

Debugowanie jądra systemu Linux

Norbert Potocki Krzysztof Gogól Piotr Dittwald

30 listopada 2007

Problemy z debugowaniem jądra

- musimy często resetować system
- czasami potrzebna jest kompilacja dużych fragmentów jądra
- niebezpieczeństwo uszkodzenia systemu plików
- błąd w jądrze może zawiesić cały system
- brak możliwości użycia debuggera — nic nie może “stać”
wyżej niż jądro

Sposoby debugowania jądra

- wypisywanie informacji debugujących podczas działania jądra — printk, itp.
- użycie wbudowanego w jądro debuggera — kdb
- wirtualizacja systemu i debugowanie jądra jako zwykłego procesu użytkownika — uml

printk() - cóż to takiego?

- funkcja wypisująca wiadomości jądra - działanie podobne do *printf()*
- wypisywane wiadomości są powiązane z różnymi poziomami logowania (ang. *loglevels*) lub priorytetami
- odpowiedni *loglevel* jest wskazywany przez makro

Przykłady:

```
printk(KERN_DEBUG "Here I am: %s:%i\n", __FILE__, __LINE_&.);  
printk(KERN_CRIT "I'm trashed; giving up on %p\n", ptr);
```

trochę o poziomach logowania (*loglevels*)

- istnieje 8 możliwych poziomów logowania, zdefiniowanych w pliku nagłówkowym `linux/kernel.h`
- mniejszy numer = wyższy priorytet
- brak wyspecyfikowanego w `printk` priorytetu \Rightarrow brany jest `DEFAULT_MESSAGE_LOGLEVEL` (wyspecyfikowany w `/kernel/printk.c`) - wartość ta zmieniała się w historii rozwoju Linuksa, więc sugerowane jest jawne specyfikowanie priorytetu
- wiadomość o priorytecie większym niż `console_loglevel` wyświetlana jest na konsoli (wiadomości jądra zależą także od działania `klogd` oraz `syslogd`)
- w `/proc/sys/kernel/printk` znajdują się następujące cztery wartości: `console_loglevel`, `default_message_loglevel`, `minimum_console_level` oraz `default_console_loglevel`

trochę o poziomach logowania (*loglevels*)

- istnieje 8 możliwych poziomów logowania, zdefiniowanych w pliku nagłówkowym `linux/kernel.h`
- mniejszy numer = wyższy priorytet
- brak wyspecyfikowanego w `printk` priorytetu \Rightarrow brany jest `DEFAULT_MESSAGE_LOGLEVEL` (wyspecyfikowany w `/kernel/printk.c`) - wartość ta zmieniała się w historii rozwoju Linuksa, więc sugerowane jest jawne specyfikowanie priorytetu
- wiadomość o priorytecie większym niż `console_loglevel` wyświetlana jest na konsoli (wiadomości jądra zależą także od działania `klogd` oraz `syslogd`)
- w `/proc/sys/kernel/printk` znajdują się następujące cztery wartości: `console_loglevel`, `default_message_loglevel`, `minimum_console_level` oraz `default_console_loglevel`

trochę o poziomach logowania (*loglevels*)

- istnieje 8 możliwych poziomów logowania, zdefiniowanych w pliku nagłówkowym `linux/kernel.h`
- mniejszy numer = wyższy priorytet
- brak wyspecyfikowanego w `printk` priorytetu \Rightarrow brany jest `DEFAULT_MESSAGE_LOGLEVEL` (wyspecyfikowany w `/kernel/printk.c`) - wartość ta zmieniała się w historii rozwoju Linuksa, więc sugerowane jest jawne specyfikowanie priorytetu
- wiadomość o priorytecie większym niż `console_loglevel` wyświetlana jest na konsoli (wiadomości jądra zależą także od działania `klogd` oraz `syslogd`)
- w `/proc/sys/kernel/printk` znajdują się następujące cztery wartości: `console_loglevel`, `default_message_loglevel`, `minimum_console_level` oraz `default_console_loglevel`

trochę o poziomach logowania (*loglevels*)

- istnieje 8 możliwych poziomów logowania, zdefiniowanych w pliku nagłówkowym `linux/kernel.h`
- mniejszy numer = wyższy priorytet
- brak wyspecyfikowanego w `printk` priorytetu \Rightarrow brany jest `DEFAULT_MESSAGE_LOGLEVEL` (wyspecyfikowany w `/kernel/printk.c`) - wartość ta zmieniała się w historii rozwoju Linuksa, więc sugerowane jest jawne specyfikowanie priorytetu
- wiadomość o priorytecie większym niż `console_loglevel` wyświetlana jest na konsoli (wiadomości jądra zależą także od działania `klogd` oraz `syslogd`)
- w `/proc/sys/kernel/printk` znajdują się następujące cztery wartości: `console_loglevel`, `default_message_loglevel`, `minimum_console_level` oraz `default_console_loglevel`

trochę o poziomach logowania (*loglevels*)

- istnieje 8 możliwych poziomów logowania, zdefiniowanych w pliku nagłówkowym `linux/kernel.h`
- mniejszy numer = wyższy priorytet
- brak wyspecyfikowanego w `printk` priorytetu \Rightarrow brany jest `DEFAULT_MESSAGE_LOGLEVEL` (wyspecyfikowany w `/kernel/printk.c`) - wartość ta zmieniała się w historii rozwoju Linuksa, więc sugerowane jest jawne specyfikowanie priorytetu
- wiadomość o priorytecie większym niż `console_loglevel` wyświetlana jest na konsoli (wiadomości jądra zależą także od działania `klogd` oraz `syslogd`)
- w `/proc/sys/kernel/printk` znajdują się następujące cztery wartości: `console_loglevel`, `default_message_loglevel`, `minimum_console_level` oraz `default_console_loglevel`

Przykłady

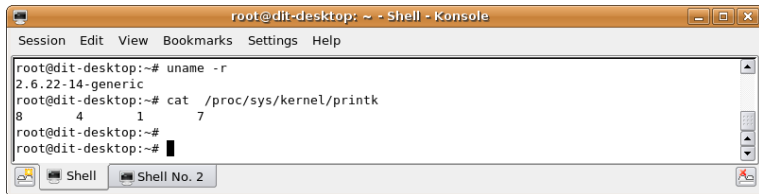
fragment pliku `/include/linux/kernel.h` (jądro 2.6.17.13):

```
#define KERN_EMERG      "<0>" /* system is unusable */
#define KERN_ALERT     "<1>" /* action must be taken immediately */
#define KERN_CRIT      "<2>" /* critical conditions */
#define KERN_ERR        "<3>" /* error conditions */
#define KERN_WARNING    "<4>" /* warning conditions */
#define KERN_NOTICE     "<5>" /* normal but significant condition */
#define KERN_INFO       "<6>" /* informational */
#define KERN_DEBUG      "<7>" /* debug-level messages */
```

Przykłady - c.d.

fragment pliku /kernel/printk.c (jądro 2.6.17.13):

```
/* printk's without a loglevel use this.. */  
#define DEFAULT_MESSAGE_LOGLEVEL 4 /* KERN_WARNING */  
  
/* We show everything that is MORE important than this.. */  
#define MINIMUM_CONSOLE_LOGLEVEL 1 /* Minimum loglevel we let people use */  
#define DEFAULT_CONSOLE_LOGLEVEL 7 /* anything MORE serious than KERN_DEBUG */
```



The screenshot shows a terminal window titled "root@dit-desktop: ~ - Shell - Konsole". The terminal content is as follows:

```
root@dit-desktop:~# uname -r  
2.6.22-14-generic  
root@dit-desktop:~# cat /proc/sys/kernel/printk  
8      4      1      7  
root@dit-desktop:~#  
root@dit-desktop:~# █
```

The terminal window has a menu bar with "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". At the bottom, there are tabs for "Shell" and "Shell No. 2".

Debugging System Faults

- fault != panic
- oops – zakłócenie w poprawnej pracy jądra Linuxa powodujące błąd
- Pomimo oopsa, system często działa dalej
- Ginie tylko proces odpowiedzialny za wywołanie błędu
- Tracona jest zapisywana przez proces pamięć

Debugging System Faults

- fault != panic
- oops – zakłócenie w poprawnej pracy jądra Linuxa powodujące błąd
- Pomimo oopsa, system często działa dalej
- Ginie tylko proces odpowiedzialny za wywołanie błędu
- Tracona jest zapisywana przez proces pamięć

Debugging System Faults

- fault != panic
- oops – zakłócenie w poprawnej pracy jądra Linuxa powodujące błąd
- Pomimo oopsa, system często działa dalej
- Ginie tylko proces odpowiedzialny za wywołanie błędu
- Tracona jest zapisywana przez proces pamięć

Debugging System Faults

- fault != panic
- oops – zakłócenie w poprawnej pracy jądra Linuxa powodujące błąd
- Pomimo oopsa, system często działa dalej
- Ginie tylko proces odpowiedzialny za wywołanie błędu
- Tracona jest zapisywana przez proces pamięć

Debugging System Faults

- fault != panic
- oops – zakłócenie w poprawnej pracy jądra Linuxa powodujące błąd
- Pomimo oopsa, system często działa dalej
- Ginie tylko proces odpowiedzialny za wywołanie błędu
- Tracona jest zapisywana przez proces pamięć

Z oopsa można odczytać wiele ciekawych informacji

```

Unable to handle kernel paging request at virtual address 000000000008000
tsk->{mm,active_mm}->context = 0000000000000410
tsk->{mm,active_mm}->pgd = fffff8000726e000

      | \
      @' /  \
     /  ..  \
    /    U    \
   /          \
  /            \
 /              \
/                \
\                /
 \              /
  \            /
   \          /
    \        /
     \      /
      \    /
       \  /
        \|

dpkg-preconfigu(16744): Oops
CPU[1]: local_irq_count[0] irq_s_running[0]
TSTATE: 0000004411f09601 TPC: 00000000004493d4 TNPC: 00000000004493d8 Y: 00000000 Not tainted
g0: 000000000062c00 g1: 00000000000001d g2: 000000000000002 g3: 000000000000000
g4: fffff80000000000 g5: 000000000000004 g6: fffff800009ee0000 g7: 000000000000001d
o0: 0000000000000000 o1: fffff800009ee0600 o2: 00000000004725e0 o3: 0000000000000000
o4: 000000000003300f o5: 000000002e706d00 sp: fffff800009ee2d11 ret_pc: 000000000040864c
l0: 0000000000000040 l1: 0000000011009600 l2: 0000000000623b2c l3: 0000000000000400
l4: 0000000000000000 l5: 0000000000000000 l6: 0000000000000000 l7: 0000000000000000
l8: 0000000000008120 l9: 0000000000000003 l10: 0000000000000001 l11: 0000000000000003
l12: 0000000000000001 l13: 0000000000000003 l14: 0000000000773000 l15: 0000000000000088 l16: fffff800009ee2dd1 l17: 000000000004725e8
Caller[00000000004725e8]
Caller[00000000004726e8]
Caller[00000000004ffdbc]
Caller[00000000004a2a98]
Caller[00000000004a2d1c]
Caller[000000000047d3f0]
Caller[000000000047dfcc]
Caller[000000000047e30c]
Caller[000000000047ab54]
Caller[000000000047062c]
Caller[0000000000436ea4]
Caller[0000000000410db4]
Caller[00000000702b2090]
Instruction DUMP: b6100019 a9520000 9190200f <ce6e0000> 0ac1c509 8143e00a a6062008 e05e2008 80a40013
CPU[0]: local_irq_count[0] irq_s_running[0]
TSTATE: 0000009980096603 TPC: 000000000041a5a4 TNPC: 000000000041a5a8 Y: 00000000 Not tainted
g0: 0000000000073dd40 g1: 00000000000000c2 g2: ffffffffcccc18 g3: 0000000000000001
g4: fffff80000000000 g5: fffff80000e4ac000 g6: 0000000000414000 g7: 0000000000000000
o0: 0000000000000000 o1: 00000000000012c1 o2: 000000000073dd40 o3: 000000000067e06
o4: 0000000000007fff o5: fffff80000fe80000 sp: 0000000000417531 ret_pc: 000000000041a5bc
  
```

Oops wg

lxr.linux.no/source/Documentation/oops-tracing.txt

- Co zrobić, gdy trafimy na oopsa?
- Znajdź Oopsa i wyślij go do osoby zajmującej się utrzymaniem systemu.
- Wyślij Oopsa do linux-kernel@vger.kernel.org
- Co zrobić jeśli błąd jest na tyle poważny, że nie można wpisywać poleceń?
- Przepisać ręcznie tekst z ekranu
- Zrobić zdjęcie aparatem cyfrowym

Oops wg

lxr.linux.no/source/Documentation/oops-tracing.txt

- Co zrobić, gdy trafimy na oopsa?
- Znajdź Oopsa i wyślij go do osoby zajmującej się utrzymaniem systemu.
- Wyślij Oopsa do linux-kernel@vger.kernel.org
- Co zrobić jeśli błąd jest na tyle poważny, że nie można wpisywać poleceń?
- Przepisać ręcznie tekst z ekranu
- Zrobić zdjęcie aparatem cyfrowym

Oops wg

lxr.linux.no/source/Documentation/oops-tracing.txt

- Co zrobić, gdy trafimy na oopsa?
- Znajdź Oopsa i wyślij go do osoby zajmującej się utrzymaniem systemu.
- Wyślij Oopsa do linux-kernel@vger.kernel.org
- Co zrobić jeśli błąd jest na tyle poważny, że nie można wpisywać poleceń?
- Przepisać ręcznie tekst z ekranu
- Zrobić zdjęcie aparatem cyfrowym

Oops wg

lxr.linux.no/source/Documentation/oops-tracing.txt

- Co zrobić, gdy trafimy na oopsa?
- Znajdź Oopsa i wyślij go do osoby zajmującej się utrzymaniem systemu.
- Wyślij Oopsa do linux-kernel@vger.kernel.org
- Co zrobić jeśli błąd jest na tyle poważny, że nie można wpisywać poleceń?
 - Przepisać ręcznie tekst z ekranu
 - Zrobić zdjęcie aparatem cyfrowym

Oops wg

lxr.linux.no/source/Documentation/oops-tracing.txt

- Co zrobić, gdy trafimy na oopsa?
- Znajdź Oopsa i wyślij go do osoby zajmującej się utrzymaniem systemu.
- Wyślij Oopsa do linux-kernel@vger.kernel.org
- Co zrobić jeśli błąd jest na tyle poważny, że nie można wpisywać poleceń?
- Przepisać ręcznie tekst z ekranu
- Zrobić zdjęcie aparatem cyfrowym

Oops wg

lxr.linux.no/source/Documentation/oops-tracing.txt

- Co zrobić, gdy trafimy na oopsa?
- Znajdź Oopsa i wyślij go do osoby zajmującej się utrzymaniem systemu.
- Wyślij Oopsa do linux-kernel@vger.kernel.org
- Co zrobić jeśli błąd jest na tyle poważny, że nie można wpisywać poleceń?
- Przepisać ręcznie tekst z ekranu
- Zrobić zdjęcie aparatem cyfrowym

Co pokazuje oops?

- 1 Numer oopsa
- 2 Status programu i stan rejestrów procesora
- 3 Zawartość stosu
- 4 Zapis ostatnich wywołań

Co pokazuje oops?

- 1 Numer oopsa
- 2 Status programu i stan rejestrów procesora
- 3 Zawartość stosu
- 4 Zapis ostatnich wywołań

Co pokazuje oops?

- 1 Numer oopsa
- 2 Status programu i stan rejestrów procesora
- 3 Zawartość stosu
- 4 Zapis ostatnich wywołań

Co pokazuje oops?

- 1 Numer oopsa
- 2 Status programu i stan rejestrów procesora
- 3 Zawartość stosu
- 4 Zapis ostatnich wywołań

Gdzie znajdują się te informacje?

```
krzys@krzys-laptop:~/wykminy50-54b77/p1
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] Oops: 0002 [#1] NUMER OOPSA
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] SMP
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] CPU: 1
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] EIP: 0060:[<e0746031>] Not tainted VLI
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] EFLAGS: 00210246 (2.6.20-16-generic #2) STATUS PROGRAMU
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] EIP is at init_module+0x11/0x1c [hello_mod]
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] eax: 00000000 ebx: e0746380 ecx: 00200008 STAN REJESTROW
2  edx: 00000000
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] esi: e0746380 edi: d964ff6c ebp: 00000001
c  esp: db4e7ed8
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] ds: 007b es: 007b ss: 0068
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] Process nodprobe (pid: 6780, ti=db4e6000 tas
k=k<9769830 task.ti=db4e6000)
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] Stack: e0746049 c014422d e07463c8 c036524e e
074638c c520d5d8 e07463c8 e074638c
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] e0746380 00000000 00000000 00000000 0
0000000 00000000 00000000 00000000
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
```

Gdzie znajdują się te informacje?

```
krzys@krzys-laptop:~/wyklady/S0_LAB/7/pl
Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] Process modprobe (pid: 6780, ti=db4e0000 tas
k=c9769030 task.ti=db4e0000)

Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] Stack: e0746049 c014422d e07463c8 c036524e e
074638c c52d05d8 e07463c8 e074638c

Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] e0746380 00000000 00000000 00000000 0
0000000 00000000 00000000 00000000

Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] 00000000 00000000 00000000 00000000 0
0000000 00000000 00000000 0000001b

Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] Call Trace:

Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] [sys_init_module+349/7072] sys_init_module+
0x15d/0x1ba0

Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] [sys_mmap2+205/208] sys_mmap2+0xcd/0xcd

Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] [sysenter_past_esp+105/109] sysenter_past_e
sp+0x69/0xa9

Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] -----

Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] Code: 3c 60 74 e0 e8 01 8c 9e df 83 c4 04 c3
 8d b6 00 00 00 00 8d bc 27 00 00 00 00 83 ec 04 c7 04 24 49 60 74 e0 e8 01 0b 9
e df 31 c0 <6> 05 00 00 00 64 83 c4 04 c3 36 3e 67 6f 6f 64 20 62 79

Message from syslogd@krzys-laptop at Fri Nov 23 07:26:18 2007 ...
krzys-laptop kernel: [ 2333.064000] EIP: [<e0746031>] init_module+0x11/0x1c [hel
to_mod] SS:ESP 0000:db4e7ed0
krzys@krzys-laptop:~/wyklady/S0_LAB/7/pl
```

ZAWARTOSC
STOSU

ZAPIS OSTATNIICH
WYWOLAN

Źródło oopsa

Moduł z błędem (pointer nie może być 0)

```
#include <linux/module.h>

MODULE_LICENSE("GPL");

int init_module(void)
{
    printk(KERN_INFO "Psujace Hello world\n");
    char * c = 0;
    *c = 100;
    return 0;
}

void cleanup_module(void)
{
    printk(KERN_INFO "good bye\n");
}
```

KLOGGD

Demon klogg dekoduje oops'y zanim zostaną zapisane w logu.
Uwaga na zdekodowany symbol EIP (Extended Instruction Pointer)

```
Unable to handle kernel NULL pointer dereference at virtual address \
00000000
printing eip:
c48370c3
*pde = 00000000
Oops: 0002
CPU: 0
EIP: 0010:[faulty: faulty_write+3/576]
EFLAGS: 00010286
eax: ffffffff ebx: c2c55ae0 ecx: c48370c0 edx: c2c55b00
esi: 0804d038 edi: 0804d038 ebp: c2337f8c esp: c2337f8c
ds: 0018 es: 0018 ss: 0018
Process cat (pid: 23413, stackpage=c2337000)
Stack: 00000001 c0135e6 c2c55ae0 0804d038 00000001 c2c55b00 c2336000 \
00000001
0804d038 bffffbd4 00000000 00000000 bffffbd4 c010b860 00000001 \
0804d038
00000001 00000001 0804d038 bffffbd4 00000004 0000002b 0000002b \
00000004
Call Trace: [sys_write+214/256] [system_call+52/56]
Code: c7 05 00 00 00 00 00 00 00 00 31 c0 89 ec 5d c3 8d b6 00 00
```

KLOGD

- Wskaźnik instrukcji był wykonywany w funkcji `faulty-write`
- KLOG ładuje informacje podczas uruchamiania, stąd jeśli moduł został załadowany po zainicjowaniu `klogd`. `Klogd` nie ma o nim informacji
- Aby temu przeciwdziałać można:
 - 1 wysyłać sygnał `SIGUSR1` po każdym załadowaniu nowego modułu
 - 2 uruchomić `klogd` z opcją `-p` („paranoid”), dzięki której `klogd` będzie ponownie wczytywać informacje o symbolach po każdym komunikacie `oops`
- `Klogd` potrzebuje aktualnej kopii `System.map`

KLOGD

- Wskaźnik instrukcji był wykonywany w funkcji faulty-write
- KLOG ładuje informacje podczas uruchamiania, stąd jeśli moduł został załadowany po zainicjowaniu klogd. Klogd nie ma o nim informacji
- Aby temu przeciwdziałać można:
 - 1 wysyłać sygnał SIGUSR1 po każdym załadowaniu nowego modułu
 - 2 uruchomić klogd z opcją -p („paranoid”), dzięki której klogd będzie ponownie wczytywać informacje o symbolach po każdym komunikacie oops
- Klogd potrzebuje aktualnej kopii System.map

KLOGD

- Wskaźnik instrukcji był wykonywany w funkcji faulty-write
- KLOG ładuje informacje podczas uruchamiania, stąd jeśli moduł został załadowany po zainicjowaniu klogd. Klogd nie ma o nim informacji
- Aby temu przeciwdziałać można:
 - 1 wysyłać sygnał SIGUSR1 po każdym załadowaniu nowego modułu
 - 2 uruchomić klogd z opcją -p („paranoid”), dzięki której klogd będzie ponownie wczytywać informacje o symbolach po każdym komunikacie oops
- Klogd potrzebuje aktualnej kopii System.map

KLOGD

- Wskaźnik instrukcji był wykonywany w funkcji faulty-write
- KLOG ładuje informacje podczas uruchamiania, stąd jeśli moduł został załadowany po zainicjowaniu klogd. Klogd nie ma o nim informacji
- Aby temu przeciwdziałać można:
 - 1 wysyłać sygnał SIGUSR1 po każdym załadowaniu nowego modułu
 - 2 uruchomić klogd z opcją `-p` („paranoid”), dzięki której klogd będzie ponownie wczytywać informacje o symbolach po każdym komunikacie `oops`
- Klogd potrzebuje aktualnej kopii `System.map`

KLOGD

- Wskaźnik instrukcji był wykonywany w funkcji faulty-write
- KLOG ładuje informacje podczas uruchamiania, stąd jeśli moduł został załadowany po zainicjowaniu klogd. Klogd nie ma o nim informacji
- Aby temu przeciwdziałać można:
 - 1 wysyłać sygnał SIGUSR1 po każdym załadowaniu nowego modułu
 - 2 uruchomić klogd z opcją -p („paranoid”), dzięki której klogd będzie ponownie wczytywać informacje o symbolach po każdym komunikacie oops
- Klogd potrzebuje aktualnej kopii System.map

KLOGD

- Wskaźnik instrukcji był wykonywany w funkcji faulty-write
- KLOG ładuje informacje podczas uruchamiania, stąd jeśli moduł został załadowany po zainicjowaniu klogd. Klogd nie ma o nim informacji
- Aby temu przeciwdziałać można:
 - 1 wysyłać sygnał SIGUSR1 po każdym załadowaniu nowego modułu
 - 2 uruchomić klogd z opcją -p („paranoid”), dzięki której klogd będzie ponownie wczytywać informacje o symbolach po każdym komunikacie oops
- Klogd potrzebuje aktualnej kopii System.map

KSYSMOOPS

- Czasem potrzeba więcej informacji niż dostarcza klogd.
- Często się zdarza, że klogd ginie w wyniku błędu systemu.
- Do serii 2.3, ksymoops był rozprowadzany ze źródłami jądra.
- Teraz można go ściągnąć np z <ftp://ftp.ocs.com.au/pub/ksymoops>

KSYSMOOPS

- Czasem potrzeba więcej informacji niż dostarcza klogd.
- Często się zdarza, że klogd ginie w wyniku błędu systemu.
- Do serii 2.3, ksyzoops był rozprowadzany ze źródłami jądra.
- Teraz można go ściągnąć np z <ftp://ftp.ocs.com.au/pub/ksyzoops>

KSYSMOOPS

- Czasem potrzeba więcej informacji niż dostarcza klogd.
- Często się zdarza, że klogd ginie w wyniku błędu systemu.
- Do serii 2.3, ksyzoops był rozprowadzany ze źródłami jądra.
- Teraz można go ściągnąć np z <ftp://ftp.ocs.com.au/pub/ksyzoops>

KSYSMOOPS

- Czasem potrzeba więcej informacji niż dostarcza klogd.
- Często się zdarza, że klogd ginie w wyniku błędu systemu.
- Do serii 2.3, ksymoops był rozprowadzany ze źródłami jądra.
- Teraz można go ściągnąć np z <ftp://ftp.ocs.com.au/pub/ksymoops>

KSYMOOPS

- Ksymoops poza komunikatem o błędzie potrzebuje kilku dodatkowych informacji:
 - 1 Plik System.map /usr/src/linux/System.map
 - 2 Lista modułów /proc/ksyms
 - 3 Symbole jądra zdefiniowane, gdy pojawia się oops /proc/ksyms
 - 4 Kopia obrazu jądra (vmlinuz, zlmage bzlmage)
 - 5 Lokalizacja modułów - można podawać za pomocą opcji -o

KSYMOOPS

- Ksymoops poza komunikatem o błędzie potrzebuje kilku dodatkowych informacji:
 - 1 Plik System.map /usr/src/linux/System.map
 - 2 Lista modułów /proc/ksyms
 - 3 Symbole jądra zdefiniowane, gdy pojawia się oops /proc/ksyms
 - 4 Kopia obrazu jądra (vmlinuz, zlmage bzlmage)
 - 5 Lokalizacja modułów - można podawać za pomocą opcji -o

KSYMOOPS

- Ksymoops poza komunikatem o błędzie potrzebuje kilku dodatkowych informacji:
 - 1 Plik System.map /usr/src/linux/System.map
 - 2 Lista modułów /proc/ksyms
 - 3 Symbole jądra zdefiniowane, gdy pojawia się oops /proc/ksyms
 - 4 Kopia obrazu jądra (vmlinuz, zlmage bzlmage)
 - 5 Lokalizacja modułów - można podawać za pomocą opcji -o

KSYMOOPS

- Ksymoops poza komunikatem o błędzie potrzebuje kilku dodatkowych informacji:
 - 1 Plik System.map /usr/src/linux/System.map
 - 2 Lista modułów /proc/ksyms
 - 3 Symbole jądra zdefiniowane, gdy pojawia się oops /proc/ksyms
 - 4 Kopia obrazu jądra (vmlinuz, zlmage bzlmage)
 - 5 Lokalizacja modułów - można podawać za pomocą opcji -o

KSYMOOPS

- Ksymoops poza komunikatem o błędzie potrzebuje kilku dodatkowych informacji:
 - 1 Plik System.map /usr/src/linux/System.map
 - 2 Lista modułów /proc/ksyms
 - 3 Symbole jądra zdefiniowane, gdy pojawia się oops /proc/ksyms
 - 4 Kopia obrazu jądra (vmlinuz, zlmage bzlmage)
 - 5 Lokalizacja modułów - można podawać za pomocą opcji -o

KSYMOOPS

- Ksymoops poza komunikatem o błędzie potrzebuje kilku dodatkowych informacji:
 - 1 Plik System.map /usr/src/linux/System.map
 - 2 Lista modułów /proc/ksyms
 - 3 Symbole jądra zdefiniowane, gdy pojawia się oops /proc/ksyms
 - 4 Kopia obrazu jądra (vmlinuz, zlmage bzlmage)
 - 5 Lokalizacja modułów - można podawać za pomocą opcji -o

KSYMOOPS

Wynik działania wygląda następująco:

```
>>>EIP; c48370c3 <[faulty] faulty .write+3/20> <=====
Trace; c01356e6 <sys .write+d6/100>
Trace; c010b860 <system .call+34/38>
Code; c48370c3 <[faulty] faulty .write+3/20>
00000000 <_EIP>:
Code; c48370c3 <[faulty] faulty .write+3/20> <=====
0: c7 05 00 00 00 movl $0x0,0x0 <=====
Code; c48370c8 <[faulty] faulty .write+8/20>
5: 00 00 00 00 00
Code; c48370cd <[faulty] faulty .write+d/20>
a: 31 c0 xorl %eax,%eax
Code; c48370cf <[faulty] faulty .write+f/20>
c: 89 ec movl %ebp,%esp
Code; c48370d1 <[faulty] faulty .write+11/20>
e: 5d popl %ebp
Code; c48370d2 <[faulty] faulty .write+12/20>
f: c3 ret
Code; c48370d3 <[faulty] faulty .write+13/20>
10: 8d b6 00 00 00 leal 0x0(%esi),%esi
Code; c48370d8 <[faulty] faulty .write+18/20>
15: 00
```


SYSTEM HANGS

- Chociaż większość błędów w kodzie jądra kończy się komunikatami oops, czasami system się całkowicie zawiesza. Dzieje się tak na przykład, gdy system zacznie wykonywać niekończącą się pętlę. Wtedy może nie pomóc nawet magiczne CTRL-ALT-DEL.
- Są dwa sposoby radzenie sobie z zawieszaniem systemu:
 - 1 uprzednio im zapobiegać
 - 2 debuggować po fakcie

SYSTEM HANGS

- Chociaż większość błędów w kodzie jądra kończy się komunikatami oops, czasami system się całkowicie zawiesza. Dzieje się tak na przykład, gdy system zacznie wykonywać niekończącą się pętlę. Wtedy może nie pomóc nawet magiczne CTRL-ALT-DEL.
- Są dwa sposoby radzenie sobie z zawieszaniem systemu:
 - 1 uprzednio im zapobiegać
 - 2 debuggować po fakcie

SYSTEM HANGS

- Chociaż większość błędów w kodzie jądra kończy się komunikatami oops, czasami system się całkowicie zawiesza. Dzieje się tak na przykład, gdy system zacznie wykonywać niekończącą się pętlę. Wtedy może nie pomóc nawet magiczne CTRL-ALT-DEL.
- Są dwa sposoby radzenie sobie z zawieszaniem systemu:
 - 1 uprzednio im zapobiegać
 - 2 debuggować po fakcie

SYSTEM HANGS

- Chociaż większość błędów w kodzie jądra kończy się komunikatami oops, czasami system się całkowicie zawiesza. Dzieje się tak na przykład, gdy system zacznie wykonywać niekończącą się pętlę. Wtedy może nie pomóc nawet magiczne CTRL-ALT-DEL.
- Są dwa sposoby radzenie sobie z zawieszaniem systemu:
 - 1 uprzednio im zapobiegać
 - 2 debuggować po fakcie

czym jest KDB?

- debugger dla Linuxa opartego na jądrze 2.{234}
- projekt *Silicon Graphics*

Krótko o łatach

- łata (patch) - plik tekstowy przechowujący różnice między dwiema wersjami kodu źródłowego
- Nakładanie łaty:
 - 1 wybieramy argument parametru `-p` (liczba ≥ 0) - ile ukośników usunąć ze ścieżki dostępnej w łacie
 - 2 porównane zostają katalogi plików - oryginalnego i zmodyfikowanego

Krótko o łatach - c.d.

Przykładowy początkowy fragment pliku `laty`

```
iff —git a/CREDITS b/CREDITS
— a/CREDITS
+++ b/CREDITS
@@ -1624,10 +1624,10 @@ E: ajoshi@shell.unixbox.com
D: fbdev hacking
N: Jesper Juhl
-E: juhl-lkml@dif.dk
-D: Various small janitor fixes , cleanups etc .
+E: jesper.juhl@gmail.com
+D: Various fixes , cleanups and minor features .
```

Cechy i możliwości

- przede wszystkim: nie wymaga dodatkowej maszyny do debugowania jądra
- wbudowany w jądro - łata (ang. *patch*) na jądro
- zakładanie pułapek (ang. *breakpoints*)
- kontrola zawartości pamięci jądra i struktur danych podczas pracy procesu
- formatowanie oraz wyświetlanie zebranych informacji
- kontrola operacji jądra (w tym zatrzymanie przy konkretnej instrukcji, wykonywanie "krok po kroku")
- sprawdzanie stanu procesu bez zewnętrznego systemu ani portu
- kontrola wielu procesorów

Cechy i możliwości

- przede wszystkim: nie wymaga dodatkowej maszyny do debugowania jądra
- wbudowany w jądro - łata (ang. *patch*) na jądro
- zakładanie pułapek (ang. *breakpoints*)
- kontrola zawartości pamięci jądra i struktur danych podczas pracy procesu
- formatowanie oraz wyświetlanie zebranych informacji
- kontrola operacji jądra (w tym zatrzymanie przy konkretnej instrukcji, wykonywanie "krok po kroku")
- sprawdzanie stanu procesu bez zewnętrznego systemu ani portu
- kontrola wielu procesorów

Cechy i możliwości

- przede wszystkim: nie wymaga dodatkowej maszyny do debugowania jądra
- wbudowany w jądro - łata (ang. *patch*) na jądro
- zakładanie pułapek (ang. *breakpoints*)
- kontrola zawartości pamięci jądra i struktur danych podczas pracy procesu
- formatowanie oraz wyświetlanie zebranych informacji
- kontrola operacji jądra (w tym zatrzymanie przy konkretnej instrukcji, wykonywanie "krok po kroku")
- sprawdzanie stanu procesu bez zewnętrznego systemu ani portu
- kontrola wielu procesorów

Cechy i możliwości

- przede wszystkim: nie wymaga dodatkowej maszyny do debugowania jądra
- wbudowany w jądro - łata (ang. *patch*) na jądro
- zakładanie pułapek (ang. *breakpoints*)
- kontrola zawartości pamięci jądra i struktur danych podczas pracy procesu
- formatowanie oraz wyświetlanie zebranych informacji
- kontrola operacji jądra (w tym zatrzymanie przy konkretnej instrukcji, wykonywanie "krok po kroku")
- sprawdzanie stanu procesu bez zewnętrznego systemu ani portu
- kontrola wielu procesorów

Cechy i możliwości

- przede wszystkim: nie wymaga dodatkowej maszyny do debugowania jądra
- wbudowany w jądro - łata (ang. *patch*) na jądro
- zakładanie pułapek (ang. *breakpoints*)
- kontrola zawartości pamięci jądra i struktur danych podczas pracy procesu
- formatowanie oraz wyświetlanie zebranych informacji
- kontrola operacji jądra (w tym zatrzymanie przy konkretnej instrukcji, wykonywanie "krok po kroku")
- sprawdzanie stanu procesu bez zewnętrznego systemu ani portu
- kontrola wielu procesorów

Cechy i możliwości

- przede wszystkim: nie wymaga dodatkowej maszyny do debugowania jądra
- wbudowany w jądro - łata (ang. *patch*) na jądro
- zakładanie pułapek (ang. *breakpoints*)
- kontrola zawartości pamięci jądra i struktur danych podczas pracy procesu
- formatowanie oraz wyświetlanie zebranych informacji
- kontrola operacji jądra (w tym zatrzymanie przy konkretnej instrukcji, wykonywanie "krok po kroku")
- sprawdzanie stanu procesu bez zewnętrznego systemu ani portu
- kontrola wielu procesorów

Cechy i możliwości

- przede wszystkim: nie wymaga dodatkowej maszyny do debugowania jądra
- wbudowany w jądro - łata (ang. *patch*) na jądro
- zakładanie pułapek (ang. *breakpoints*)
- kontrola zawartości pamięci jądra i struktur danych podczas pracy procesu
- formatowanie oraz wyświetlanie zebranych informacji
- kontrola operacji jądra (w tym zatrzymanie przy konkretnej instrukcji, wykonywanie "krok po kroku")
- sprawdzanie stanu procesu bez zewnętrznego systemu ani portu
- kontrola wielu procesorów

Cechy i możliwości

- przede wszystkim: nie wymaga dodatkowej maszyny do debugowania jądra
- wbudowany w jądro - łata (ang. *patch*) na jądro
- zakładanie pułapek (ang. *breakpoints*)
- kontrola zawartości pamięci jądra i struktur danych podczas pracy procesu
- formatowanie oraz wyświetlanie zebranych informacji
- kontrola operacji jądra (w tym zatrzymanie przy konkretnej instrukcji, wykonywanie "krok po kroku")
- sprawdzanie stanu procesu bez zewnętrznego systemu ani portu
- kontrola wielu procesorów

Dygresja (praktyczna), czyli instalacja łąty na przykładzie KDB

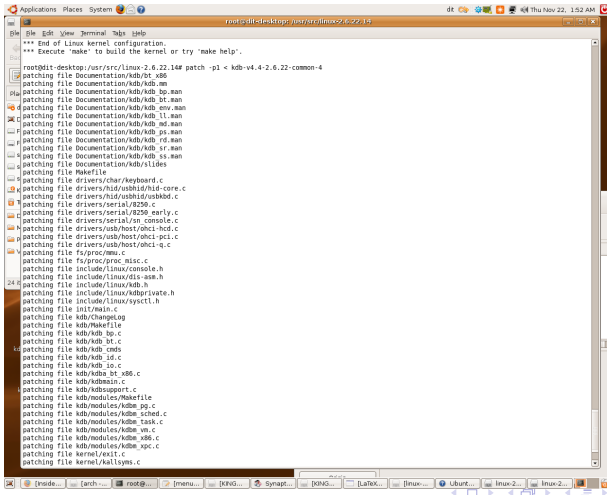
- pobieramy plik (<http://oss.sgi.com/projects/kdb/>)
- rozpakowujemy plik do katalogu ze źródłami Linuksa
- nakładamy patch
- konfigurujemy i kompilujemy jądro (za chwilę więcej o opcjach konfiguracji)

Oto najważniejsze kroki:

```
$ cd linux  
$ patch -p1 < kdb-xxx %nakładamy patch  
$ make menuconfig %interaktywna konfiguracja jadra
```


A teraz wersja (głównie) obrazkowa

Instalujemy patch ogólny - common



```
root@still-desktop: /usr/src/linux-2.6.22.14
*** End of Linux kernel configuration.
*** Execute 'make' to build the kernel or try 'make help'.

root@still-desktop: /usr/src/linux-2.6.22.14# patch -p1 < kdb-v4.4-2.6.22-common-4
patching file Documentation/kdb/ot_x86
patching file Documentation/kdb/kdb_mm
patching file Documentation/kdb/kdb_bp.man
patching file Documentation/kdb/kdb_bt.man
patching file Documentation/kdb/kdb_env.man
patching file Documentation/kdb/kdb_ll.man
patching file Documentation/kdb/kdb_md.man
patching file Documentation/kdb/kdb_ps.man
patching file Documentation/kdb/kdb_rd.man
patching file Documentation/kdb/kdb_sr.man
patching file Documentation/kdb/kdb_ss.man
patching file Documentation/kdb/slides
patching file Makefile
patching file drivers/char/keyboard.c
patching file drivers/hid/usbhid/hid-core.c
patching file drivers/hid/usbhid/usbkbd.c
patching file drivers/serial/8250.c
patching file drivers/serial/8250_early.c
patching file drivers/serial/sn_console.c
patching file drivers/usb/host/ohci-hcd.c
patching file drivers/usb/host/ohci-pci.c
patching file drivers/usb/host/ohci-qc.c
patching file fs/proc/mmu.c
patching file fs/proc/proc_basics.c
patching file include/linux/console.h
patching file include/linux/dis_asm.h
patching file include/linux/kdb.h
patching file include/linux/kdbprivate.h
patching file include/linux/sysctl.h
patching file init/main.c
patching file kdb/ChangeLog
patching file kdb/Makefile
patching file kdb/kdb_bp.c
patching file kdb/kdb_bt.c
patching file kdb/kdb_cmds
patching file kdb/kdb_id.c
patching file kdb/kdb_io.c
patching file kdb/kdb_bt_x86.c
patching file kdb/kdbmain.c
patching file kdb/kdbsupport.c
patching file kdb/modules/Makefile
patching file kdb/modules/kdbm_pg.c
patching file kdb/modules/kdbm_sched.c
patching file kdb/modules/kdbm_task.c
patching file kdb/modules/kdbm_vn.c
patching file kdb/modules/kdbm_x86.c
patching file kdb/modules/kdbm_xpc.c
patching file kernel/rstx.c
patching file kernel/kallsyms.c
```

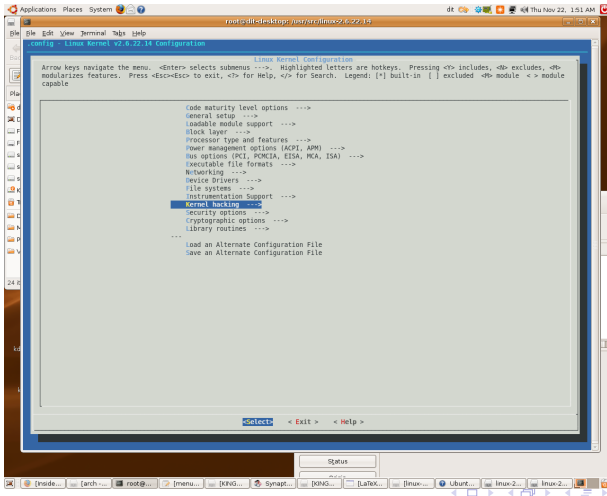
Instalujemy patch dla odpowiedniej architektury - tu: i386

```
root@i386:~/usr/src/linux-2.6.22.14# patch -p1 < kdb-v4.4-2.6.22-1386-2
patching file arch/i386/Kconfig.debug
patching file arch/i386/Makefile
patching file arch/i386/kdb/ChangeLog
patching file arch/i386/kdb/Makefile
patching file arch/i386/kdb/i386-dis.c
patching file arch/i386/kdb/kdb_cmds
patching file arch/i386/kdb/kdba_bp.c
patching file arch/i386/kdb/kdba_bt.c
patching file arch/i386/kdb/kdba_id.c
patching file arch/i386/kdb/kdba_io.c
patching file arch/i386/kdb/kdbasupport.c
patching file arch/i386/kdb/pc_keyb.h
patching file arch/i386/kernel/entry.S
patching file arch/i386/kernel/io_apic.c
patching file arch/i386/kernel/reboot.c
patching file arch/i386/kernel/traps.c
patching file include/asm-i386/ansidecl.h
patching file include/asm-i386/bfd.h
patching file include/asm-i386/kdb.h
patching file include/asm-i386/kdbprivate.h
patching file include/asm-i386/kdebug.h
patching file include/asm-i386/kmap_types.h
patching file include/asm-i386/mach-default/irq_vectors.h
patching file include/asm-i386/ptrace.h
root@i386:~/usr/src/linux-2.6.22.14#
```

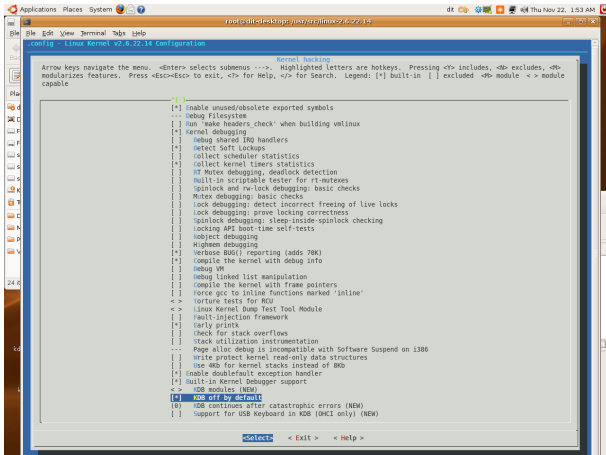
Wybrane opcje konfiguracji jądra

- CONFIG_KDB
- *Compile the kernel with frame pointers* ustawia flagę CONFIG_FRAME_POINTER - sugerowana (choć używa dodatkowego rejestru oraz ma negatywny wpływ na szybkość generowania kodu jądra)
- *KDB off by default* ustawia flagę CONFIG_KDB_OFF, która domyślnie wyłącza KDB

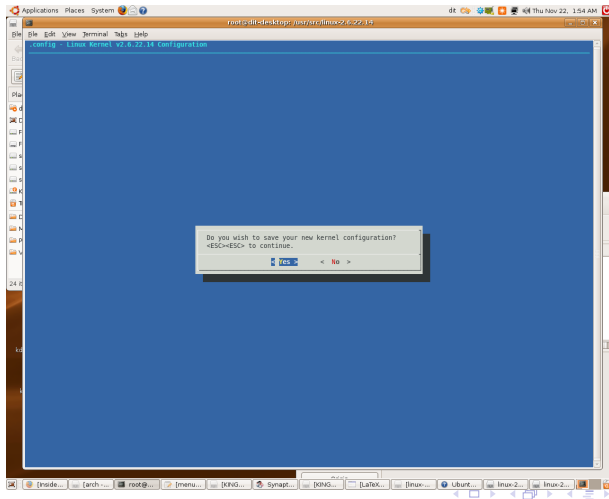
Po wpisaniu make menuconfig...



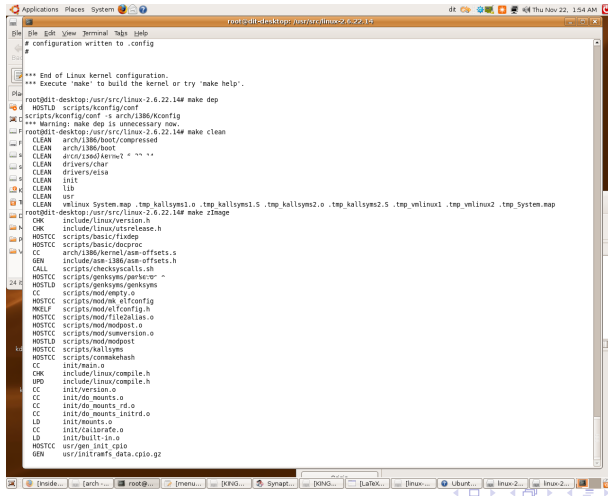
... wybieramy odpowiednie opcje dla debugowania (nie zapominamy o opcjach dla systemów plików itp.)



Zatwierdzamy opcje konfiguracji...



... i kompilujemy jądro



```
root@stl-desktop: /usr/src/linux-2.6.22.14
# configuration written to .config
#

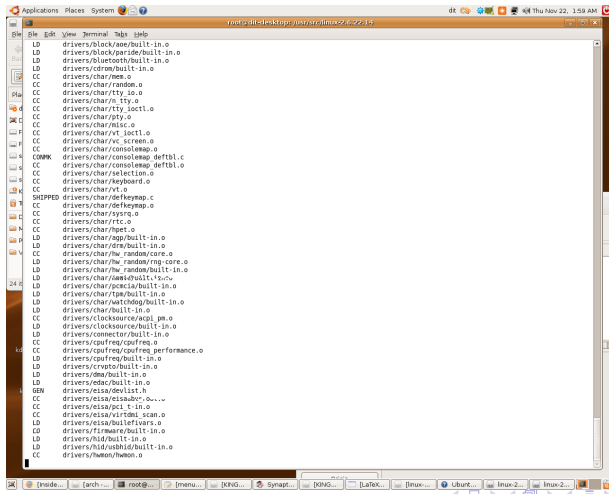
*** End of Linux kernel configuration.
*** Execute 'make' to build the kernel or try 'make help'.

root@stl-desktop: /usr/src/linux-2.6.22.14# make dep
HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf -s arch/i386/kconfig
*** Warning: make dep is unnecessary now.

root@stl-desktop: /usr/src/linux-2.6.22.14# make clean
CLEAN  arch/i386/boot/compressed
CLEAN  arch/i386/boot
CLEAN  arch/i386/kernel/z
CLEAN  drivers/char
CLEAN  drivers/eisa
CLEAN  init
CLEAN  lib
CLEAN  usr
CLEAN  vmlinux System.map .tmp.kallsyms1.o .tmp.kallsyms1.5 .tmp.kallsyms2.o .tmp.kallsyms2.5 .tmp.vmlinux1 .tmp.vmlinux2 .tmp_System.map

root@stl-desktop: /usr/src/linux-2.6.22.14# make zImage
CHK     include/linux/version.h
CHK     include/linux/utsrelease.h
HOSTCC  scripts/basic/fixdep
HOSTCC  scripts/basic/docproc
CC      arch/i386/kernel/asm-offsets.s
GEN     include/asm-i386/asm-offsets.h
CALL    scripts/checksyscalls.sh
HOSTCC  scripts/genksyms/genksyms.o
HOSTLD  scripts/genksyms/genksyms
CC      scripts/mod/empty.o
HOSTCC  scripts/mod/mk_elfconfig
MRELFP  scripts/mod/elfconfig.h
HOSTCC  scripts/mod/file2alias.o
HOSTCC  scripts/mod/modpost.o
HOSTCC  scripts/mod/suversion.o
HOSTLD  scripts/mod/modpost
HOSTCC  scripts/kallsyms
HOSTCC  scripts/conmakehash
CC      init/main.o
CHK     include/linux/compile.h
UPD     include/linux/compile.h
CC      init/version.o
CC      init/do_mounts.o
CC      init/do_mounts.rd.o
CC      init/do_mounts_initrd.o
LD      init/mounts.o
CC      init/calibrate.o
LD      init/built-in.o
HOSTCC  usr/gen_init_cpio
GEN     usr/initramfs_data.cpio.gz
```


(co może trochę potrwać)



Na końcu używamy jeszcze *make bzImage*

```

root@still-desktop: /usr/src/linux-2.6.22.14
AS arch/1386/lib/getuser.o
CC arch/1386/lib/memcpy.o
AS arch/1386/lib/putuser.o
AS arch/1386/lib/smemphr.o
CC arch/1386/lib/strstr.o
CC arch/1386/lib/usercopy.o
AR arch/1386/lib/lib.a
GEN version
CHK include/linux/compile.h
UPD include/linux/compile.h
CC init/version.o
LD init/built-in.o
LD .tmp_vmlinux1
KSYM .tmp_kallsyms1.5
AS .tmp_kallsyms1.o
LD .tmp_vmlinux2
KSYM .tmp_kallsyms2.5
AS .tmp_kallsyms2.o
LD vmlinux
SYSMAP System.map
SYSMAP .tmp_System.map
MODPOST vmlinux
WARNING: arch/1386/kernel/built-in.o(.text+0xba3b): Section mismatch: reference to .init.text:trap_init_f0bf_bug (between 'init_intel' and 'cpuid4_cac
he_lookup')
AS arch/1386/boot/bootsect.o
LD arch/1386/boot/bootsect
AS arch/1386/boot/setup.o
LD arch/1386/boot/setup
AS arch/1386/boot/compressed/head.o
CC arch/1386/boot/compressed/misc.o
OBJCOPY arch/1386/boot/compressed/vmlinux.bin
HOSTCC arch/1386/boot/compressed/relocs
RELOCS arch/1386/boot/compressed/vmlinux.relocs
BUILD arch/1386/boot/compressed/vmlinux.bin.all
GZIP arch/1386/boot/compressed/vmlinux.bin.gz
LD arch/1386/boot/compressed/piggy.o
LD arch/1386/boot/compressed/vmlinux
OBJCOPY arch/1386/boot/vmlinux.bin
HOSTCC arch/1386/boot/tools/build
BUILD arch/1386/boot/zImage
Root device is [8, 7]
Boot sector 512 bytes.
Setup is 7233 bytes.
System is 1787 kB
System is too big. Try using bzImage or modules.
make[1]: *** [arch/1386/boot/zImage] Error 1
make: *** [zImage] Error 2
root@still-desktop: /usr/src/linux-2.6.22.14# make bzImage
CHK include/linux/version.h
CHK include/linux/utsrelease.h
CALL scripts/checksyscalls.sh
CHK include/linux/compile.h

```

Pozostało jeszcze kilka kroków

- dodanie obrazu jądra do odpowiedniego miejsca , np. /boot (plik z obrazem: bzImage; powinien - dla architektury i386 - być w katalogu {katalog ze źródłami}/arch/i386/boot/)
- ewentualna zmiana ustawień pliku bootloadera (lilo, grub)

Uruchamianie

Jeśli flaga CONFIG_KDB_OFF nie jest aktywna, wówczas KDB jest domyślnie aktywne.

Aby je deaktywować, należy przekazać flagę kdb=on do jądra podczas bootowania lub użyć polecenia (gdy system został załadowany):

```
#echo "1" >/proc/sys/kernel/kdb
```

Przywołania KDB można dokonać za pomocą:

- 1 klawisza PAUSE
- 2 kombinacji Ctrt-A
- 3 gdy KDB jest włączone, wówczas zostaje przywołane automatycznie w przypadku *kernel panic*

Wyświetlanie zawartości pamięci oraz jej modyfikacja

wyświetlenie 15 linii pamięci począwszy od 0xc000000:

```
[0] kdb> md 0xc000000 15
```

zmiana zawartości pamięci 0xc000000 na 0x10:

```
[0] kdb> mm 0xc000000 0x10
```

Wyświetlanie zawartości rejestrów oraz ich modyfikacja

wyświetlenie zbioru rejestrów:

```
[0] kdb> rd
```

ustawienie zawartości rejestru ebx na 0x25:

```
[0] kdb> rm %ebx 0x25
```

Ustawianie pułapek

```
[1]kdb> bp scull_read  
Instruction(i) BP #0 at 0xc8833514 (scull_read)  
    is enabled on cpu 1  
[1]kdb> go
```

Więcej - przykładowe źródła

- /Documentation/kdb
- książka *Linux Device Drivers* - część o debuggowaniu
<http://www.xml.com/ldd/chapter/book/ch04.html>
- przewodnik dla początkujących użytkowników Kdb
<http://linuxdevices.com/articles/AT3761062961.html>

KGDB

- Czym jest KGDB?
 - patch do konkretnej wersji jądra
 - wymagane są dwie maszyny:
 - 1 target machine
 - 2 development machine
 - tu uruchamiamy jądro, które chcemy debugować
- maszyna, z której debugujemy przy użyciu gdb
- maszyny są połączone przez porty szeregowo
- debugowanie odbywa się w gdb

KGDB

- Czym jest KGDB?
- patch do konkretnej wersji jądra
- wymagane są dwie maszyny:
 - 1 target machine
 - 2 development machine
- tu uruchamiamy jądro, które chcemy debuggować
- maszyna, z której debuggujemy przy użyciu gdb
- maszyny są połączone przez porty szeregowo
- debuggowanie odbywa się w gdb

KGDB

- Czym jest KGDB?
- patch do konkretnej wersji jądra
- wymagane są dwie maszyny:
 - 1 target machine
 - tu uruchamiamy jądro, które chcemy debuggować
 - 2 development machine
 - maszyna, z której debuggujemy przy użyciu gdb
 - maszyny są połączone przez porty szeregowo
 - debuggowanie odbywa się w gdb

KGDB

- Czym jest KGDB?
- patch do konkretnej wersji jądra
- wymagane są dwie maszyny:
 - 1 target machine
 - tu uruchamiamy jądro, które chcemy debuggować
 - 2 development machine
 - maszyna, z której debuggujemy przy użyciu gdb
 - maszyny są połączone przez porty szeregowo
 - debuggowanie odbywa się w gdb

KGDB

- Czym jest KGDB?
- patch do konkretnej wersji jądra
- wymagane są dwie maszyny:
 - 1 target machine
 - tu uruchamiamy jądro, które chcemy debuggować
 - 2 development machine
- maszyna, z której debuggujemy przy użyciu gdb
- maszyny są połączone przez porty szeregowo
- debuggowanie odbywa się w gdb

KGDB

- Czym jest KGDB?
- patch do konkretnej wersji jądra
- wymagane są dwie maszyny:
 - 1 target machine
 - 2 development machine
- maszyna, z której debugujemy przy użyciu gdb
- maszyny są połączone przez porty szeregowo
- debugowanie odbywa się w gdb

KGDB

- Czym jest KGDB?
- patch do konkretnej wersji jądra
- wymagane są dwie maszyny:
 - 1 target machine
 - 2 development machine
- maszyna, z której debugujemy przy użyciu gdb
 - maszyny są połączone przez porty szeregowo
 - debugowanie odbywa się w gdb

KGDB

- Czym jest KGDB?
- patch do konkretnej wersji jądra
- wymagane są dwie maszyny:
 - 1 target machine
- tu uruchamiamy jądro, które chcemy debuggować
- 2 development machine
- maszyna, z której debuggujemy przy użyciu gdb
- maszyny są połączone przez porty szeregowo
- debuggowanie odbywa się w gdb

KGDB

- Czym jest KGDB?
- patch do konkretnej wersji jądra
- wymagane są dwie maszyny:
 - 1 target machine
- tu uruchamiamy jądro, które chcemy debuggować
- 2 development machine
- maszyna, z której debuggujemy przy użyciu gdb
- maszyny są połączone przez porty szeregowo
- debuggowanie odbywa się w gdb

KGDB

- zalety:
 - 1 śledzenie wykonania kodu jądra instrukcja po instrukcji (w C lub w assemblerze)
 - 2 Breakpointy, Obserwowanie wartości zmiennych i rejestrów
 - 3 Obsługa wątków
 - 4 Debuggowanie modułów
- wady:
 - 1 dwie maszyny i port szeregowy
 - 2 kgdb działa tylko dla starszych wersji jądra

KGDB

- Instalacja i uruchamianie:
 - 1 instalacja patchy dla odpowiedniej wersji jądra
 - 2 połączenie dwóch maszyn
 - 3 support files dla gdb
- Szczegółowe informacje znajdują się w patchu w pliku:
`Documentation/i386/gdb-serial.txt`

KGDB

- Instalacja i uruchamianie:
 - 1 instalacja patchy dla odpowiedniej wersji jądra
 - 2 połączenie dwóch maszyn
 - 3 support files dla gdb
- Szczegółowe informacje znajdują się w patchu w pliku:
`Documentation/i386/gdb-serial.txt`

KGDB

- Instalacja i uruchamianie:
 - 1 instalacja patchy dla odpowiedniej wersji jądra
 - 2 połączenie dwóch maszyn
 - 3 support files dla gdb
- Szczegółowe informacje znajdują się w patchu w pliku:
`Documentation/i386/gdb-serial.txt`

KGDB

- Instalacja i uruchamianie:
 - 1 instalacja patchy dla odpowiedniej wersji jądra
 - 2 połączenie dwóch maszyn
 - 3 support files dla gdb
- Szczegółowe informacje znajdują się w patchu w pliku:
`Documentation/i386/gdb-serial.txt`

KGDB

- Instalacja i uruchamianie:
 - 1 instalacja patchy dla odpowiedniej wersji jądra
 - 2 połączenie dwóch maszyn
 - 3 support files dla gdb
- Szczegółowe informacje znajdują się w patchu w pliku:
`Documentation/i386/gdb-serial.txt`

KGDB

```

aterm
panic: brenfree: removing a buffer not on a queue
panic messages:
----
panic: ffs_blkfree: freeing free block

syncing disks, buffers remaining... panic: brenfree: removing a buffer not on a queue
Uptime: 3h45m25s
Dumping 255 MB
ata0: resetting devices ..
done
_16[CTRL-C to abort] [CTRL-C to abort] [CTRL-C to abort] [CTRL-C to abort] 32 48 64 80 96 112 128 144 160 176 192 208 224 240
----
Reading symbols from /usr/obj/usr/src/sys/PAW/modules/usr/src/sys/modules/linux/linux.ko.debug...done.
Loaded symbols for /usr/obj/usr/src/sys/PAW/modules/usr/src/sys/modules/linux/linux.ko.debug
Reading symbols from /boot/kernel/snd_via8233.ko...done.
Loaded symbols for /boot/kernel/snd_via8233.ko
Reading symbols from /boot/kernel/snd_pcm.ko...done.
Loaded symbols for /boot/kernel/snd_pcm.ko
Reading symbols from /usr/obj/usr/src/sys/PAW/modules/usr/src/sys/modules/acpi/acpi.ko.debug...done.
Loaded symbols for /usr/obj/usr/src/sys/PAW/modules/usr/src/sys/modules/acpi/acpi.ko.debug
Reading symbols from /boot/kernel/radeon.ko...done.
Loaded symbols for /boot/kernel/radeon.ko
#0 doadump () at /usr/src/sys/kern/kern_shutdown.c:238
238      dumping++
(kgdb) where
#0 doadump () at /usr/src/sys/kern/kern_shutdown.c:238
#1 0xc02213b8 in boot (howto=260) at /usr/src/sys/kern/kern_shutdown.c:370
#2 0xc02216ab in panic () at /usr/src/sys/kern/kern_shutdown.c:543
#3 0xc025fd70 in brenfree (bp=0xc7802088) at /usr/src/sys/kern/vfs_bio.c:648
#4 0xc025fca5 in brenfree (bp=0x0) at /usr/src/sys/kern/vfs_bio.c:630
#5 0xc02630e8 in vop_stdfsync (ap=0xd5d339f8) at /usr/src/sys/kern/vfs_default.c:759
#6 0xc01e0c00 in spec_fsync (ap=0xd5d339f8) at /usr/src/sys/fs/specfs/spec_vnops.c:418
#7 0xc01e0298 in spec_vnoperate (ap=0x0) at /usr/src/sys/fs/specfs/spec_vnops.c:123
#8 0xc02fd24d in ffs_sync (mp=0xc26ee000, waitfor=2, cred=0xc0eb5e80, td=0xc03f35a0) at vnode_if.h:612
#9 0xc0274c7b in sync (td=0xc03f35a0, uap=0x0) at /usr/src/sys/kern/vfs_syncalls.c:137
#10 0xc0220fc2 in boot (howto=256) at /usr/src/sys/kern/kern_shutdown.c:279
#11 0xc02216ab in panic () at /usr/src/sys/kern/kern_shutdown.c:543
#12 0xc02651ba in ffs_blkfree (fs=0xc2758000) devvp=0xc27155b4, bno=3702640, size=16384, inum=35768) at /usr/src/sys/ufs/ufs/ufs_alloc.c:1768
#13 0xc02f51af in indir_trunc (freeblks=0xc2de2500, dbn=24882976, level=0, lbn=12, count=0xd5d33c10)
    at /usr/src/sys/ufs/ufs/ufs_softdep.c:2615
#14 0xc02f4c45 in handle_workitem_freeblocks (freeblks=0xc2de2500, flags=0) at /usr/src/sys/ufs/ufs/ufs_softdep.c:2480
#15 0xc02f1daa in process_worklist_item (matchmnt=0x0, flags=0) at /usr/src/sys/ufs/ufs/ufs_softdep.c:756
#16 0xc02f14df in softdep_process_worklist (matchmnt=0x0) at /usr/src/sys/ufs/ufs/ufs_softdep.c:622
#17 0xc02714fe in sched_sync () at /usr/src/sys/kern/vfs_subr.c:1761
#18 0xc020e24e in fork_exit (callout=0xc0271210 <sched_sync>, arg=0x0, frame=0x0) at /usr/src/sys/kern/kern_fork.c:793
    
```


KGDB

- źródła jądra (www.kernel.org/pub/linux/kernel) - szczególnie v2.6
- dokumentacja do KGDB (kgdb.linsyssoft.com/downloads/kgdb-2/kgdb_docu_full-2.4.pdf.bz2) - do wersji 2.4
- patche z KGDB (kgdb.linsyssoft.com/downloads.htm) - szczególnie v2.4
- starsze wersje KGDB (www.popies.net/kgdb/index.html) - dla jąder 2.4
- instalacja (www.kernelhacking.org/docs/kernelhacking-HOWTO/indexs09.html) - mało szczegółowa

KGDB

- źródła jądra (www.kernel.org/pub/linux/kernel) - szczególnie v2.6
- dokumentacja do KGDB (kgdb.linsyssoft.com/downloads/kgdb-2/kgdb_docu_full-2.4.pdf.bz2) - do wersji 2.4
- patche z KGDB (kgdb.linsyssoft.com/downloads.htm) - szczególnie v2.4
- starsze wersje KGDB (www.popies.net/kgdb/index.html) - dla jąder 2.4
- instalacja (www.kernelhacking.org/docs/kernelhacking-HOWTO/indexs09.html) - mało szczegółowa

KGDB

- źródła jądra (www.kernel.org/pub/linux/kernel) - szczególnie v2.6
- dokumentacja do KGDB (kgdb.linsyssoft.com/downloads/kgdb-2/kgdb_docu_full-2.4.pdf.bz2) - do wersji 2.4
- patche z KGDB (kgdb.linsyssoft.com/downloads.htm) - szczególnie v2.4
- starsze wersje KGDB (www.popies.net/kgdb/index.html) - dla jąder 2.4
- instalacja (www.kernelhacking.org/docs/kernelhacking-HOWTO/indexs09.html) - mało szczegółowa

KGDB

- źródła jądra (www.kernel.org/pub/linux/kernel) - szczególnie v2.6
- dokumentacja do KGDB (kgdb.linsyssoft.com/downloads/kgdb-2/kgdb_docu_full-2.4.pdf.bz2) - do wersji 2.4
- patche z KGDB (kgdb.linsyssoft.com/downloads.htm) - szczególnie v2.4
- starsze wersje KGDB (www.popies.net/kgdb/index.html) - dla jąder 2.4
- instalacja (www.kernelhacking.org/docs/kernelhacking-HOWTO/indexs09.html) - mało szczegółowa

KGDB

- źródła jądra (www.kernel.org/pub/linux/kernel) - szczególnie v2.6
- dokumentacja do KGDB (kgdb.linsyssoft.com/downloads/kgdb-2/kgdb_docu_full-2.4.pdf.bz2) - do wersji 2.4
- patche z KGDB (kgdb.linsyssoft.com/downloads.htm) - szczególnie v2.4
- starsze wersje KGDB (www.popies.net/kgdb/index.html) - dla jąder 2.4
- instalacja (www.kernelhacking.org/docs/kernelhacking-HOWTO/indexs09.html) - mało szczegółowa

KGDB

- instalacja (oslab.info/index.php/Misc/KGDB) - pod VMware
- gdb (safari.phptr.com/0131492470/ch03#ch03) - poradnik
- gdbmod ([kgdb.linsyssoft.com/downloads/gdbmod - 2.4.bz2](http://kgdb.linsyssoft.com/downloads/gdbmod-2.4.bz2))
- Zmodyfikowane gdb dające możliwość debugowania modułów
- sDebug (www.linuxjournal.com/article/4525) modułów za pomocą KGDB

KGDB

- instalacja (oslab.info/index.php/Misc/KGDB) - pod VMware
- gdb (safari.phptr.com/0131492470/ch03#ch03) - poradnik
- gdbmod ([kgdb.linsyssoft.com/downloads/gdbmod - 2.4.bz2](http://kgdb.linsyssoft.com/downloads/gdbmod-2.4.bz2))
- Zmodyfikowane gdb dające możliwość debugowania modułów
- sDebug (www.linuxjournal.com/article/4525) modułów za pomocą KGDB

KGDB

- instalacja (oslab.info/index.php/Misc/KGDB) - pod VMware
- gdb (safari.phptr.com/0131492470/ch03#ch03) - poradnik
- gdbmod ([kgdb.linsyssoft.com/downloads/gdbmod – 2.4.bz2](http://kgdb.linsyssoft.com/downloads/gdbmod-2.4.bz2))
- Zmodyfikowane gdb dające możliwość debugowania modułów
- sDebug (www.linuxjournal.com/article/4525) modułów za pomocą KGDB

KGDB

- instalacja (oslab.info/index.php/Misc/KGDB) - pod VMware
- gdb (safari.phptr.com/0131492470/ch03#ch03) - poradnik
- gdbmod ([kgdb.linsyssoft.com/downloads/gdbmod - 2.4.bz2](http://kgdb.linsyssoft.com/downloads/gdbmod-2.4.bz2))
- Zmodyfikowane gdb dające możliwość debugowania modułów
- sDebug (www.linuxjournal.com/article/4525) modułów za pomocą KGDB

UML w teorii

- UML = user mode linux..
- .. czyli system uruchomiony w przestrzeni programów użytkownika
- jedna z metod wirtualizacji
- możliwość zagnieżdżania systemów
- dwa tryby działania:
 - TT - Tracing Thread
 - SKAS - Single Kernel Address Space
- jądra gospodarza i gościa
- może korzystać z systemu plików hosta, lub zawartego w pliku

UML - skąd go wziąć

- w wersjach jądra $< 2.6.9$ — jako łątka
- w wersjach jądra $\geq 2.6.9$ — jest włączony do drzewa jądra
- większość starszych jąder wymaga dodatkowych łatek..
- .. np. 2.6.17.13 — o czym nikt wcześniej nie pisał..
- .. a na laby może się przydać ;)

UML w praktyce

- metoda prostsza:
 - postępujemy zgodnie z instrukcjami ze strony <http://user-mode-linux.sourceforge.net/> i pobieramy gotowe jądro oraz system plików
- metoda trudniejsza:
 - kompilujemy własne jądro
 - budujemy własny system plików — więcej na stronie UML'a

Debugowanie jądra 2.6.17.13

Pobieramy jądro

```
$ wget ftp://ftp.kernel.org/pub/linux/kernel/v2.6/linux-2.6.17.13.tar.bz2  
$ wget http://user-mode-linux.sourceforge.net/old/work/current/2.6/2.6.17/patches/jmpbuf
```

Rozpakowujemy

```
$ tar -xjf linux-2.6.17.13.tar.bz2  
$ cd linux-2.6.17.13  
$ patch -p1 < ../jmpbuf
```

Konfigurujemy i kompilujemy

```
$ make mrproper  
$ make menuconfig ARCH=um  
$ make linux ARCH=um
```

Budowanie jądra 2.6.17.13

.. i odpalamy

```
$ linux ubd0=root_fs mem=256M
```

.. a gdy chcemy debugować – w trybie SKAS

```
$ gdb linux  
(gdb) handle SIGSEGV pass nostop noprint  
(gdb) handle SIGUSR1 pass nostop noprint  
(gdb) run ubd0=root_fs mem=256M
```

Debugowanie jądra 2.6.17.13 - problem interaktywności

Odnajdujemy najniższy pid procesu uruchomionego przez nas jądra

Przywracamy do życia gdb

```
$ kill -INT <nasz_pid>
```

Debugowanie jądra 2.6.17.13 - moduły

- rozpoczynamy debugowanie jądra

Dodajemy debugowanie modułu

```
(gdb) break __link_module
(gdb) continue
(gdb) print ((struct module *) _mod)->module_core
(gdb) add-symbol-file SCIEZKA_DO_MODULU_NA_HOSCIE <wynik_print>
(gdb) break funkcja_z_modulu
(gdb) continue
```


Bonus - prezentacja UML na jądrze 2.6.17.13 - na potrzeby labów