627hf hwmon_vid hwmon i2c_isa i2c_i801 skge af_packet
ip_conntrack_netbios_ns ipt_REJECT
CPU:    0
EIP:    0060:[<c0138722>]    Not tainted VLI
EFLAGS: 00010046   (2.6.18-rc2-mm1 #78)
EIP is at __lock_acquire+0x362/0xaea
eax: 00000000   ebx: 6b6b6b6b   ecx: c0360358  edx: 00000000
esi: 00000000   edi: 00000000   ebp: f544ddf4  esp: f544ddc0
ds: 007b   es: 007b   ss: 0068
Process udevd (pid: 1353, ti=f544d000 task=f6fce8f0 task.ti=f544d000)
Stack: 00000000 00000000 00000000 c7749ea4 f6fce8f0 c0138e74 000001e8 00000000
       00000000 f6653fa4 00000246 00000000 00000000 f544de1c c0139214 00000000
       00000002 00000000 c014fe3a c7749ea4 c7749e90 f6fce8f0 f5b19b04 f544de34
Call Trace:
[<c0139214>] lock_acquire+0x71/0x91
[<c02f2bfb>] _spin_lock+0x23/0x32
[<c014fe3a>] __delayacct_blkio_ticks+0x16/0x67
[<c01a4f76>] do_task_stat+0x3df/0x6c1
[<c01a5265>] proc_tgid_stat+0xd/0xf
[<c01a29dd>] proc_info_read+0x50/0xb3
[<c0171cbb>] vfs_read+0xcb/0x177
[<c017217c>] sys_read+0x3b/0x71
[<c0103119>] sysenter_past_esp+0x56/0x8d
DWARF2 unwinder stuck at sysenter_past_esp+0x56/0x8d
Leftover inexact backtrace:
[<c0104318>] show_stack_log_lvl+0x8c/0x97
[<c010447f>] show_registers+0x15c/0x1ed
Code: 68 4b 75 2f c0 68 d5 04 00 00 68 b9 75 31 c0 68 e3 06 31 c0 e8 ce 7e fe ff
e8 87 c2 fc ff 83 c4 10 eb 08 85 db 0f 84 6b 07 00 00 <f0> ff 83 9c 00 00 00 8b
EIP: [<c0138722>] __lock_acquire+0x362/0xaea SS:ESP 0068:f544ddc0
```

Oops-przykład z życia

```
BUG: unable to handle kernel paging request at virtual address 6b6b6c07
 printing eip:
c0138722
*pde = 00000000
```

Licznik oops'ow

Oops: 0002 [#1]

```
4K_STACKS PREEMPT SMP
last sysfs file: /devices/pci0000:00/0000:00:1d.7/uevent
Modules linked in: snd_timer snd soundcore snd_page_alloc intel_agp agpgart
ide_cd cdrom ipv6 w83627hf hwmon_vid hwmon i2c_isa i2c_i801 skge af_packet
ip_conntrack_netbios_ns ipt_REJECT
CPU:     0
EIP:     0060:[<c0138722>]     Not tainted VLI
EFLAGS: 00010046     (2.6.18-rc2-mm1 #78)
EIP is at __lock_acquire+0x362/0xaea
eax: 00000000    ebx: 6b6b6b6b    ecx: c0360358   edx: 00000000
esi: 00000000    edi: 00000000    ebp: f544ddf4   esp: f544ddc0
ds: 007b    es: 007b    ss: 0068
Process udevd (pid: 1353, ti=f544d000 task=f6fce8f0 task.ti=f544d000)
Stack: 00000000 00000000 00000000 c7749ea4 f6fce8f0 c0138e74 000001e8 00000000
         00000000 f6653fa4 00000246 00000000 00000000 f544de1c c0139214 00000000
         00000002 00000000 c014fe3a c7749ea4 c7749e90 f6fce8f0 f5b19b04 f544de34
Call Trace:
[<c0139214>] lock_acquire+0x71/0x91
[<c02f2bfb>] _spin_lock+0x23/0x32
[<c014fe3a>] __delayacct_blkio_ticks+0x16/0x67
[<c01a4f76>] do_task_stat+0x3df/0x6c1
[<c01a5265>] proc_tgid_stat+0xd/0xf
[<c01a29dd>] proc_info_read+0x50/0xb3
[<c0171cbb>] vfs_read+0xcb/0x177
[<c017217c>] sys_read+0x3b/0x71
[<c0103119>] sysenter_past_esp+0x56/0x8d
DWARF2 unwinder stuck at sysenter_past_esp+0x56/0x8d
Leftover inexact backtrace:
[<c0104318>] show_stack_log_lvl+0x8c/0x97
[<c010447f>] show_registers+0x15c/0x1ed
Code: 68 4b 75 2f c0 68 d5 04 00 00 68 b9 75 31 c0 68 e3 06 31 c0 e8 ce 7e fe ff
e8 87 c2 fc ff 83 c4 10 eb 08 85 db 0f 84 6b 07 00 00 <f0> ff 83 9c 00 00 00 8b
EIP: [<c0138722>] __lock_acquire+0x362/0xaea SS:ESP 0068:f544ddc0
```

Oops-przykład z życia

```
BUG: unable to handle kernel paging request at virtual address 6b6b6c07
 printing eip:
c0138722
*pde = 00000000
Oops: 0002 [\#1]
4K_STACKS PREEMPT SMP
last sysfs file: /devices/pci0000:00/0000:00:1d.7/uevent
Modules linked in: snd_timer snd soundcore snd_page_alloc intel_agp agpgart
ide_cd cdrom ipv6 w83627hf hwmon_vid hwmon i2c_isa i2c_i801 skge af_packet
ip_conntrack_netbios_ns ipt_REJECT
CPU:    0
```

Wykonywana instrukcja

```
EIP:      0060:[<c0138722>]     Not tainted VLI
```

```
EFLAGS: 00010046    (2.6.18-rc2-mm1 #78)
EIP is at __lock_acquire+0x362/0xaea
eax: 00000000   ebx: 6b6b6b6b   ecx: c0360358  edx: 00000000
esi: 00000000   edi: 00000000   ebp: f544ddf4  esp: f544ddc0
ds: 007b   es: 007b   ss: 0068
Process udevd (pid: 1353, ti=f544d000 task=f6fce8f0 task.ti=f544d000)
Stack: 00000000 00000000 00000000 c7749ea4 f6fce8f0 c0138e74 000001e8 00000000
         00000000 f6653fa4 00000246 00000000 00000000 f544de1c c0139214 00000000
         00000002 00000000 c014fe3a c7749ea4 c7749e90 f6fce8f0 f5b19b04 f544de34
Call Trace:
[<c0139214>] lock_acquire+0x71/0x91
[<c02f2bfb>] _spin_lock+0x23/0x32
[<c014fe3a>] __delayacct_blkio_ticks+0x16/0x67
[<c01a4f76>] do_task_stat+0x3df/0x6c1
[<c01a5265>] proc_tgid_stat+0xd/0xf
[<c01a29dd>] proc_info_read+0x50/0xb3
[<c0171cbb>] vfs_read+0xcb/0x177
[<c017217c>] sys_read+0x3b/0x71
[<c0103119>] sysenter_past_esp+0x56/0x8d
DWARF2 unwinder stuck at sysenter_past_esp+0x56/0x8d
Leftover inexact backtrace:
[<c0104318>] show_stack_log_lvl+0x8c/0x97
[<c010447f>] show_registers+0x15c/0x1ed
Code: 68 4b 75 2f c0 68 d5 04 00 00 68 b9 75 31 c0 68 e3 06 31 c0 e8 ce 7e fe ff
e8 87 c2 fc ff 83 c4 10 eb 08 85 db 0f 84 6b 07 00 00 <f0> ff 83 9c 00 00 00 8b
EIP: [<c0138722>] __lock_acquire+0x362/0xaea SS:ESP 0068:f544ddc0
```

Oops-przykład z życia

```
BUG: unable to handle kernel paging request at virtual address 6b6b6c07
 printing eip:
c0138722
*pde = 00000000
Oops: 0002 [\#1]
4K_STACKS PREEMPT SMP
last sysfs file: /devices/pci0000:00/0000:00:1d.7/uevent
Modules linked in: snd_timer snd soundcore snd_page_alloc intel_agp agpgart
ide_cd cdrom ipv6 w83627hf hwmon_vid hwmon i2c_isa i2c_i801 skge af_packet
ip_conntrack_netbios_ns ipt_REJECT
CPU:     0
EIP:     0060:[<c0138722>]     Not tainted VLI
```

Rejestry i flagi (2ga linijka pomaga zlokalizowac wiersz w kodzie źródłowym)

```
EFLAGS: 00010046     (2.6.18-rc2-mm1 #78)
EIP is at __lock_acquire+0x362/0xaea
eax: 00000000    ebx: 6b6b6b6b    ecx: c0360358  edx: 00000000
esi: 00000000    edi: 00000000    ebp: f544ddf4  esp: f544ddc0
ds: 007b    es: 007b    ss: 0068
```

```
Process udevd (pid: 1353, ti=f544d000 task=f6fce8f0 task.ti=f544d000)
Stack: 00000000 00000000 00000000 c7749ea4 f6fce8f0 c0138e74 000001e8 00000000
       00000000 f6653fa4 00000246 00000000 00000000 f544de1c c0139214 00000000
       00000002 00000000 c014fe3a c7749ea4 c7749e90 f6fce8f0 f5b19b04 f544de34
Call Trace:
[<c0139214>] lock_acquire+0x71/0x91
[<c02f2bfb>] _spin_lock+0x23/0x32
[<c014fe3a>] __delayacct_blkio_ticks+0x16/0x67
[<c01a4f76>] do_task_stat+0x3df/0x6c1
[<c01a5265>] proc_tgid_stat+0xd/0xf
[<c01a29dd>] proc_info_read+0x50/0xb3
[<c0171cbb>] vfs_read+0xcb/0x177
[<c017217c>] sys_read+0x3b/0x71
[<c0103119>] sysenter_past_esp+0x56/0x8d
DWARF2 unwinder stuck at sysenter_past_esp+0x56/0x8d
Leftover inexact backtrace:
[<c0104318>] show_stack_log_lvl+0x8c/0x97
[<c010447f>] show_registers+0x15c/0x1ed
Code: 68 4b 75 2f c0 68 d5 04 00 00 68 b9 75 31 c0 68 e3 06 31 c0 e8 ce 7e fe ff
e8 87 c2 fc ff 83 c4 10 eb 08 85 db 0f 84 6b 07 00 00 <f0> ff 83 9c 00 00 00 8b
EIP: [<c0138722>] __lock_acquire+0x362/0xaea SS:ESP 0068:f544ddc0
```

Oops-przykład z życia

```
BUG: unable to handle kernel paging request at virtual address 6b6b6c07
 printing eip:
c0138722
*pde = 00000000
Oops: 0002 [\#1]
4K_STACKS PREEMPT SMP
last sysfs file: /devices/pci0000:00/0000:00:1d.7/uevent
Modules linked in: snd_timer snd soundcore snd_page_alloc intel_agp agpgart
ide_cd cdrom ipv6 w83627hf hwmon_vid hwmon i2c_isa i2c_i801 skge af_packet
ip_conntrack_netbios_ns ipt_REJECT
CPU:     0
EIP:     0060:[<c0138722>]     Not tainted VLI
EFLAGS: 00010046     (2.6.18-rc2-mm1 #78)
EIP is at __lock_acquire+0x362/0xaea
eax: 00000000    ebx: 6b6b6b6b    ecx: c0360358  edx: 00000000
esi: 00000000    edi: 00000000    ebp: f544ddf4  esp: f544ddc0
ds: 007b    es: 007b    ss: 0068
Process udevd (pid: 1353, ti=f544d000 task=f6fce8f0 task.ti=f544d000)
```

Stos

Stack: 00000000 00000000 00000000 c7749ea4 f6fce8f0 c0138e74 000001e8 00000000 00000000 f6653fa4 00000246 00000000 00000000 f544de1c c0139214 00000000 00000002 00000000 c014fe3a c7749ea4 c7749e90 f6fce8f0 f5b19b04 f544de34

```
Call Trace:
[<c0139214>] lock_acquire+0x71/0x91
[<c02f2bfb>] _spin_lock+0x23/0x32
[<c014fe3a>] __delayacct_blkio_ticks+0x16/0x67
[<c01a4f76>] do_task_stat+0x3df/0x6c1
[<c01a5265>] proc_tgid_stat+0xd/0xf
[<c01a29dd>] proc_info_read+0x50/0xb3
[<c0171cbb>] vfs_read+0xcb/0x177
[<c017217c>] sys_read+0x3b/0x71
[<c0103119>] sysenter_past_esp+0x56/0x8d
DWARF2 unwinder stuck at sysenter_past_esp+0x56/0x8d
Leftover inexact backtrace:
[<c0104318>] show_stack_log_lvl+0x8c/0x97
[<c010447f>] show_registers+0x15c/0x1ed
Code: 68 4b 75 2f c0 68 d5 04 00 00 68 b9 75 31 c0 68 e3 06 31 c0 e8 ce 7e fe ff
e8 87 c2 fc ff 83 c4 10 eb 08 85 db 0f 84 6b 07 00 00 <f0> ff 83 9c 00 00 00 8b
EIP: [<c0138722>] __lock_acquire+0x362/0xaea SS:ESP 0068:f544ddc0
```

Oops-przykład z życia

```
BUG: unable to handle kernel paging request at virtual address 6b6b6c07
 printing eip:
c0138722
*pde = 00000000
Oops: 0002 [\#1]
4K_STACKS PREEMPT SMP
last sysfs file: /devices/pci0000:00/0000:00:1d.7/uevent
Modules linked in: snd_timer snd soundcore snd_page_alloc intel_agp agpgart
ide_cd cdrom ipv6 w83627hf hwmon_vid hwmon i2c_isa i2c_i801 skge af_packet
ip_conntrack_netbios_ns ipt_REJECT
CPU:     0
EIP:     0060:[<c0138722>]     Not tainted VLI
EFLAGS: 00010046     (2.6.18-rc2-mm1 #78)
EIP is at __lock_acquire+0x362/0xaea
eax: 00000000    ebx: 6b6b6b6b    ecx: c0360358   edx: 00000000
esi: 00000000    edi: 00000000    ebp: f544ddf4   esp: f544ddc0
ds: 007b    es: 007b    ss: 0068
Process udevd (pid: 1353, ti=f544d000 task=f6fce8f0 task.ti=f544d000)
Stack: 00000000 00000000 00000000 c7749ea4 f6fce8f0 c0138e74 000001e8 00000000
         00000000 f6653fa4 00000246 00000000 00000000 f544de1c c0139214 00000000
         00000002 00000000 c014fe3a c7749ea4 c7749e90 f6fce8f0 f5b19b04 f544de34
```

Ślad wykonania

```
Call Trace:
[<c0139214>] lock_acquire+0x71/0x91
[<c02f2bfb>] _spin_lock+0x23/0x32
[<c014fe3a>] __delayacct_blkio_ticks+0x16/0x67
[<c01a4f76>] do_task_stat+0x3df/0x6c1
[<c01a5265>] proc_tgid_stat+0xd/0xf
[<c01a29dd>] proc_info_read+0x50/0xb3
[<c0171cbb>] vfs_read+0xcb/0x177
[<c017217c>] sys_read+0x3b/0x71
[<c0103119>] sysenter_past_esp+0x56/0x8d
DWARF2 unwinder stuck at sysenter_past_esp+0x56/0x8d
Leftover inexact backtrace:
[<c0104318>] show_stack_log_lvl+0x8c/0x97
[<c010447f>] show_registers+0x15c/0x1ed
```

```
Code: 68 4b 75 2f c0 68 d5 04 00 00 68 b9 75 31 c0 68 e3 06 31 c0 e8 ce 7e fe ff
e8 87 c2 fc ff 83 c4 10 eb 08 85 db 0f 84 6b 07 00 00 <f0> ff 83 9c 00 00 00 8b
EIP: [<c0138722>] __lock_acquire+0x362/0xaea SS:ESP 0068:f544ddc0
```

Oops-przykład z życia

```
[michal@localhost ~]$ ksymoops
```

Uwaga

ksymoops jest bezużyteczny na jądrach 2.6

Ksymoops-pomaga analizowac oops'y INSTALACJA ——>

```
mkdir ~/ksymoops
cd ~/ksymoops

wget http://www.kernel.org/pub/linux/utils/kernel/ksymoops/v2.4/ksymoops-2.4.9.tar.gz

tar xzf ksymoops-2.4.9.tar.gz
cd ksymoops-2.4.9
make
make install # jako root
```

ksymoops korzystając z :

1. *System.map*
2. */proc/ksyms*
3. */lib/modules*

tłumaczy adresy z oops na symbole.

Demony klogd i syslogd

Definicja

Demony, inaczej usługi, są pewnym rodzajem programów, które są uruchamiane zazwyczaj przy starcie systemu, i działają potem cały czas w tle, zużywając minimalne zasoby.

Źr. Wikipedia

Demony klogd i syslogd

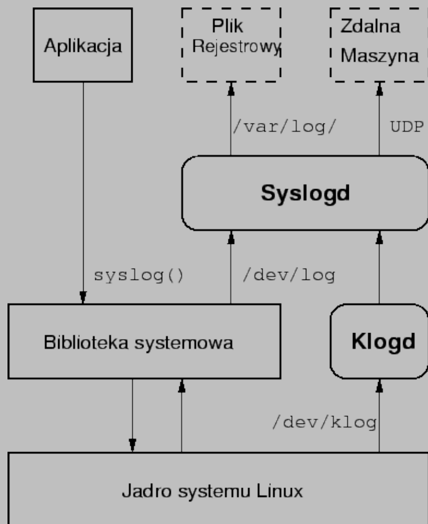
syslogd

- syslogd - przyjmuje informacje zgłaszane przez bibliotekę systemową.
- Aplikacje, najczęściej różne demony, przekazują informacje poprzez standardową funkcję syslog(3). Są one następnie przekazywane syslogd, który w zależności od ich typu i priorytetu, zapisuje je w odpowiednich plikach (plikach rejestrowych (ang. log files)).
- Każdy wpis do pliku rejestrowego dokonywany przez syslogd zawiera informacje dotyczące czasu jego powstania, maszyny i procesu.

klogd

- klogd - zbiera raporty od jądra systemu Linux
- Ponieważ mają one inną postać niż te, które przekazuje biblioteka systemowa, zostają poddane obróbce przed przesłaniem ich do syslogd. Ta obróbka polega na opatrzeniu ich informacjami o typie i priorytecie wiadomości zgodnymi z syslogd.

Schemat działania klogd i syslogd



źródło:http://home.agh.edu.pl/~gjn/dydaktyka/papers/GJN-Syslog/GJN-Syslog.html#SECTION00030000000000000000

strace i ltrace

Strace to narzędzie do analizy kodu badające interakcję programu z jądrem systemu operacyjnego.

```
[michal@localhost ~]$ strace /bin/true
```

```
execve("/bin/true", ["/bin/true"], [/* 41 vars */]) = 0
brk(0)                                  = 0x89e8000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=71493, ...}) = 0
mmap2(NULL, 71493, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7f54000
close(3)                                = 0
open("/lib/libc.so.6", O_RDONLY)        = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\320 \265\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1673804, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7f53000
mmap2(0xb3c000, 1390032, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb3c000
mmap2(0xc8a000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x14e) = 0xc8a00
mmap2(0xc8d000, 9680, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xc8d000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7f52000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb7f526c0, limit:1048575, seg_32bit:1, contents:0,
_in_pages:1, seg_not_present:0, useable:1}) = 0
mprotect(0xc8a000, 8192, PROT_READ)     = 0
mprotect(0x186000, 4096, PROT_READ)     = 0
munmap(0xb7f54000, 71493)               = 0
brk(0)                                  = 0x89e8000
brk(0x8a09000)                          = 0x8a09000
open("/usr/lib/locale/locale-archive", O_RDONLY|O_LARGEFILE) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=65179696, ...}) = 0
mmap2(NULL, 2097152, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7d52000
close(3)                                = 0
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
```

strace i ltrace

Ltrace to narzędzie analizy kodu badające interakcję programu z używanymi przez niego bibliotekami dzielonymi (shared libraries).

```
[michal@localhost ~]$ ltrace /bin/true
```

```
 __libc_start_main(0x8048c60, 1, 0xbfaadaf4, 0x804a340, 0x804a330 <unfinished ...>
setlocale(6, "")                                                       = "en_US.UTF-8"
bindtextdomain("coreutils", "/usr/share/locale")                       = "/usr/share/locale"
textdomain("coreutils")                                                = "coreutils"
__cxa_atexit(0x8049070, 0, 0, 0x804c368, 0xbfaada58)                   = 0
exit(0 <unfinished ...>
__fpending(0xc8c4c0, 0xc8d120, 0x804a66e, 0x804c368, 0x804a66e)        = 0
fclose(0xc8c4c0)                                                       = 0
__fpending(0xc8c560, 0xc8d120, 0, 0x804c368, 0x804a66e)                = 0
fclose(0xc8c560)                                                       = 0
+++ exited (status 0) +++
```

Strace i Ltrace

UWAGA!

Program, który będzie śledzony przez strace/ltrace nie musi być skompilowany z informacjami dla debuggera (-g)

GDB

GNU Debugger

Jest to debugger będący częścią projektu GNU, napisany w 1988 r. przez Richarda Stallmana.

Nazwa

Zazwyczaj zamiast pełnej nazwy używa się akronimu GDB.

Licencja

GNU Debugger jest dostępny na warunkach Powszechnej Licencji Publicznej GNU.

GDB

GNU Debugger

Jest to debugger będący częścią projektu GNU, napisany w 1988 r. przez Richarda Stallmana.

Nazwa

Zazwyczaj zamiast pełnej nazwy używa się akronimu GDB.

Licencja

GNU Debugger jest dostępny na warunkach Powszechnej Licencji Publicznej GNU.

GDB

GNU Debugger

Jest to debugger będący częścią projektu GNU, napisany w 1988 r. przez Richarda Stallmana.

Nazwa

Zazwyczaj zamiast pełnej nazwy używa się akronimu GDB.

Licencja

GNU Debugger jest dostępny na warunkach Powszechnej Licencji Publicznej GNU.

GDB

GNU Debugger

Jest to debugger będący częścią projektu GNU, napisany w 1988 r. przez Richarda Stallmana.

Nazwa

Zazwyczaj zamiast pełnej nazwy używa się akronimu GDB.

Licencja

GNU Debugger jest dostępny na warunkach Powszechnej Licencji Publicznej GNU.

Działanie GDB

Interfejs

GNU Debugger działa w trybie tekstowym, lecz zdecydowana większość zintegrowanych srodowisk programistycznych posiadających interfejs graficzny potrafi prezentować wyniki działania GDB.

Działanie GDB

Interfejs

GNU Debugger działa w trybie tekstowym, lecz zdecydowana większość zintegrowanych srodowisk programistycznych posiadających interfejs graficzny potrafi prezentować wyniki działania GDB.

Funkcjonalność

Funkcjonalność

GDB oferuje możliwość dokładnego śledzenia wykonania programów komputerowych, oraz wprowadzania zmian w czasie tego wykonania.

Możliwości

Użytkownik może:

- podejrzeć wartość zmiennych
- zmienić ich wartość
- wywołać funkcje w programie (niezależnie od jego normalnego działania)

Funkcjonalność

Funkcjonalność

GDB oferuje możliwość dokładnego śledzenia wykonania programów komputerowych, oraz wprowadzania zmian w czasie tego wykonania.

Możliwości

Użytkownik może:

- podejrzeć wartość zmiennych
- zmienić ich wartość
- wywołać funkcje w programie (niezależnie od jego normalnego działania)

Funkcjonalność

Funkcjonalność

GDB oferuje możliwość dokładnego śledzenia wykonania programów komputerowych, oraz wprowadzania zmian w czasie tego wykonania.

Możliwości

Użytkownik może:

- podejrzeć wartość zmiennych
- zmienić ich wartość
- wywołać funkcje w programie (niezależnie od jego normalnego działania)

Debugowanie zdalne

Remote debugging

GDB może być użyte do debugowania programu uruchomionego na innym komputerze. Wtedy GDB jest uruchomione na jednym komputerze, debugowany program na drugim, a komunikacja odbywa się za pomocą portów szeregowych lub TCP/IP.

KGDB

Ten sam tryb jest właśnie wykorzystywany przez KGDB.

Debugowanie zdalne

Remote debugging

GDB może być użyte do debugowania programu uruchomionego na innym komputerze. Wtedy GDB jest uruchomione na jednym komputerze, debugowany program na drugim, a komunikacja odbywa się za pomocą portów szeregowych lub TCP/IP.

KGDB

Ten sam tryb jest właśnie wykorzystywany przez KGDB.

Debugowanie zdalne

Remote debugging

GDB może być użyte do debugowania programu uruchomionego na innym komputerze. Wtedy GDB jest uruchomione na jednym komputerze, debugowany program na drugim, a komunikacja odbywa się za pomocą portów szeregowych lub TCP/IP.

KGDB

Ten sam tryb jest właśnie wykorzystywany przez KGDB.

Przygotowanie do pracy

Symbole

Pliki kompilowane normalnie (co jest typowe dla gotowych aplikacji) są pozbawione symboli. W takim przypadku nie można dowiedzieć się za wiele poza adresem instrukcji, która spowodowała niepoprawne zachowanie programu.

Kompilacja

Oczywiście istnieje rozwiązanie tego problemu — programy można kompilować z opcją -g, która doda nam informacje potrzebne do debugowania. Dzięki temu będziemy mieli dostęp do m.in. typów danych poszczególnych zmiennych, nazw funkcji, a nie tylko adresów w pamięci.

Przygotowanie do pracy

Symbole

Pliki kompilowane normalnie (co jest typowe dla gotowych aplikacji) są pozbawione symboli. W takim przypadku nie można dowiedzieć się za wiele poza adresem instrukcji, która spowodowała niepoprawne zachowanie programu.

Kompilacja

Oczywiście istnieje rozwiązanie tego problemu — programy można kompilować z opcją -g, która doda nam informacje potrzebne do debugowania. Dzięki temu będziemy mieli dostęp do m.in. typów danych poszczególnych zmiennych, nazw funkcji, a nie tylko adresów w pamięci.

Przygotowanie do pracy

Symbole

Pliki kompilowane normalnie (co jest typowe dla gotowych aplikacji) są pozbawione symboli. W takim przypadku nie można dowiedzieć się za wiele poza adresem instrukcji, która spowodowała niepoprawne zachowanie programu.

Kompilacja

Oczywiście istnieje rozwiązanie tego problemu — programy można kompilować z opcją -g, która doda nam informacje potrzebne do debugowania. Dzięki temu będziemy mieli dostęp do m.in. typów danych poszczególnych zmiennych, nazw funkcji, a nie tylko adresów w pamięci.

Synopsis

Synopsis

- *gdb program* — najczęściej używana wersja
- *gdb program core* — tutaj specyfikujemy plik zrzutu pamięci

Synopsis

Synopsis

- *gdb program* — najczęściej używana wersja
- *gdb program core* — tutaj specyfikujemy plik zrzutu pamięci

Synopsis

Argumenty

Do przekazywania parametrów do programu można użyć opcji args, np. *gdb –args gcc -O2 -c foo.c*
Ta opcja wyłącza przetwarzanie innych opcji.

Synopsis

Argumenty

Do przekazywania parametrów do programu można użyć opcji args, np. *gdb –args gcc -O2 -c foo.c*
Ta opcja wyłącza przetwarzanie innych opcji.

Kontrola wykonania programu

Breakpoint

Breakpoint — powoduje zatrzymuje wykonywania programu, wtedy gdy dojdzie się do pewnego punktu w programie. Breakpointy mogą być uszczegółowione przez warunki, gdy program ma się zatrzymać

Watchpoint

Watchpoint — specjalny breakpoint, który zatrzymuje program gdy wartość pewnego wyrażenia (np. zmiennej) zmienia się.

Catchpoint

Catchpoint — inny rodzaj breakpointa, który zatrzymuje program gdy pewne zachodzi pewne wydarzenie, np. wyjątek w C++ lub załadowanie biblioteki.

Kontrola wykonania programu

Breakpoint

Breakpoint — powoduje zatrzymuje wykonywania programu, wtedy gdy dojdzie się do pewnego punktu w programie. Breakpointy mogą być uszczegółowione przez warunki, gdy program ma się zatrzymać

Watchpoint

Watchpoint — specjalny breakpoint, który zatrzymuje program gdy wartość pewnego wyrażenia (np. zmiennej) zmienia się.

Catchpoint

Catchpoint — inny rodzaj breakpointa, który zatrzymuje program gdy pewne zachodzi pewne wydarzenie, np. wyjątek w C++ lub załadowanie biblioteki.

Kontrola wykonania programu

Breakpoint

Breakpoint — powoduje zatrzymuje wykonywania programu, wtedy gdy dojdzie się do pewnego punktu w programie. Breakpointy mogą być uszczegółowione przez warunki, gdy program ma się zatrzymać

Watchpoint

Watchpoint — specjalny breakpoint, który zatrzymuje program gdy wartość pewnego wyrażenia (np. zmiennej) zmienia się.

Catchpoint

Catchpoint — inny rodzaj breakpointa, który zatrzymuje program gdy pewne zachodzi pewne wydarzenie, np. wyjątek w C++ lub załadowanie biblioteki.

Kontrola wykonania programu

Breakpoint

Breakpoint — powoduje zatrzymuje wykonywania programu, wtedy gdy dojdzie się do pewnego punktu w programie. Breakpointy mogą być uszczegółowione przez warunki, gdy program ma się zatrzymać

Watchpoint

Watchpoint — specjalny breakpoint, który zatrzymuje program gdy wartość pewnego wyrażenia (np. zmiennej) zmienia się.

Catchpoint

Catchpoint — inny rodzaj breakpointa, który zatrzymuje program gdy pewne zachodzi pewne wydarzenie, np. wyjątek w C++ lub załadowanie biblioteki.

Operacje na breakpointach

- *break* function
- *break* lineNum
- *break* ... if cond
- *break* — breakpoint na następnej instrukcji, przydatne gdy będziemy skakać po programie
- *info break* — wyrzuca informację o breakpoincie, m.in. liczbę trafień (użyteczne przy *ignore*)
- *condition bnum expr* — dodaje warunek, zatrzymanie tylko gdy wyrażenie jest prawdziwe (niezerowe)
- *condition bnum* — usuwa warunek
- *ignore bnum count*
- *commands bnum ... cmdlist ... end* — lista instrukcji do wykonania przy breakpoincie

Operacje na breakpointach

- *break* function
- *break* lineNum
- *break* ... if cond
- *break* — breakpoint na następnej instrukcji, przydatne gdy będziemy skakać po programie
- *info break* — wyrzuca informację o breakpoincie, m.in. liczbę trafień (użyteczne przy *ignore*)
- *condition bnum expr* — dodaje warunek, zatrzymanie tylko gdy wyrażenie jest prawdziwe (niezerowe)
- *condition bnum* — usuwa warunek
- *ignore bnum count*
- *commands bnum ... cmdlist ... end* — lista instrukcji do wykonania przy breakpoincie

Kontrola wykonania programu

Continue

continue [ignore count] — dalsze wykonywanie programu, aż do kolejnego breakpointa

Step

step — przejście do kolejnej instrukcji

Next

next — jak *step*, ale wywołania funkcji nie są uszczegóławiane

Kontrola wykonania programu

Continue

continue [ignore count] — dalsze wykonywanie programu, aż do kolejnego breakpointa

Step

step — przejście do kolejnej instrukcji

Next

next — jak *step*, ale wywołania funkcji nie są uszczegóławiane

Kontrola wykonania programu

Continue

continue [ignore count] — dalsze wykonywanie programu, aż do kolejnego breakpointa

Step

step — przejście do kolejnej instrukcji

Next

next — jak *step*, ale wywołania funkcji nie są uszczegóławiane

Kontrola wykonania programu

Continue

continue [ignore count] — dalsze wykonywanie programu, aż do kolejnego breakpointa

Step

step — przejście do kolejnej instrukcji

Next

next — jak *step*, ale wywołania funkcji nie są uszczegóławiane

Kontrola wykonania programu

Sygnały

- *info handle*
- *handle signal [keywords]* — definiuje zachowanie gdb przy otrzymaniu sygnalu

Kontrola wykonania programu

Sygnały

- *info handle*
- *handle signal [keywords]* — definiuje zachowanie gdb przy otrzymaniu sygnalu

Kontrola wykonania programu

Stos wywołań

- *frame num* — wybiera ramkę *num*, gdzie 0 to obecna ramka
- *info frame*
- *backtrace*

Kontrola wykonania programu

Stos wywołań

- *frame num* — wybiera ramkę *num*, gdzie 0 to obecna ramka
- *info frame*
- *backtrace*

Modyfikacje kodu

Kod

- *list* — wyświetla linie kodu
- *edit*
- *edit number*
- *edit function*

Modyfikacje kodu

Kod

- *list* — wyświetla linie kodu
- *edit*
- *edit number*
- *edit function*

Modyfikacje danych

Dane

- *print expr*
- *x addr* — podgląd pamięci (examine)
- *display expr* — wyświetla wartość wyrażenia przy każdym zatrzymaniu
- *info registers* — wypisuje zawartość rejestrów
- *info variables* — wypisuje listę zmiennych w programie

Modyfikacje danych

Dane

- *print expr*
- *x addr* — podgląd pamięci (examine)
- *display expr* — wyświetla wartość wyrażenia przy każdym zatrzymaniu
- *info registers* — wypisuje zawartość rejestrów
- *info variables* — wypisuje listę zmiennych w programie

Modyfikacje danych

Dane

- *set [var] variable=value* — przypisanie na zmienną
- *signal signal*
- *return*
- *return expr*
- *call expr*
- *info variables* — wypisuje listę zmiennych w programie

Modyfikacje danych

Dane

- *set [var] variable=value* — przypisanie na zmienną
- *signal signal*
- *return*
- *return expr*
- *call expr*
- *info variables* — wypisuje listę zmiennych w programie

Inne instrukcje

Inne instrukcje

- *set args a1 a2 ... aN* — ustawia argumenty dla programu
- *show args* — pokazuje nam ustawione argumenty
- *set env vr [=val]* — ustawia nam zmienną lokalną vr na val

Procesy potomne

fork()

Jeśli chcemy debugować proces potomny, powstały przy użyciu *fork()*, najlepszym sposobem jest dodanie instrukcji *sleep()* na początku kodu potomka. Wtedy można uzyskać jego pid (przez *ps*), i uruchomić *gdb -p pid*.

Pliki core

Synopsis

Uruchomienie programu przez *gdb executable coreFile*, a następnie użycie *where* pozwala nam przenieść się do miejsca, gdzie program został zakończony.

KGDB

KGDB

KGDB jest debuggerem dla jądra linuksowego. Użytkownicy tego programu mogą odpluskwiać jądro podobnie jak zwyczajne aplikacje przy użyciu GDB.

Wymagania

Aby KGDB zaczęło działać, potrzebne są dwa komputery. Są one połączone szeregowo, na jednej z nich jest uruchomione jądro, a na drugiej GDB. Dane przesyłane są wspomnianym połączeniem szeregowym.

KGDB

KGDB

KGDB jest debuggerem dla jądra linuksowego. Użytkownicy tego programu mogą odpluskwiać jądro podobnie jak zwyczajne aplikacje przy użyciu GDB.

Wymagania

Aby KGDB zaczęło działać, potrzebne są dwa komputery. Są one połączone szeregowo, na jednej z nich jest uruchomione jądro, a na drugiej GDB. Dane przesyłane są wspomnianym połączeniem szeregowym.

KGDB

KGDB

KGDB jest debuggerem dla jądra linuksowego. Użytkownicy tego programu mogą odpluskwiać jądro podobnie jak zwyczajne aplikacje przy użyciu GDB.

Wymagania

Aby KGDB zaczęło działać, potrzebne są dwa komputery. Są one połączone szeregowo, na jednej z nich jest uruchomione jądro, a na drugiej GDB. Dane przesyłane są wspomnianym połączeniem szeregowym.

Instalacja

Wymagania

KGDB przyjmuje postać patcha na jądro. Jądro zostaje wzbogacone o następujące funkcje:

- namiastka GDB — serce debuggera, które obsługuje żądania przychodzące od GDB uruchomionego na drugim komputerze. Ma kontrolę nad wszystkimi procesorami na maszynie podczas, gdy jest uruchomiony debugger
- modyfikacje obsługi błędów — jądro oddaje kontrolę do debuggera w przypadku niespodziewanych błędów (unexpected fault). W normalnym przypadku doszłoby do kernel panic.
- komunikacja — komponent ten używa sterownika portu szeregowego w jądrze i oferuje interfejs do w/w namiastki gdb. Jest odpowiedzialny za przesyłanie danych oraz obsługę przerwania kontroli

Instalacja

Wymagania

KGDB przyjmuje postać patcha na jądro. Jądro zostaje wzbogacone o następujące funkcje:

- namiastka GDB — serce debuggera, które obsługuje żądania przychodzące od GDB uruchomionego na drugim komputerze. Ma kontrolę nad wszystkimi procesorami na maszynie podczas, gdy jest uruchomiony debugger
- modyfikacje obsługi błędów — jądro oddaje kontrolę do debuggera w przypadku niespodziewanych błędów (unexpected fault). W normalnym przypadku doszłoby do kernel panic.
- komunikacja — komponent ten używa sterownika portu szeregowego w jądrze i oferuje interfejs do w/w namiastki gdb. Jest odpowiedzialny za przesyłanie danych oraz obsługę przerwania kontroli

Zalety i wady

Zalety

- debugowanie jądra wygląda tak samo jak debugowanie normalnego programu w GDB
- możliwe debugowanie modułów
- praca zdalna — nie ryzykujemy uszkodzeniem naszego komputera

Wady

- potrzebne są dwa komputery
- nie ma patchy dla najnowszych wersji jądra

Zalety i wady

Zalety

- debugowanie jądra wygląda tak samo jak debugowanie normalnego programu w GDB
- możliwe debugowanie modułów
- praca zdalna — nie ryzykujemy uszkodzeniem naszego komputera

Wady

- potrzebne są dwa komputery
- nie ma patchy dla najnowszych wersji jądra

Zalety i wady

Zalety

- debugowanie jądra wygląda tak samo jak debugowanie normalnego programu w GDB
- możliwe debugowanie modułów
- praca zdalna — nie ryzykujemy uszkodzeniem naszego komputera

Wady

- potrzebne są dwa komputery
- nie ma patchy dla najnowszych wersji jądra

Uruchomienie

1. `patch -p1 < patchfile`
2. `make menuconfig`
3. wybieramy *Remote (serial) debugging with gdb*
4. `make clean`
5. `make bzImage`
6. kopiujemy nowopowstały obraz jądra na maszynę testującą

Modyfikujemy lilo.conf

```
image= /boot/vmlinuz-target
     label=target_kernel
     read-only
     root=/dev/hda1
```

Uruchomienie cd.

1. kopiujemy */usr/src/linux/arch/i386/kernel/gdbstart* na maszynę testową
2. wybieramy port (ttyS0 lub ttyS1) i transfer
3. łączymy komputery
4. `gdbstart  - s 38400 - t /dev/ttyS0`
5. `cd /usr/src/linux`
6. `gdb vmlinux`

Połączenie

```
[root@development /root]# cd /usr/src/linux
[root@development linux]# gdb vmlinux
GNU gdb 19991004
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the
GNU General Public License, and you are
welcome to change it and/or distribute copies
of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.
Type "show warranty" for details.
This GDB was configured as
"i386-redhat-linux"...
(gdb) rmt
breakpoint () at gdbstub.c:1240
1240    }
(gdb)
```

Step

```
[root@development linux]# gdb vmlinux
GNU gdb 19991004
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the
GNU General Public License, and you are
welcome to change it and/or distribute copies
of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.
Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) rmt
breakpoint () at gdbstub.c:1240
1240    }
(gdb) s
gdb_hook () at gdbserial.c:219
219             gdb_null() ;
(gdb)
```

Moduł simple.c

1. `gcc -c -O2 -g simple.c`
2. kopiujemy *simple.o* na maszynę testową
3. `insmod -m simple.o`

Struktura naszego modułu

```
Sections:         Size      Address   Align
.this             0000004c  c4800000  2**2
.text             0000002c  c480004c  2**2
.rodata           000000ec  c4800080  2**5
.data             00000000  c480016c  2**2
.kstrtab          0000006c  c480016c  2**0
.bss              00000000  c48001d8  2**2
__ksymtab         00000018  c48001d8  2**2

Symbols:
00000000 a simple.c
c4800000 D __insmod_simple_Osimple.o_M3A24DFF7_V132096
c4800000 d __this_module
c480004c T __insmod_simple_S.text_L44
c480004c t .text
c480004c t init_module
c4800064 t cleanup_module
c4800080 r .rodata
c4800080 R __insmod_simple_S.rodata_L236
c480016c d .data
c48001d8 d .bss
```

Moduł simple.c

1. zapisujemy początek kodu (*.text*)
2. modyfikujemy kod modułu, np. dodając wysłanie *SIGTRAP* przy ładowaniu i odładowaniu
3. kompilujemy i kopiujemy *simple.o*
4. `gdbstart`
5. `gdb vmlinux`
6. `add-symbol-file /root/simple.o 0xc480004c`
7. pozwólmy na dalsze działanie jądra przez *continue*
8. `insmod simple.o`

Uruchomiony moduł

```
...
(gdb) add-symbol-file /root/simple.o 0xc480004c
add symbol table from file "/root/simple.o" at
        .text_addr = 0xc480004c
(y or n) y
Reading symbols from /root/simple.o...done.
(gdb) c
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
init_module () at simple.c:34
34      printk(KERN_ALERT
 "\nThe kernel module %s has been loaded.\n", MOD);
(gdb) s
printk (fmt=0xc4800100 "<1>\nThe kernel module %s has
    been loaded.\n");
    at printk.c:263
263             spin_lock_irqsave(&console_lock, flags);
(gdb) s
264 va_start(args, fmt);
```

etc...

Odładowanie

1. pozwólmy na dalsze działanie jądra przez *continue*
2. `rmmod simple`

Odładowany moduł

```
(gdb) c
Continuing.
Program received signal SIGTRAP, Trace/breakpoint trap.
0xc4800068 in cleanup_module () at simple.c:38
38      } // end function init_module
(gdb)
```

Co to jest User-mode Linux?

`user-mode-linux.sf.net`

- Program umożliwiający uruchomienie Linuksa pod Linuksem bez obaw o uszkodzenie macierzystego systemu operacyjnego
- Rozwijany od 1999 roku, rozprowadzany w formie łatki na jądro

Cechy UML

- działa jako zwykły proces użytkownika
- nie operuje bezpośrednio na sprzęcie jak zwykłe jądro
- można uruchamiać wiele komputerów wirtualnych na jednym hoście fizycznym
- nie trzeba mieć uprawnień administratora, aby dowolnie modyfikować jądro

Zalety UML

- szybkość - straty wydajnościowe sięgają co najwyżej 30%
- elastyczność - UML może pełnić funkcję biblioteki dzielonej przez inne aplikacje
- dostępność - licencja GPL jako część jądra Linuksa
- bezpieczeństwo - odizolowanie przestrzeni procesów UML od systemu hosta
- konfigurowalność - *./linux ubd0=rootfs.debian ubd1=swapfs.debian con0=fd:0,fd:1 eth0=tuntap,F5:FD,78:AD:FG:CC uml-debby mem=32M*
- oszczędność - UML zawiera cechę współdzielenia obrazu głównego systemu plików

Zastosowania UML

środowisko do testowania

dzięki UML-owi można uniknąć przeładowywania systemu, gdy wymagają tego przeprowadzane testy. Bardzo przydatna cecha dla administratorów

testowanie nowych wersji jądra/dystrybucji

w razie problemów wystarczy zabić proces UML bez żadnych skutków ubocznych

edukacja

przy pomocy UML-a możemy w bezpieczny i wygodny sposób poznawać zasady działania systemu operacyjnego, modyfikować jego jądro oraz administrowania siecią (możliwość budowy wirtualnej sieci)

Zastosowania UML

środowisko do testowania

dzięki UML-owi można uniknąć przeładowywania systemu, gdy wymagają tego przeprowadzane testy. Bardzo przydatna cecha dla administratorów

testowanie nowych wersji jądra/dystrybucji

w razie problemów wystarczy zabić proces UML bez żadnych skutków ubocznych

edukacja

przy pomocy UML-a możemy w bezpieczny i wygodny sposób poznawać zasady działania systemu operacyjnego, modyfikować jego jądro oraz administrowania siecią (możliwość budowy wirtualnej sieci)

Zastosowania UML

środowisko do testowania

dzięki UML-owi można uniknąć przeładowywania systemu, gdy wymagają tego przeprowadzane testy. Bardzo przydatna cecha dla administratorów

testowanie nowych wersji jądra/dystrybucji

w razie problemów wystarczy zabić proces UML bez żadnych skutków ubocznych

edukacja

przy pomocy UML-a możemy w bezpieczny i wygodny sposób poznawać zasady działania systemu operacyjnego, modyfikować jego jądro oraz administrowania siecią (możliwość budowy wirtualnej sieci)

Zastosowania UML

środowisko do testowania

dzięki UML-owi można uniknąć przeładowywania systemu, gdy wymagają tego przeprowadzane testy. Bardzo przydatna cecha dla administratorów

testowanie nowych wersji jądra/dystrybucji

w razie problemów wystarczy zabić proces UML bez żadnych skutków ubocznych

edukacja

przy pomocy UML-a możemy w bezpieczny i wygodny sposób poznawać zasady działania systemu operacyjnego, modyfikować jego jądro oraz administrowania siecią (możliwość budowy wirtualnej sieci)

Zastosowania UML cd.

wirtualizacja

tworzenie wirtualnych maszyn wieloprocesorowych lub posiadających wirtualny hardware

jailing

Dzięki UML w prosty sposób można izolować od hosta rzeczy, którym nie ufamy, w tym także użytkowników

rozwijanie jądra

rozwijanie i debugowanie jądra jak zwykłego procesu, za pomocą standardowych debuggerów (*gdb*) i profilerów (*gprof*)

Zastosowania UML cd.

wirtualizacja

tworzenie wirtualnych maszyn wieloprocesorowych lub posiadających wirtualny hardware

jailing

Dzięki UML w prosty sposób można izolować od hosta rzeczy, którym nie ufamy, w tym także użytkowników

rozwijanie jądra

rozwijanie i debugowanie jądra jak zwykłego procesu, za pomocą standardowych debuggerów (*gdb*) i profilerów (*gprof*)

Zastosowania UML cd.

wirtualizacja

tworzenie wirtualnych maszyn wieloprocesorowych lub posiadających wirtualny hardware

jailing

Dzięki UML w prosty sposób można izolować od hosta rzeczy, którym nie ufamy, w tym także użytkowników

rozwijanie jądra

rozwijanie i debugowanie jądra jak zwykłego procesu, za pomocą standardowych debuggerów (*gdb*) i profilerów (*gprof*)

Zastosowania UML cd.

wirtualizacja

tworzenie wirtualnych maszyn wieloprocesorowych lub posiadających wirtualny hardware

jailing

Dzięki UML w prosty sposób można izolować od hosta rzeczy, którym nie ufamy, w tym także użytkowników

rozwijanie jądra

rozwijanie i debugowanie jądra jak zwykłego procesu, za pomocą standardowych debuggerów (*gdb*) i profilerów (*gprof*)

Tryb TT (Tracing Thread) [DEPRECATED]

- każdemu procesowi odpowiada dokładnie 1 proces w systemie macierzystym
- dodatkowy proces (Tracing Thread), przechwytujący wywołania systemowe - nieoptymalne!
- każdy proces UML-a ma dostęp do przestrzeni adresowej jądra
- UML musi być uruchamiany w specjalnym trybie "jail", w którym pamięć zajmowana przez niego jest tylko do odczytu.
- napastnik ;) wie, że ma do czynienia z UML

Tryb SKAS (Single Kernel Address Space)

- mała liczba procesów widocznych z systemu macierzystego (w SKAS3 dokładnie 4)
- jądro w odrębnej przestrzeni adresowej niż procesy
- tryb SKAS3 wymaga łatki na jądro gospodarza
- tryb SKAS0 dostępny bezpośrednio (używany w dalszej części prezentacji)

Instalacja UML - wstęp

Następujące czynności pozwolą nam zainstalować User-mode Linux w trybie SKAS0:

1. kompilacja jądra gościa z flagami wspierającymi jego debugowanie oraz (obowiązkowo) ARCH=um
2. wirtualny system plików (w postaci pliku montowanego jako urządzenie loop device), na którym operować będzie nasz User-mode Linux

Instalacja UML - kernel

- ściągamy najnowszego Linuksa

```
$wget http://kernel.org/pub/linux/kernel/v2.6/
  linux-2.6.23.8.tar.bz2
$tar xjvf linux-2.6.23.8
$cd linux-2.6.23.8/
```

- konfigurujemy i kompilujemy jądro

```
$make defconfig ARCH=um
$make menuconfig ARCH=um

UML-specific options
Enable the block layer --> Block devices
Character devices
Kernel hacking

$make ARCH=um
```

Instalacja UML - konfiguracja jądra - kluczowe opcje

```
.config - Linux Kernel v2.6.23.8 Configuration
                              Kernel hacking
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted
  letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend:
  [*] built-in  [ ] excluded  <M> module  < > module capable
      ^(-)
      [ ]   Collect scheduler statistics
      [ ]   Collect kernel timers statistics
      [*]   Debug slab memory allocations
      [ ]     Memory leak debugging
      [ ] RT Mutex debugging, deadlock detection
      [ ] Built-in scriptable tester for rt-mutexes
      [ ] Spinlock and rw-lock debugging: basic checks
      [ ] Mutex debugging: basic checks
      [ ] Spinlock debugging: sleep-inside-spinlock checking
      [ ] Locking API boot-time self-tests
      [ ] kobject debugging
      [*] Compile the kernel with debug info
      [ ] Debug VM
      [ ] Debug linked list manipulation
      [*] Compile the kernel with frame pointers
      [*] Force gcc to inline functions marked 'inline'
      < > torture tests for RCU
      [ ] Fault-injection framework
      [ ] Enable gprof support
      [ ] Enable gcov support
      [ ] Stack utilization instrumentation

              <Select>    < Exit >    < Help >
```

Instalacja UML - konfiguracja jądra - kluczowe opcje cd.

```
.config - Linux Kernel v2.6.23.8 Configuration
                         Kernel hacking
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted
  letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend:
  [*] built-in  [ ] excluded  <M> module  < > module capable
    ^(-)
        [ ]    Collect scheduler statistics
        [ ]    Collect kernel timers statistics
        [*]    Debug slab memory allocations
        [ ]      Memory leak debugging
        [ ] RT Mutex debugging, deadlock detection
        [ ] Built-in scriptable tester for rt-mutexes
        [ ] Spinlock and rw-lock debugging: basic checks
        [ ] Mutex debugging: basic checks
        [ ] Spinlock debugging: sleep-inside-spinlock checking
        [ ] Locking API boot-time self-tests
        [ ] kobject debugging
        [*] Compile the kernel with debug info
        [ ] Debug VM
        [ ] Debug linked list manipulation
        [*] Compile the kernel with frame pointers
        [*] Force gcc to inline functions marked 'inline'
        < > torture tests for RCU
        [ ] Fault-injection framework
        [ ] Enable gprof support
        [ ] Enable gcov support
        [ ] Stack utilization instrumentation

                   <Select>    < Exit >    < Help >
```

Instalacja UML - wirtualny system plików

Jedyne, co pozostało do szczęśliwego końca instalacji to:

1. Wybieramy na stronie `uml.nagafix.co.uk` system plików odpowiedni dla naszej architektury. Wybieranie mniejszych dystrybucji może okazać się zwodnicze (np. Slackware), gdyż występują w nich problemy z obsługą *udev*, polecam więc system plików Debiana/Fedory
2.
```
cd ..
wget http://uml.nagafix.co.uk/Debian-4.0/
  Debian-4.0-AMD64-root_fs.bz2
bunzip2 Debian-4.0-AMD64-root_fs.bz2
```

Instalacja UML - zakończenie instalacji

Wzorem Herculesa Poirot możemy krzyknąć:

Eh bien!

```
$cd linux-2.6.23.8/
$./linux ubd0=../Debian-4.0-AMD64-root_fs
```

Instalacja UML - moduły

1. Instalujemy moduły uprzednio montując system plików, na którym będą one rezydowały:

```
$mkdir mnt
$sudo mount -o loop ../Debian-4.0-AMD64-root_fs mnt/
  -O uid=1000
$make modules ARCH=um
$make modules_install INSTALL_MOD_PATH=/mnt/loop/
  ARCH=um
$sudo umount mnt/
```

2. Sprawdzamy, że instalacja się powiodła:

```
uml# modprobe isofs
uml# lsmod
Module                   Size  Used by
isofs                   27016  0
```

Debugowanie jądra

1. Wywołanie GDB

```
$gdb --args ./linux ubd0=../Debian-4.0-AMD64-root_fs
```

2. Ignorowanie wewnętrznych sygnałów generowanych przez GDB

```
(gdb) handle SIGSEGV pass nostop noprint
(gdb) handle SIGUSR1 pass nostop noprint
```

3. Ustalenie wewnętrznych breakpointów i start systemu gościa

```
(gdb) b start_kernel
(gdb) run
```

Inne metody debugowania kernela

1. Podłączenie się do zewnętrznego debuggera

```
./linux ubd0=../Debian-4.0-AMD64-root_fs debug
  gdb-pid=pid_debuggera
```

2. Podłączenie się do aktywnego procesu jądra UML

```
$kill -USR1 pid_UML
```

Debugowanie modułów

Aby zdebugować moduł trzeba nakazać *gdb* załadowanie symboli modułu i miejsce, gdzie został załadowany

- Uruchom jądro w *gdb* i załaduj odpowiedni moduł

```
# modprobe loop
```

- Znajdź strukturę module_struct załadowanego modułu

```
(gdb) p modules
$1 = {next = 0x3502cea8, prev = 0x3502cea8}
```

- Wyłuskaj ze struktury pole module_core (uwaga na arytmetykę na wskaźniku)

```
(gdb) p *((struct module *)0x3502cea0)
$3 = {state = MODULE_STATE_LIVE, list =
  {next = 0x81e0dec, prev = 0x81e0dec},
[...]  module_core = 0x3502a000, [...]}
```

Debugowanie modułów cd.

- Załaduj symbole modułu (podstaw za ADDR module_core, a za PATH ścieżkę do odpowiedniego pliku .ko na hoście)

```
(gdb) add-symbol-file PATH ADDR
```

- Czas się pobawić:

```
(gdb) b loop_init
Breakpoint 1 at 0x3502a009: file /home/bd/bzdurki/
  linux-2.6.23.8/drivers/block/loop.c, line 1242.
(gdb) c
Continuing.

Breakpoint 1, loop_init ()
```

Krótki film fabularny o zastosowaniu GDB i UML

Na zakończenie zapraszamy Państwa na pokaz filmu fabularnego ilustrującego zastosowania przedstawionych technik

Mamy nadzieję, iż ten poglądowy tekst kultury dodatkowo zachęci Państwa do rozwijania jądra Linuksa, jako, że pozwoliliśmy sobie zaszyć w nim lokalizację dwóch *soft kernel lockups*, których eliminację pozostawiamy jako łatwe ćwiczenie dla widza.

Bibliografia

- Wikipedia
- prezentacje dotyczące odpluskwiania jądra z zeszłych lat
- `http://kernel.wikidot.com/kernel:teoria:debugowanie`
- man ltrace, man strace
- wszystko co było zaznaczone wcześniej, w treści
- GDB User Manual
- KGDB Documentation
- `http://www.linuxjournal.com/article/4525` – debugowanie modułów w KGDB
- `http://user-mode-linux.sourceforge.net`
- `http://lists.sourceforge.net/lists/listinfo/user-mode-linux-user`