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, zdefinowanych w pliku nagłówkowym linux/kernel.h
- mniejszy numer = wyższy priorytet
- brak wyspecyfikowanego w printfk priorytetu ⇒ 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, zdefinowanych w pliku nagłówkowym linux/kernel.h
- mniejszy numer = wyższy priorytet
- brak wyspecyfikowanego w printfk priorytetu ⇒ 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, zdefinowanych w pliku nagłówkowym linux/kernel.h
- mniejszy numer = wyższy priorytet
- brak wyspecyfikowanego w printfk priorytetu ⇒ 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, zdefinowanych w pliku nagłówkowym linux/kernel.h
- mniejszy numer = wyższy priorytet
- brak wyspecyfikowanego w printfk priorytetu ⇒ 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, zdefinowanych w pliku nagłówkowym linux/kernel.h
- mniejszy numer = wyższy priorytet
- brak wyspecyfikowanego w printfk priorytetu ⇒ 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):

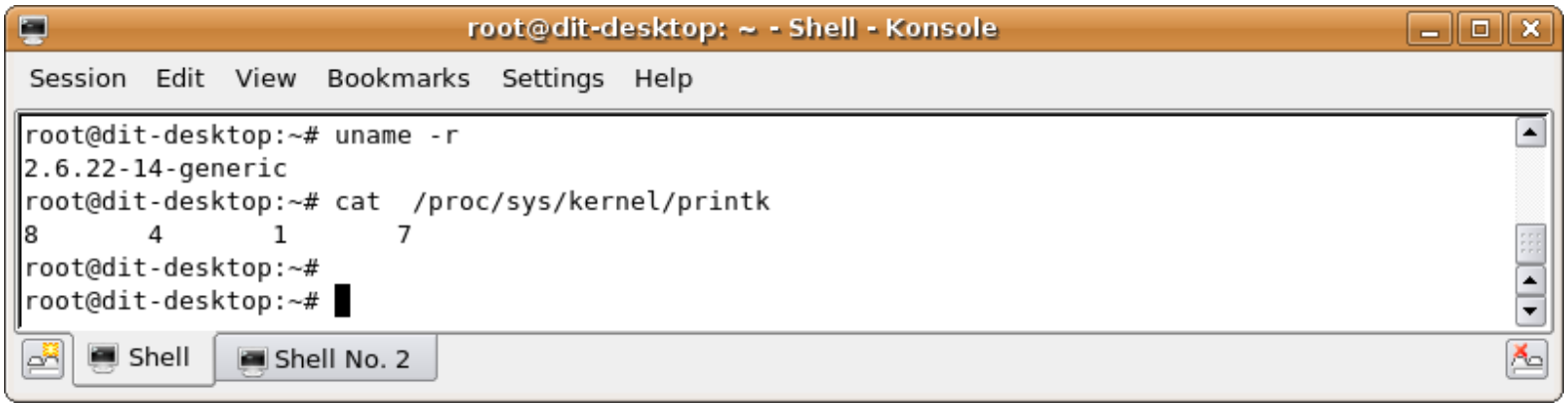
```
#define KERN_EMERG      "<0>"   /* system is unusable                   */
#define KERN_ALERT      "<1>"   /* action must be taken immediately     */
#define KERN_CRIT       "<2>"   /* critical conditions                  */
#define KERN_ERR        "<3>"   /* error conditions                     */
#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 */
```



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 0000000000008000
tsk->{mm,active_mm}->context = 0000000000000410
tsk->{mm,active_mm}->pgd = fffff8000726e000
              \|/ ____ \|/
              "@'/ .. \`@"
              /_| \__/ |_\
                 \__U_/
dpkg-preconfigu(16744): Oops
CPU[1]: local_irq_count[0] irqs_running[0]
TSTATE: 0000004411f09601 TPC: 00000000004493d4 TNPC: 00000000004493d8 Y: 00000000    Not tainted
g0: 0000000000662c00 g1: 000000000000001d g2: 0000000000000002 g3: 0000000000000000
g4: fffff80000000000 g5: 0000000000000004 g6: fffff80009ee0000 g7: 000000000000001d
o0: 0000000000000001 o1: fffff80009ee0600 o2: 00000000004725e0 o3: 0000000000000000
o4: 000000000003300f o5: 000000002e706d00 sp: fffff80009ee2d11 ret_pc: 0000000000040864c
l0: 0000000000000040 l1: 0000000011009600 l2: 0000000000623b2c l3: 0000000000000400
l4: 0000000000000000 l5: 0000000000000000 l6: 0000000000000000 l7: 0000000000000008
i0: 0000000000008120 i1: 0000000000000003 i2: 0000000000000001 i3: 0000000000000003
i4: 0000000000773000 i5: 0000000000000088 i6: fffff80009ee2dd1 i7: 00000000004725e8
Caller[00000000004725e8]
Caller[00000000004726e8]
Caller[00000000004ffdbc]
Caller[00000000004a2a98]
Caller[00000000004a2d1c]
Caller[000000000047d3f0]
Caller[000000000047dfcc]
Caller[000000000047e30c]
Caller[000000000047eb54]
Caller[000000000047062c]
Caller[0000000000436ee4]
Caller[0000000000410db4]
Caller[00000000702b2090]
Instruction DUMP: b6100019  a9520000  9190200f <ce6e0000> 0ac1c509  8143e00a  a6062008  e05e2008  80a40013
CPU[0]: local_irq_count[0] irqs_running[0]
TSTATE: 0000009980009603 TPC: 0000000000041a5a4 TNPC: 0000000000041a5a8 Y: 00000000    Not tainted
g0: 000000000073dd40 g1: 00000000000000c2 g2: fffffffffffffc18 g3: 0000000000000001
g4: fffff80000000000 g5: fffff8000e4ac000 g6: 0000000000414000 g7: 0000000000000000
o0: 0000000000000000 o1: 00000000000012c1 o2: 000000000073dd40 o3: 0000000000067e06
o4: 0000000000007fff o5: fffff8000fe80000 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

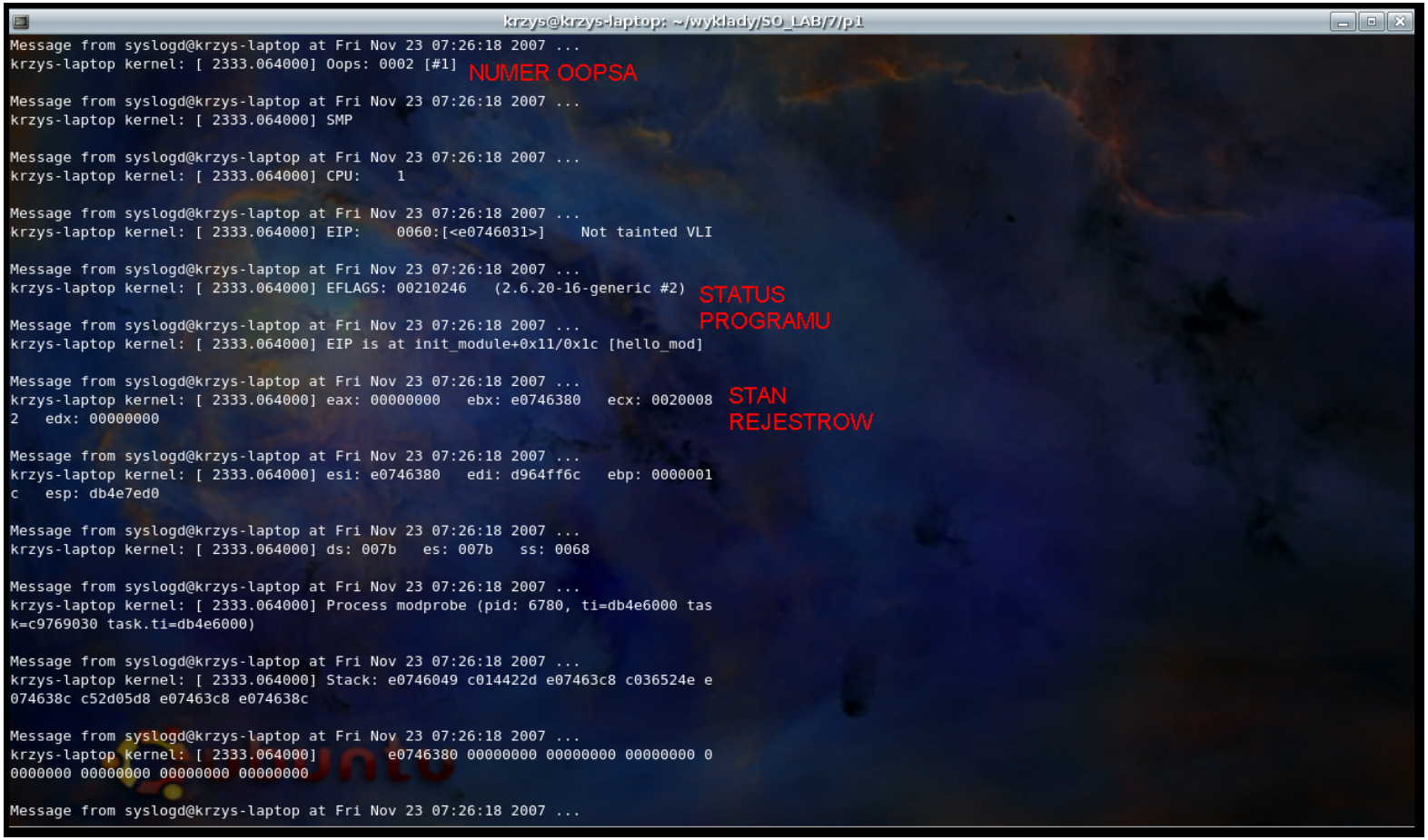
Co pokazuje oops?

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

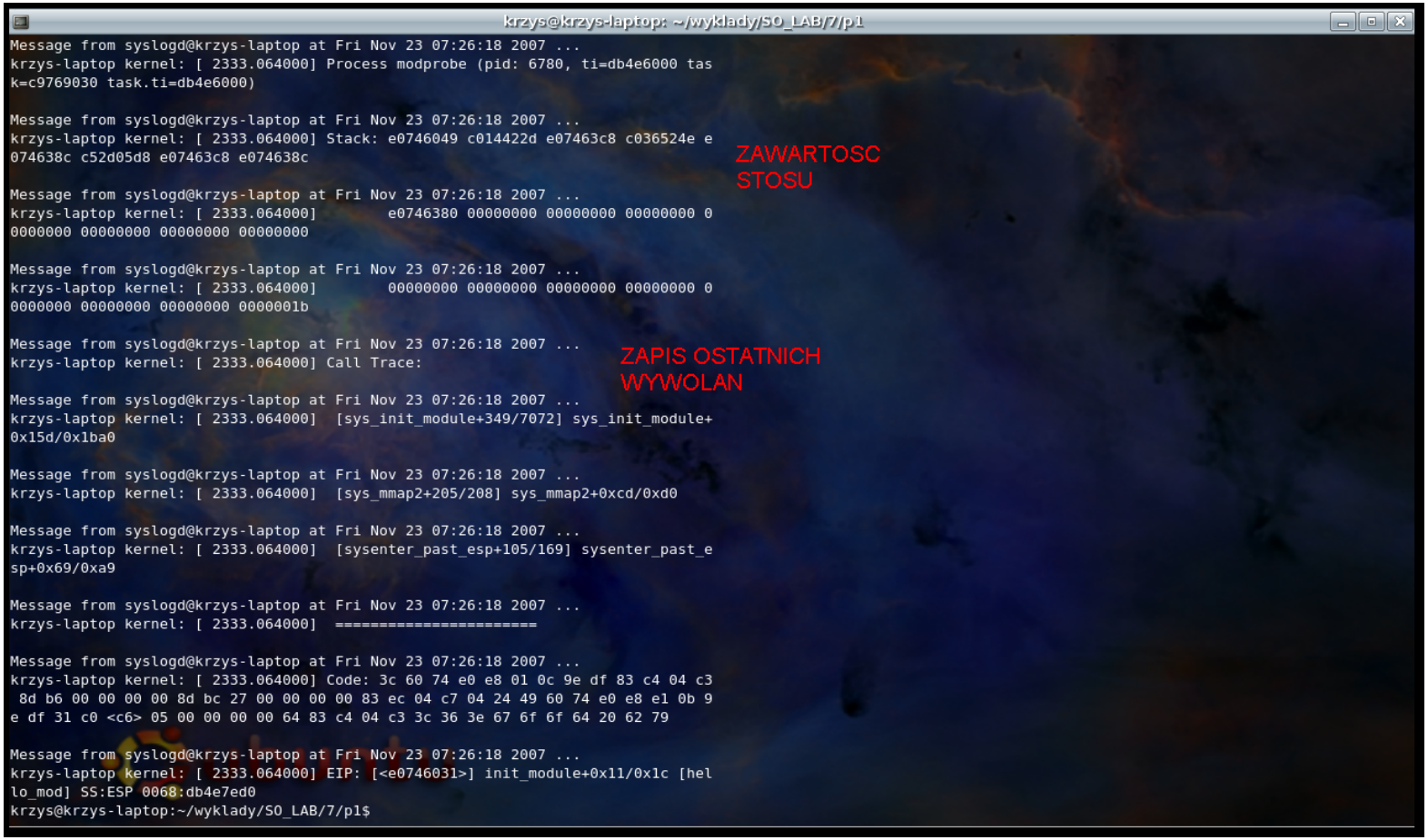
Co pokazuje oops?

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

Gdzie znajdują się te informacje?



Gdzie znajdują się te informacje?



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");
}
```

KLOGD

Demon klog 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: fffffffea   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 c01356e6 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

KSYMOOPS

- 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

- 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

- 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

- 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, zImage bzImage)
5. Lokalizacja modułów - można podawać za pomocą opsji -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, zImage bzImage)
5. Lokalizacja modułów - można podawać za pomocą opsji -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, zImage bzImage)
5. Lokalizacja modułów - można podawać za pomocą opsji -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, zImage bzImage)
5. Lokalizacja modułów - można podawać za pomocą opsji -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, zImage bzImage)
5. Lokalizacja modułów - można podawać za pomocą opsji -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, zImage bzImage)
5. Lokalizacja modułów - można podawać za pomocą opsji -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 $\geqslant 0$) - ile ukośnikow usunąć ze ścieżki dostępnej w łacie
  2. porównane zostają katalogi plików - oryginalnego i zmodyfikowanego

Krótko o łatach - c.d.

Przykladowy poczatkowy 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 łaty na przykładzie KDB

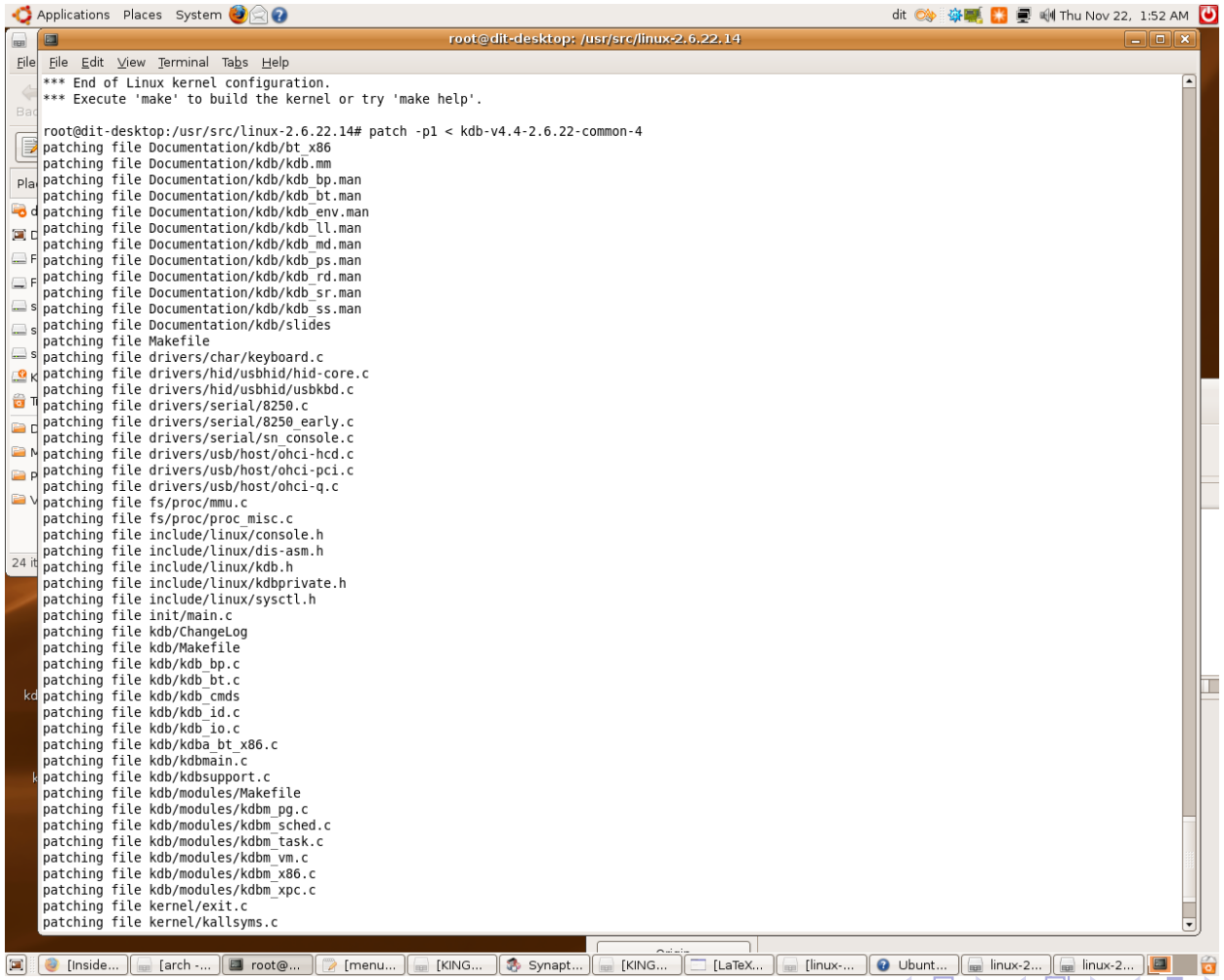
- pobieramy plik (`http://oss.sgi.com/projects/kdb/`)
- rozpakowujemy plik do katalogu ze źródłami Linuksa
- nakładamy patch
- konfigurujemy i kompilujemy jądro (za chwilę więcej o opcjach konfiguracji)

Oto najważniejsze kroki:

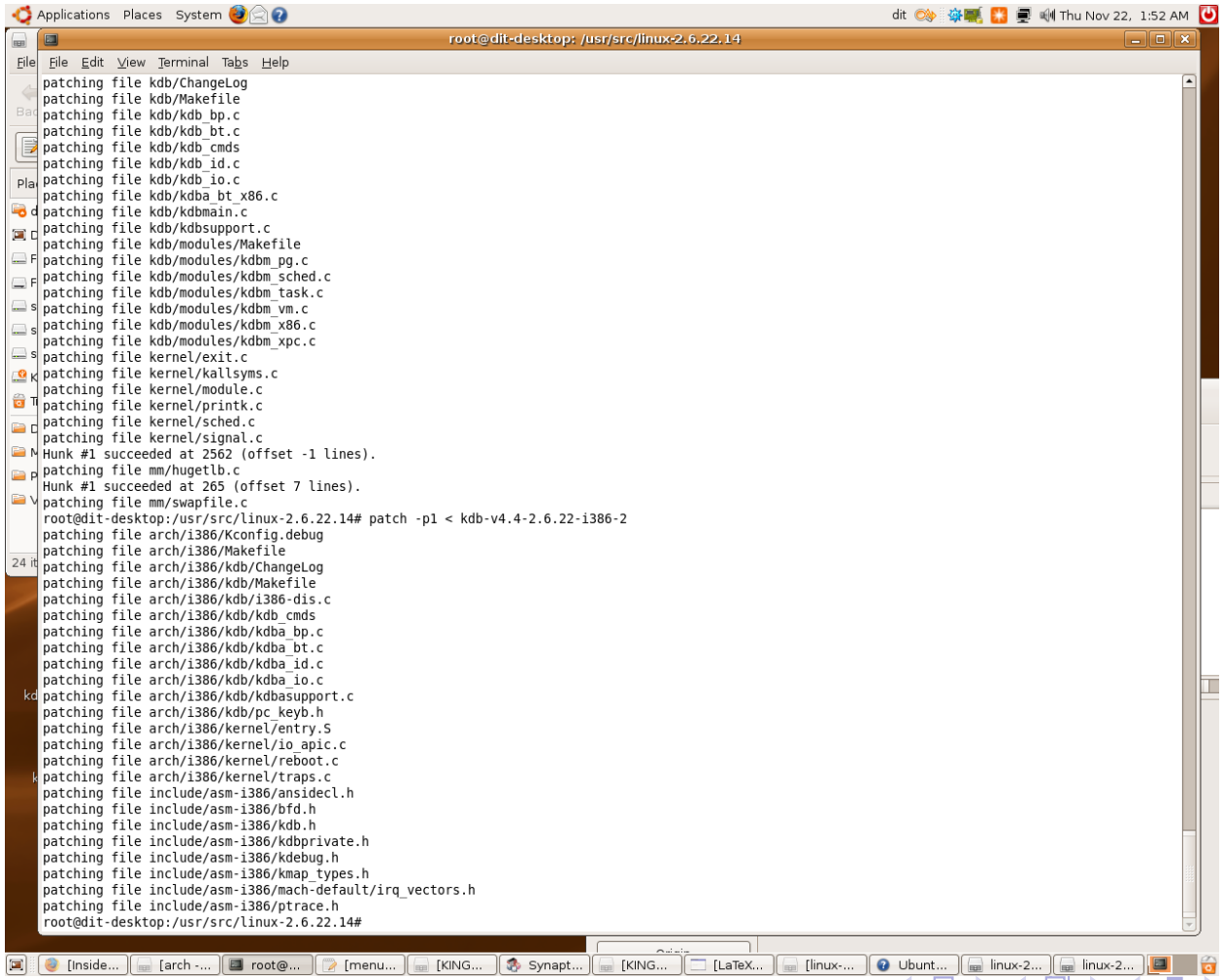
```
$ cd linux
$ patch -p1 < kdb-xxx  %nakladamy patch
$ make menuconfig      %interaktywna konfiguracja jadra
```

A teraz wersja (głównie) obrazkowa

Instalujemy patch ogólny - common



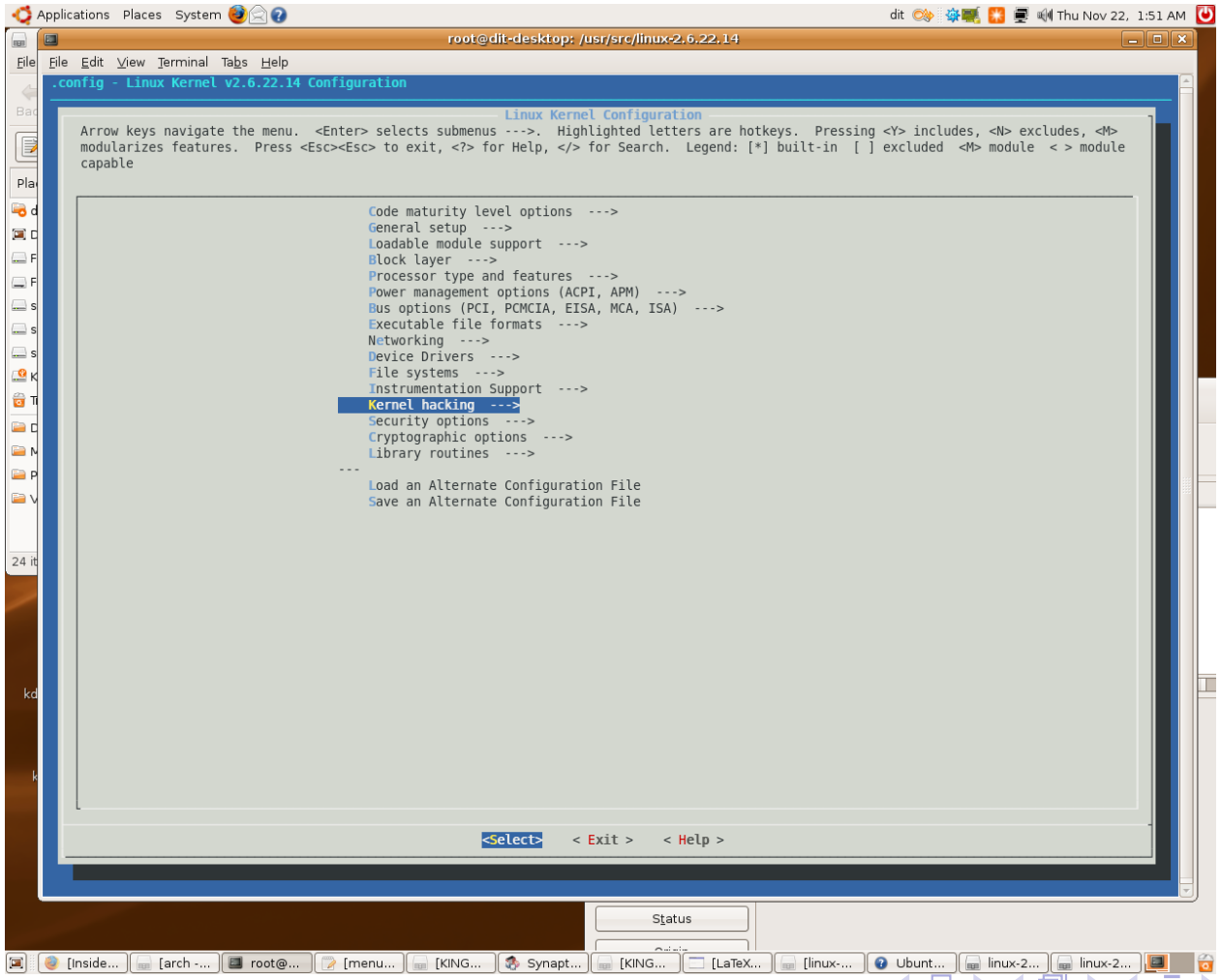
Instalujemy patch dla odpowiedniej architektury - tu: i386



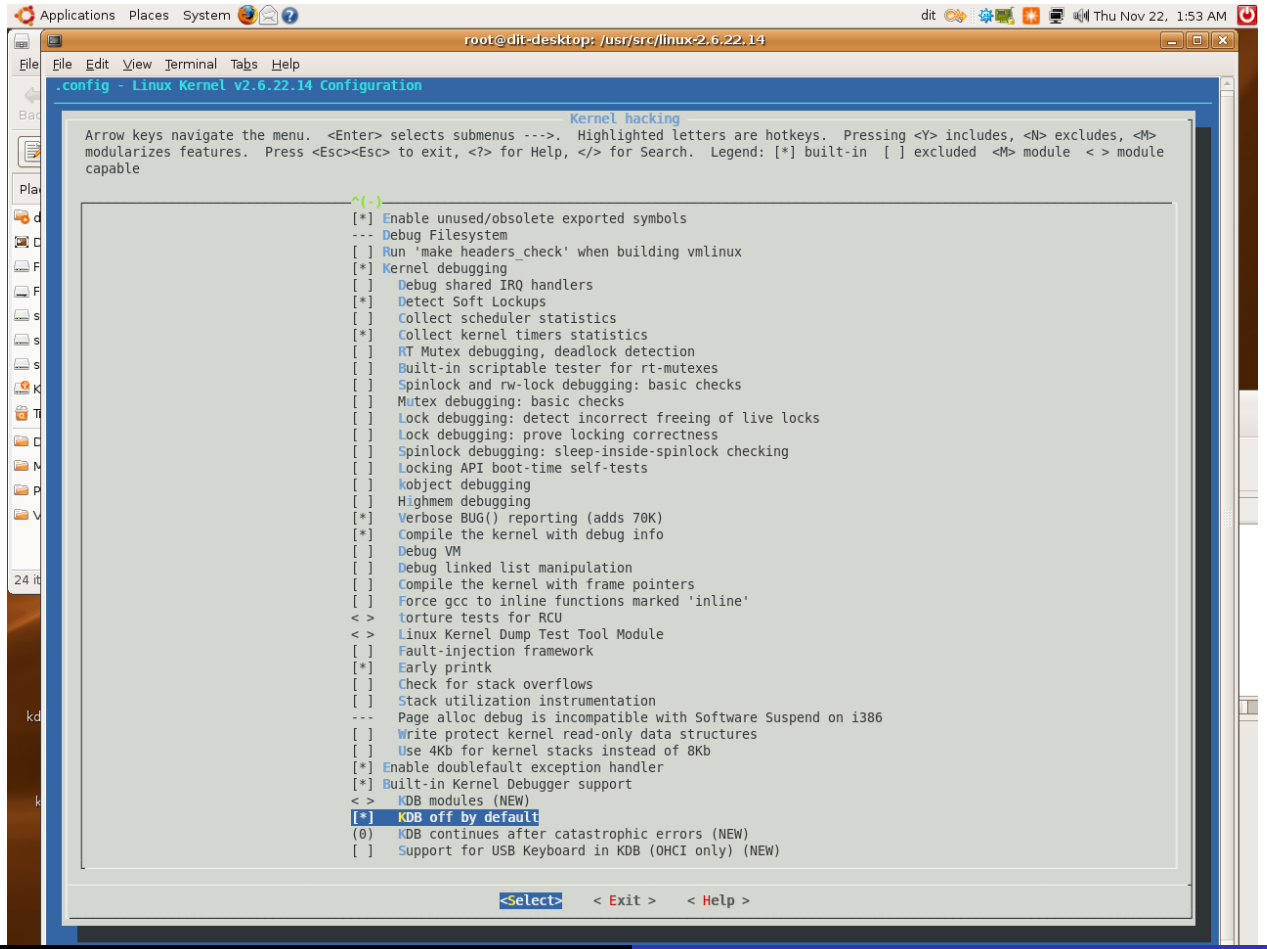
Wybrane opcje konfiguracji jądra

- 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

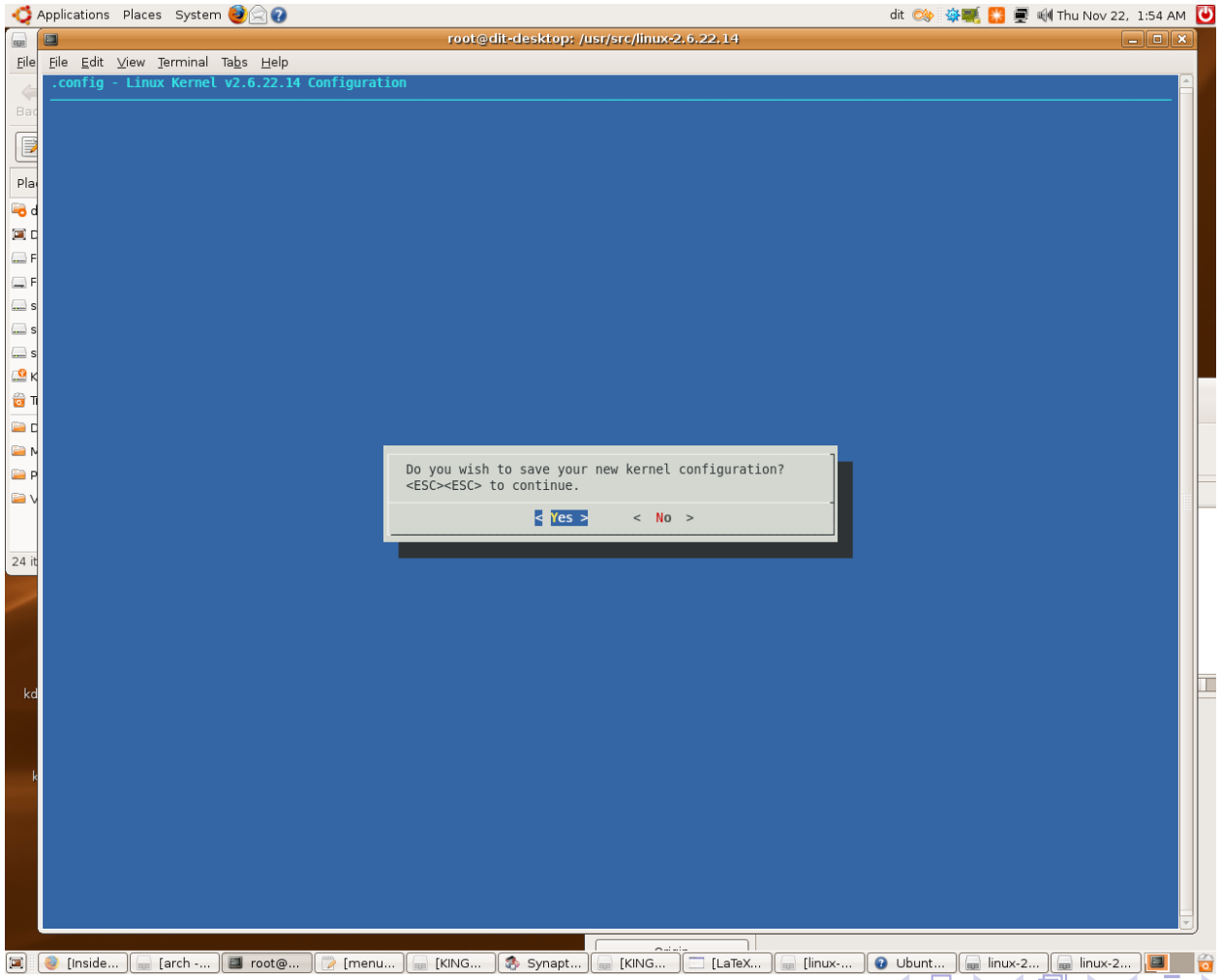
Po wpisaniu make menuconfig...



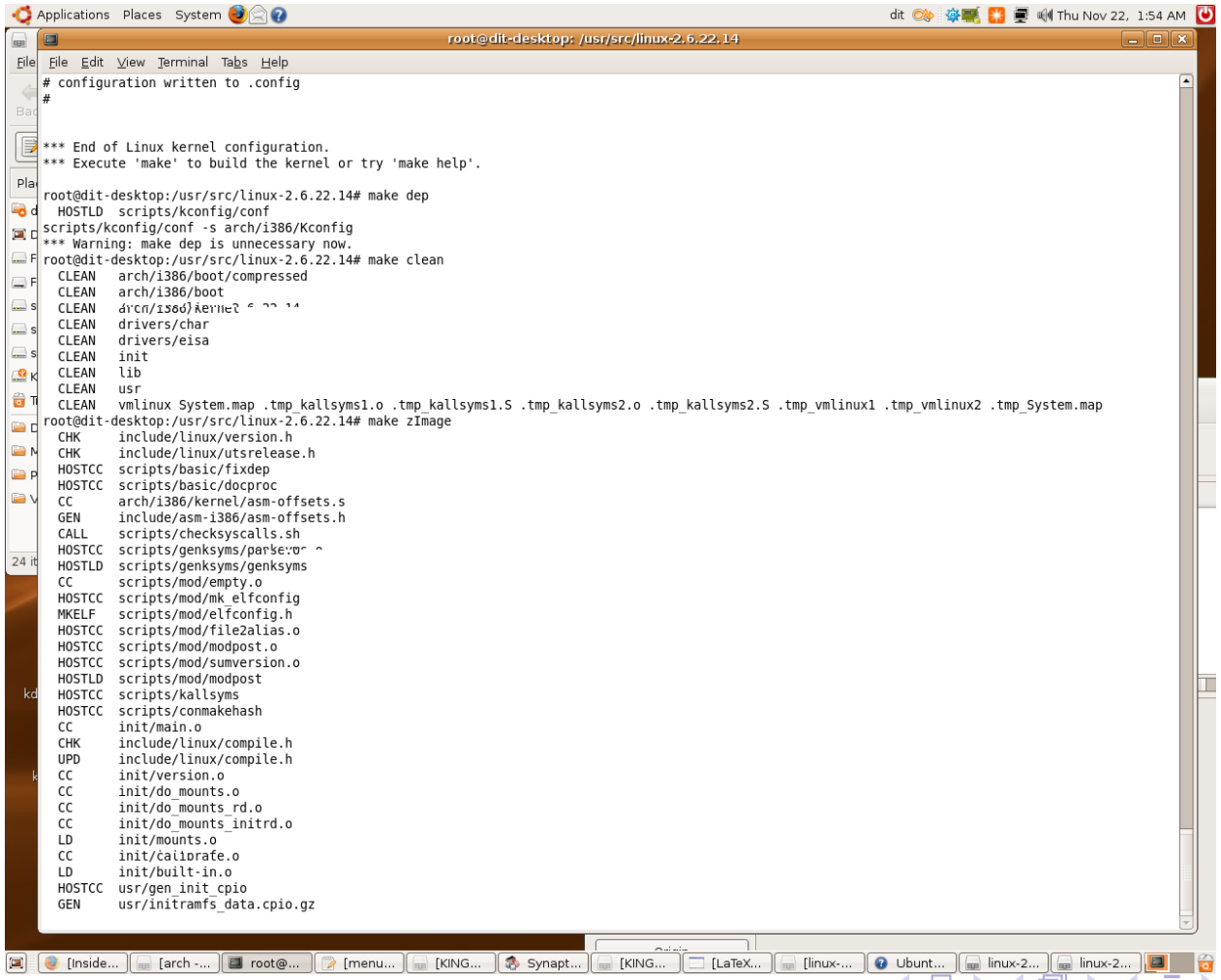
... wybieramy odpowiednie opcje dla debugowania (nie zapominamy o opcjach dla systemów plików itp.)



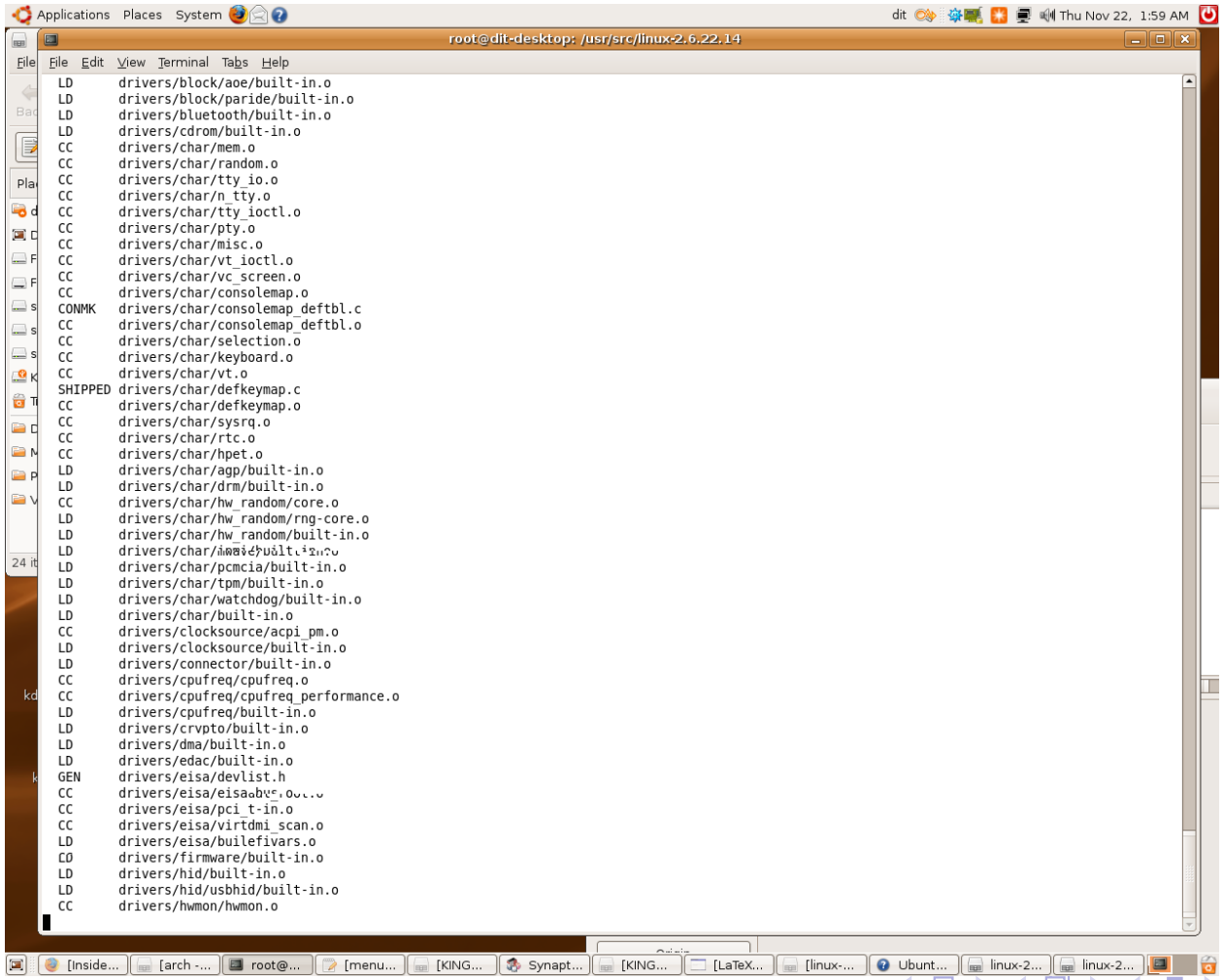
Zatwierdzamy opcje konfiguracji...



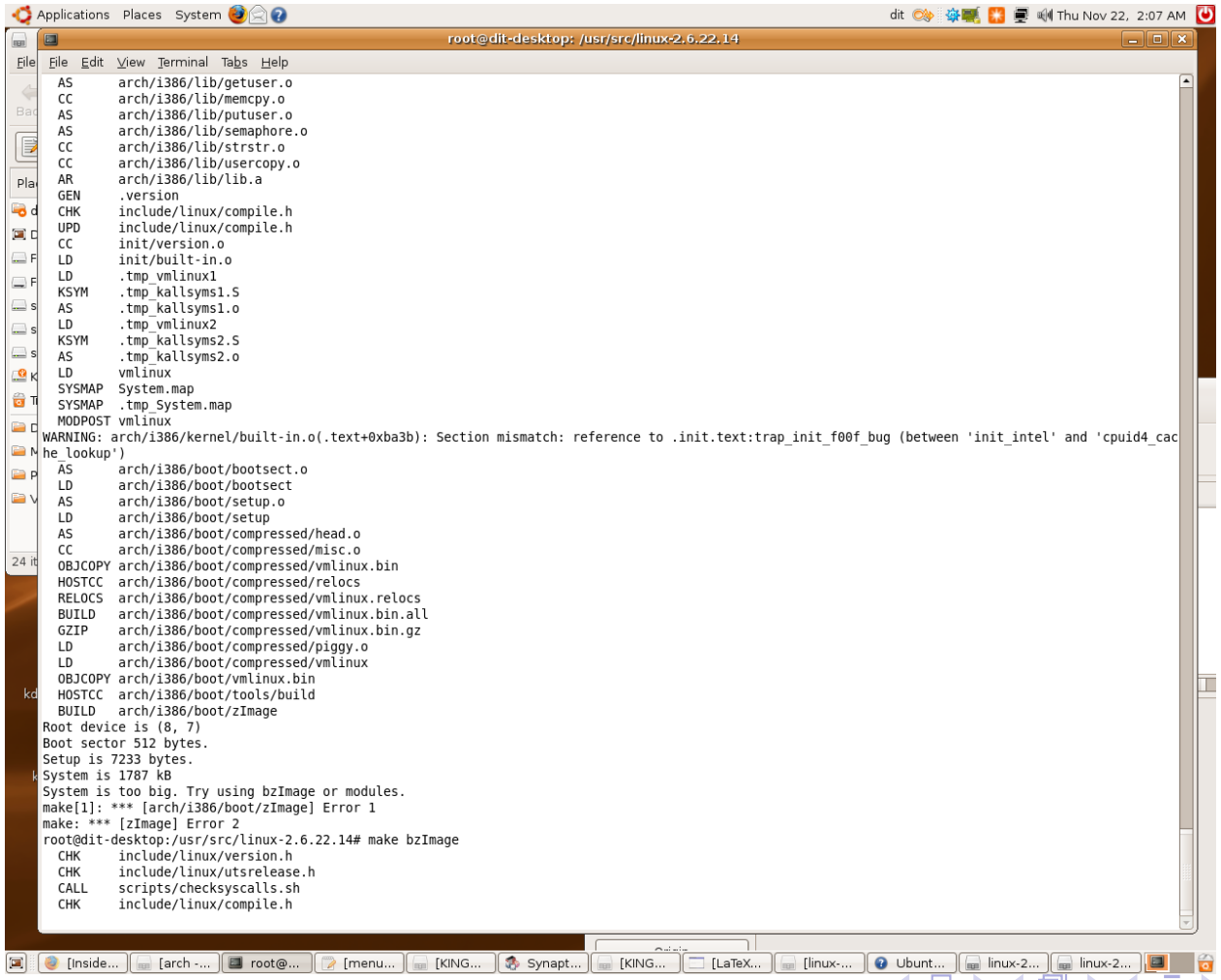
... i kompilujemy jądro



(co może trochę potrwać)



Na końcu używamy jeszcze *make bzImage*



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 mozna 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`

KGBD

- 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 szeregowe
   - debuggowanie odbywa się w gdb

KGBD

- 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 szeregowe
- debuggowanie odbywa się w gdb

KGBD

- 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 szeregowe
- debuggowanie odbywa się w gdb

KGBD

- 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 szeregowe
   - debuggowanie odbywa się w gdb

KGBD

- 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 szeregowe
   - debuggowanie odbywa się w gdb

KGBD

- 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 szeregowe
   - debuggowanie odbywa się w gdb

KGBD

- 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 szeregowe
- debuggowanie odbywa się w gdb

KGBD

- 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 szeregowe
   - debuggowanie odbywa się w gdb

KGBD

- 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 szeregowe
   - debuggowanie odbywa się w gdb

KGBD

- 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

KGBD

- 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

KGBD

- 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

KGBD

- 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

KGBD

- 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

KGBD

- 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

KGBD

aterm

```
panic: bremfree: removing a buffer not on a queue
panic messages:
---
panic: ffs_blkfree: freeing free block

syncing disks, buffers remaining... panic: bremfree: 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/PAV/modules/usr/src/sys/modules/linux/linux.ko.debug...done.
Loaded symbols for /usr/obj/usr/src/sys/PAV/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/PAV/modules/usr/src/sys/modules/acpi/acpi.ko.debug...done.
Loaded symbols for /usr/obj/usr/src/sys/PAV/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 bremfreel (bp=0xc7802088) at /usr/src/sys/kern/vfs_bio.c:648
#4  0xc025fca5 in bremfree (bp=0x0) at /usr/src/sys/kern/vfs_bio.c:630
#5  0xc02690e8 in vop_stdfsync (ap=0xd5d339f8) at /usr/src/sys/kern/vfs_default.c:759
#6  0xc01ebc00 in spec_fsync (ap=0xd5d339f8) at /usr/src/sys/fs/specfs/spec_vnops.c:418
#7  0xc01eb298 in spec_vnoperate (ap=0x0) at /usr/src/sys/fs/specfs/spec_vnops.c:123
#8  0xc02fd24d in ffs_sync (mp=0xc26eee00, 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_syscalls.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 0xc02e51ba in ffs_blkfree (fs=0xc2758000, devvp=0xc27165b4, bno=3703640, size=16384, inum=35768) at /usr/src/sys/ufs/ffs/ffs_alloc.c:1768
#13 0xc02f51af in indir_trunc (freeblks=0xc2de2500, dbn=24882976, level=0, lbn=12, countp=0xd5d33c10)
    at /usr/src/sys/ufs/ffs/ffs_softdep.c:2615
#14 0xc02f4c45 in handle_workitem_freeblocks (freeblks=0xc2de2500, flags=0) at /usr/src/sys/ufs/ffs/ffs_softdep.c:2480
#15 0xc02f1daa in process_worklist_item (matchmnt=0x0, flags=0) at /usr/src/sys/ufs/ffs/ffs_softdep.c:756
#16 0xc02f1ad0 in softdep_process_worklist (matchmnt=0x0) at /usr/src/sys/ufs/ffs/ffs_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
```

KGBD

- źródła jądra (*www.kernel.org/pub/linux/kernel*) - szczegolnie 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*) - szczegolnie 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

KGBD

- źródła jądra (*www.kernel.org/pub/linux/kernel*) - szczegolnie 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*) - szczegolnie 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

KGBD

- źródła jądra (*www.kernel.org/pub/linux/kernel*) - szczegolnie 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*) - szczegolnie 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

KGBD

- źródła jądra (*www.kernel.org/pub/linux/kernel*) - szczegolnie 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*) - szczegolnie 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

KGBD

- źródła jądra (*www.kernel.org/pub/linux/kernel*) - szczegolnie 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*) - szczegolnie 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

KGBD

- instalacja (*oslab.info/index.php/Misc/KGDB*) - pod VMware
- gdb (*safari.phptr.com/*0131492470*/ch*03*#ch*03) - poradnik
- gdbmod (*kgdb.linsyssoft.com/downloads/gdbmod* − 2.4.*bz*2) - Zmodyfikowane gdb dające możliwość debugowania modułów
- sDebug (*www.linuxjournal.com/article/*4525) modułów za pomocą KGDB

KGBD

- instalacja (*oslab.info/index.php/Misc/KGDB*) - pod VMware
- gdb (*safari.phptr.com*/0131492470/*ch*03#*ch*03) - poradnik
- gdbmod (*kgdb.linsyssoft.com/downloads/gdbmod* − 2.4.*bz*2) - Zmodyfikowane gdb dające możliwość debugowania modułów
- sDebug (*www.linuxjournal.com/article*/4525) modułów za pomocą KGDB

KGBD

- instalacja (*oslab.info/index.php/Misc/KGDB*) - pod VMware
- gdb (*safari.phptr.com*/0131492470/*ch*03#*ch*03) - poradnik
- gdbmod (*kgdb.linsyssoft.com/downloads/gdbmod* – 2.4.*bz*2) - Zmodyfikowane gdb dające możliwość debugowania modułów
- sDebug (*www.linuxjournal.com/article*/4525) modułów za pomocą KGDB

KGBD

- instalacja (*oslab.info/index.php/Misc/KGDB*) - pod VMware
- gdb (*safari.phptr.com*/0131492470/*ch*03#*ch*03) - poradnik
- gdbmod (*kgdb.linsyssoft.com/downloads/gdbmod* – 2.4.*bz*2) - 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 łatka
- w wersjach jądra >= 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