Laborator 2 - Module kernel

Obiectivele laboratorului

- crearea unor module simple pentru Linux şi pentru Windows
- descrierea procesului de compilare a surselor unui modul
- prezentarea modului în care un modul poate fi adăugat în kernel
- prezentarea metodelor de depanare a modulelor kernel

Cuvinte cheie

- built-in, loadable
- make, kbuild
- insmod, rmmod
- printk
- objdump, addr2line
- netconsole
- driver load/unload
- DbgPrint

Materiale ajutătoare

- Slide-uri de suport pentru laborator
- SO2 Reference Card

Noţiuni generale

Un kernel monolitic, deşi mai rapid decât un microkernel, are dezavantajul lipsei de modularitate şi extensibilitate. La kernel-ele monolitice moderne, acest lucru a fost rezolvat prin utilizarea de module de kernel. Un modul de kernel (sau modul de kernel încărcabil) este un fişier obiect care conţine cod ce poate extinde funcţionalitatea kernel-ului în timp real (este încărcat la nevoie); când un modul de kernel nu mai este necesar, acesta poate fi descărcat. Cea mai mare parte a driver-elor de dispozitiv (device drivers) sunt utilizate în forma de module de kernel.

Module kernel în Linux

Pentru dezvoltarea de device drivere în Linux se recomandă descărcarea surselor nucleului, configurarea şi compilarea acestora, iar apoi instalarea versiunii compilate pe maşina de test/dezvoltare. Este recomandat sa se folosească imaginea vmware pe care o găsiţi aici.

Un exemplu de modul kernel

În cele ce urmează prezentăm un exemplu foarte simplu de modul kernel. La încărcarea în kernel acesta va genera mesajul "Hi". La descărcarea modulului din kernel se va genera mesajul "Bye".

hello_lin.c

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

MODULE_DESCRIPTION("My kernel module");
MODULE_AUTHOR("Me");
MODULE_LICENSE("GPL");

static int dummy_init(void)
{
        printk( KERN_DEBUG "Hi\n" );
```

```
        return 0;
}

static void dummy_exit(void)
{
        printk( KERN_DEBUG "Bye\n" );
}

module_init(dummy_init);
module_exit(dummy_exit);
```

Mesajele generate nu vor fi afişate la consolă ci vor fi salvate într-o zonă de memorie special rezervată pentru acest lucru, de unde vor fi extrase de către daemonul de loguri (*syslog*). Pentru a afişa mesajele de kernel puteţi să folosiţi comanda *dmesg* sau să inspectaţi logurile:

```
# cat /var/log/syslog | tail -2
Feb 20 13:57:38 asgard kernel: Hi
Feb 20 13:57:43 asgard kernel: Bye

# dmesg | tail -2
Hi
Bye
```

Compilarea modulelor de kernel

Compilarea unui modul kernel diferă de compilarea unui program obişnuit. În primul rând, trebuie folosite alte headere. De asemenea, modulul nu trebuie legat de biblioteci. Şi, nu în ultimul rând, modulul trebuie compilat cu aceleaşi opţiuni ca şi nucleul în care vom încărca modulul. Din aceste motive există o metodă standard de compilare (kbuild). Această metodă necesită folosirea a două fişiere: un fişier `Makefile` şi un fişier `Kbuild`.

În continuare este prezentat un exemplu de fişier Makefile

```
KDIR=/lib/modules/`uname -r`/build

kbuild:
        make -C $(KDIR) M=`pwd`

clean:
        make -C $(KDIR) M=`pwd` clean
```

şi exemplul de fişier Kbuild asociat, folosit la compilarea unui modul:

```
EXTRA_CFLAGS = -g

obj-m        = modul.o
```

După cum se observă, invocarea make pe fişierul Makefile din exemplul prezentat va duce la invocarea make în directorul cu sursele kernelului (*/lib/modules/`uname -r`/build*) şi cu referinţă la directorul curent (*M=`pwd`*). Acest proces duce în cele din urmă la citirea fişierului Kbuild din directorul curent şi la compilarea modulului conform instrucţiunilor din acest fişier.

Un fişier Kbuild conţine una sau mai multe directive pentru compilarea unui modul de kernel. Cel mai simplu exemplu de astfel de directivă este *obj-m = modul.o*. În urma acestei directive va fi creat un modul de kernel *modul.ko* (ko - kernel object), plecând de la fişierul *modul.o*. *modul.o* va fi creat plecând de la *modul.c* sau *modul.S*. Toate aceste fişiere se găsesc în directorul în care se află şi Kbuild.

Un exemplu de fisier Kbuild care foloseste mai multe sub-module este prezentat mai jos:

```
EXTRA_CFLAGS = -g

obj-m   = supermodul.o
supermodul-y =  modul-a.o modul-b.o
```

Pentru exemplul de mai sus, paşii efectuaţi la compilare sunt:

- se vor compila sursele modul-a.c, modul-b.c, rezultând fişierele obiect modul-a.o şi modul-b.o
- modul-a.o şi modul-b.o vor fi apoi legate în supermodul.o
- din supermodul.o se va crea modulul supermodul.ko

Pentru mai multe detalii consultaţi fişierul makefiles.txt şi fişierul modules.txt din cadrul surselor kernel-ului.

Încărcarea/descarcarea unui modul de kernel

Pentru a încărca un modul în kernel se foloseşte utilitarul `insmod`. Acest utilitar primeşte ca parametru calea către fişierul .ko în care a fost compilat şi link-editat modulul. Descărcarea modulului din kernel se face cu ajutorul comenzi `rmmod`, care primeste ca parametru numele modulului.

```
# insmod modul.ko

# rmmod modul
```

La încărcarea modulului în kernel va fi executată rutina specificată ca parametru macroului *module_init*. Similar, la descărcarea modulului va fi executată rutina specificată ca parametru macroului *module_exit*.

Un exemplu complet de compilare si incarcare/descarcare modul este prezentat in continuare:

```
faust:~/lab-01/modul-lin# ls
Kbuild  Makefile  modul.c

faust:~/lab-01/modul-lin# make
make -C /lib/modules/`uname -r`/build M=`pwd`
make[1]: Entering directory `/usr/src/linux-2.6.28.4'
  LD      /root/lab-01/modul-lin/built-in.o
  CC [M]  /root/lab-01/modul-lin/modul.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/lab-01/modul-lin/modul.mod.o
  LD [M]  /root/lab-01/modul-lin/modul.ko
make[1]: Leaving directory `/usr/src/linux-2.6.28.4'

faust:~/lab-01/modul-lin# ls
built-in.o  Kbuild  Makefile  modul.c  Module.markers
modules.order  Module.symvers  modul.ko  modul.mod.c
modul.mod.o  modul.o

faust:~/lab-01/modul-lin# insmod modul.ko

faust:~/lab-01/modul-lin# dmesg | tail -1
Hi

faust:~/lab-01/modul-lin# rmmod modul

faust:~/lab-01/modul-lin# dmesg | tail -2
Hi
Bye
```

Informaţii despre modulele încărcate în kernel se pot afla cu ajutorul comenzii `lsmod`, prin inspectarea fişierului `/proc/modules` sau a directorului `/sys/module`.

Depanarea unui modul de kernel

Depanarea unui modul de kernel este mult mai complicată decât depanarea unui program obişnuit. În primul rând, o greşeală într-un modul kernel poate duce la blocarea întregului sistem. Depanarea este din această cauză mult încetinită. Pentru a evita secvenţele de reboot se recomandă instalarea unei maşini virtuale şi utilizarea snapshot-urilor.

Atunci când un modul ce conţine bug-uri este inserat în kernel, se va genera în cele din urmă un kernel oops. Un kernel oops reprezintă o operaţie invalidă detectată de nucleu şi poate fi generată doar de către acesta. Pentru o versiune stabilă de kernel, aceasta înseamnă, aproape sigur, că modulul contine un bug. Dupa apariţia oops-ului, kernelul va continua să funcţioneze.

Foarte importantă la apariţia unui kernel oops este salvarea mesajului generat. După cum s-a precizat anterior, mesajele generate de kernel se salvează în loguri şi pot fi afişate cu comanda `dmesg`. Pentru a fi siguri că nu se pierde un kernel oops se recomanda inserarea/testarea modulului kernel direct din consolă, sau verificarea periodică a mesajelor de kernel. De remarcat este faptul că un oops poate apărea din cauza unei erori de programare, dar şi a unei erori hardware.

În cazul în care apare o eroare fatala după care sistemul nu mai poate reveni la o stare stabilă, se generează un kernel panic.

Printk Debugging

:`The two oldest and most useful debugging aids are Your brain and Printf`

Pentru depanare cel mai adesea se foloseşte un mijloc primitiv, dar destul de eficient: afişarea de mesaje (printk

debugging). Deşi se poate folosi şi un debugger, în general acesta nu este foarte folositor: bug-urile simple (variabile neiniţializate, probleme la gestiunea memoriei etc.) pot fi uşor localizate cu ajutorul mesajelor de control si mesajului de oops decodificat de către kernel.

La bugurile mai complexe, nici chiar un debugger nu ne poate ajuta prea mult dacă nu se înţelege foarte bine structura sistemului de operare. La depanarea unui modul de kernel intervin o mulţime de necunoscute în ecuaţie: contexte multiple (avem mai multe procese şi threaduri ce rulează la un moment dat), întreruperi, memorie virtuală, etc.

Pentru afişarea mesajelor din kernel către user space se poate folosi `printk` [1]. Acesta este similar ca funcţionalitate lui `printf`; singura diferenţă constă în faptul că mesajul transmis se poate prefixa cu un şir de forma "", unde n indica nivelul (loglevel-ul) erorii şi are valori între 0 şi 7. În loc de "", nivelele pot fi codificate şi prin constante simbolice:

- `KERN_EMERG` - n = 0
- `KERN_ALERT` - n = 1
- `KERN_CRIT` - n = 2
- `KERN_ERR` - n = 3
- `KERN_WARNING` - n = 4
- `KERN_NOTICE` - n = 5
- `KERN_INFO` - n = 6
- `KERN_DEBUG` - n = 7

Definiţiile tuturor loglevel-urilor se găsesc în `linux/kernel.h`. Practic, aceste loglevel-uri sunt utilizate de sistem pentru a ruta mesajele trimise către diverse output-uri: consolă, fişiere log din `/var/log`, etc.

Pentru a afişa mesajele trimise cu `printk` în user space, trebuie ca nivelul folosit la apelul `printk` să fie mai mare sau egal (în sensul importanţei, de fapt mai mic sau egal în realitate, adică spre "<1>" KERN_ALERT) decât variabila de sistem `console_loglevel` [2].

Mesajele redirectate la consolă pot fi utile pentru a vizualiza rapid efectul execuţiei codului inserat în kernel, însa ele nu mai sunt aşa folositoare în cazul în care în kernel apare o eroare irecuperabilă, iar sistemul îngheaţă. În acest caz, trebuiesc consultate log-urile sistemului, deoarece în ele se păstrează informaţiile între restart-uri ale sistemului. Acestea se găsesc în `/var/log` [3] şi sunt fişiere text, populate cu informaţie de `syslogd` şi `klogd` pe parcursul rulării kernelului. `syslogd` şi `klogd` preiau la rândul lor informaţiile din sistemul virtual de fişiere montat în `/proc`. În principiu, cu `syslogd` şi `klogd` pornite, toate mesajele venite de la kernel vor ajunge în `/var/log/kern.log`.

O variantă mai simplă pentru etapa de debugging este folosirea fişierului `/var/log/debug`. Acesta este populat numai cu mesajele `printk` venite de la kernel cu loglevel-ul `KERN_DEBUG`.

Având în vedere faptul că un kernel de producţie (similar celui pe care probabil îl rulăm şi noi :P) conţine doar cod de release, modulul nostru este printre puţinele care trimit mesaje prefixate cu `KERN_DEBUG`. În acest fel, putem naviga uşor prin informaţiile din `/var/log/debug`, găsind mesajele corespunzatoare unei sesiuni de debug pentru modulul nostru.

Un exemplu de utilizare ar fi urmatorul:

- se curăţă fişierul debug de informaţiile anterioare (sau, eventual, se face un backup):

```
echo "New debug session" > /var/log/debug
```

- se rulează testul/testele
- dacă nu apare o eroare critică, care să cauzeze un kernel panic, se consultă output-ul
- dacă apare o eroare critică, iar maşina nu mai raspunde decat la restart, atunci se reporneste sistemul, şi se consultă `/var/log/debug`

Formatul mesajelor trebuie, evident, să conţină toate informaţiile de interes pentru a depista eroarea, însă inserarea în cod a "printk-urilor" care să ofere informaţii detaliate poate fi la fel de time-consuming ca şi scrierea codului pentru rezolvarea problemei. De aceea se face de obicei un trade-off între completitudinea mesajelor de debugging afişate folosind printk şi timpul necesar pentru inserarea acestor mesaje în text.

O varianta foarte simplă, puţin costisitoare din punctul de vedere al timpului necesar pentru inserarea printk-urilor, şi care oferă posibilitatea analizării fluxului de instrucţiuni în cazul testelor, este cea a folosirii constantelor predefinite `__FILE__`, `__LINE__` şi `__func__`:

- `__FILE__` este înlocuită, la compilare, de către compilator, cu numele fişierului sursă în care se găseşte la momentul respectiv.
- `__LINE__` este înlocuită, la compilare, de către compilator, cu numărul liniei pe care se găseşte instrucţiunea curentă în

cadrul fişierului sursă curent.
- `__func__` / `__FUNCTION__` este înlocuită, la compilare, de către compilator, cu numele funcţiei în care se găseşte instrucţiunea curentă.

Observatie: `__LINE__` şi `__FILE__` fac parte din specificaţiile standardului ANSI C; `__func__` face parte din specificaţiile C99; `__FUNCTION__` reprezinta o extensie GNU C şi nu este portabilă; însă, având în vedere că scriem cod pentru kernelul de Linux, o putem folosi fără probleme.

Se poate folosi în acest caz următoarea macrodefiniţie:

```
#define PRINT_DEBUG    \
        printk(KERN_DEBUG "[%s]:FUNC:%s:LINE:%d\n", __FILE__ , __FUNCTION__ , __LINE__)
```

Apoi, în fiecare punct în care dorim să vedem dacă este "atins" în execuţie, inserăm PRINT_DEBUG; aceasta este o modalitate simplă şi rapidă, şi poate da roade analizând cu atenţie output-ul oferit.

Pentru a vedea mesajele afişate cu `printk`, dar care nu apar la consolă, se foloseşte comanda dmesg.

Pentru a şterge toate mesajele anterioare dintr-un fişier de log, se rulează `cat /dev/null > /var/log/debug`. Pentru a şterge mesajele afişate de comanda `dmesg`, se foloseşte `dmesg -c`.

Exemplu de modul de kernel pentru depanare

Fie următorul modul de kernel care conţine un bug pentru generarea unui oops:

debug_lin.c

```
/*
 * Oops generating kernel module
 */

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>

MODULE_DESCRIPTION ("Oops");
MODULE_LICENSE ("GPL");
MODULE_AUTHOR ("PSO");

#define OP_READ         0
#define OP_WRITE        1
#define OP_OOPS         OP_WRITE

static int my_oops_init (void)
{
        int *a;

        a = (int *) 0x00001234;
#if OP_OOPS == OP_WRITE
        *a = 3;
#elif OP_OOPS == OP_READ
        printk (KERN_ALERT "value = %d\n", *a);
#else
#error "Unknown op for oops!"
#endif

        return 0;
}

static void my_oops_exit (void)
{
}

module_init (my_oops_init);
module_exit (my_oops_exit);
```

Inserarea acestui modul in kernel va genera un oops:

```
faust:~/lab-01/modul-oops# insmod oops.ko
[...]

faust:~/lab-01/modul-oops# dmesg | tail -32
BUG: unable to handle kernel paging request at 00001234
IP: [<c89d4005>] my_oops_init+0x5/0x20 [oops]
```

```
*pde = 00000000
Oops: 0002 [#1] PREEMPT DEBUG_PAGEALLOC
last sysfs file: /sys/devices/virtual/net/lo/operstate
Modules linked in: oops(+) netconsole ide_cd_mod pcnet32 crc32 cdrom [last unloaded: modul]

Pid: 4157, comm: insmod Not tainted (2.6.28.4 #2) VMware Virtual Platform
EIP: 0060:[<c89d4005>] EFLAGS: 00010246 CPU: 0
EIP is at my_oops_init+0x5/0x20 [oops]
EAX: 00000000 EBX: fffffffc ECX: c89d4300 EDX: 00000001
ESI: c89d4000 EDI: 00000000 EBP: c5799e24 ESP: c5799e24
 DS: 007b ES: 007b FS: 0000 GS: 0033 SS: 0068
Process insmod (pid: 4157, ti=c5799000 task=c665c780 task.ti=c5799000)
Stack:
 c5799f8c c010102d c72b51d8 0000000c c5799e58 c01708e4 00000124 00000000
 c89d4300 c5799e58 c724f448 00000001 c89d4300 c5799e60 c0170981 c5799f8c
 c014b698 00000000 00000000 c5799f78 c5799f20 00000500 c665cb00 c89d4300
Call Trace:
 [<c010102d>] ? _stext+0x2d/0x170
 [<c01708e4>] ? __vunmap+0xa4/0xf0
 [<c0170981>] ? vfree+0x21/0x30
 [<c014b698>] ? load_module+0x19b8/0x1a40
 [<c035e965>] ? __mutex_unlock_slowpath+0xd5/0x140
 [<c0140da6>] ? trace_hardirqs_on_caller+0x106/0x150
 [<c014b7aa>] ? sys_init_module+0x8a/0x1b0
 [<c0140da6>] ? trace_hardirqs_on_caller+0x106/0x150
 [<c0240a08>] ? trace_hardirqs_on_thunk+0xc/0x10
 [<c0103407>] ? sysenter_do_call+0x12/0x43
Code: <c7> 05 34 12 00 00 03 00 00 00 5d c3 eb 0d 90 90 90 90 90 90 90 90
EIP: [<c89d4005>] my_oops_init+0x5/0x20 [oops] SS:ESP 0068:c5799e24
---[ end trace 2981ce73ae801363 ]---
```

Deşi relativ criptic, mesajul oferit de kernel la apariţia unui oops oferă informaţii preţioase despre eroarea aparută. Prima linie:

```
BUG: unable to handle kernel paging request at 00001234
IP: [<c89d4005>] my_oops_init+0x5/0x20 [oops]
```

ne spune cauza şi adresa instrucţiunii care a generat eroarea. În cazul de faţă este vorba de un acces invalid la memorie.

Linia următoare:

```
Oops: 0002 [#1] PREEMPT DEBUG_PAGEALLOC
```

ne spune că este vorba de primul oops (`#1`). Acest lucru este important în contextul în care un oops poate duce la apariţia altor oops-uri. De obicei doar primul oops este relevant. Mai mult, codul oops-ului (`0002`) oferă informaţii despre tipul erorii (în memory manager [fault.c](#)):

```
* bit 0 == 0 means no page found, 1 means protection fault
* bit 1 == 0 means read, 1 means write
* bit 2 == 0 means kernel, 1 means user-mode
```

În cazul de faţă avem un acces de tip scriere care a generat oops-ul (bitul 1 este 1).

În continuare se afişează un dump al registrelor. Se decodifică valoarea EIP (instruction pointer) şi se observă că bug-ul a aparut în cadrul funcţiei my_oops_init cu un offset de 5 octeţi (`EIP: [] my_oops_init+0x5`). Mesajul prezintă şi conţinutul stivei şi un backtrace al apelurilor de până atunci.

În cazul în care se generează un apel invalid de citire (`#define OP_OOPS OP_READ`), mesajul va fi asemănător. Va diferi codul de oops, care acum ar avea valoarea `0000`:

```
faust:~/lab-01/modul-oops# dmesg | tail -33
BUG: unable to handle kernel paging request at 00001234
IP: [<c89c3016>] my_oops_init+0x6/0x20 [oops]
*pde = 00000000
Oops: 0000 [#1] PREEMPT DEBUG_PAGEALLOC
last sysfs file: /sys/devices/virtual/net/lo/operstate
Modules linked in: oops(+) netconsole pcnet32 crc32 ide_cd_mod cdrom

Pid: 2754, comm: insmod Not tainted (2.6.28.4 #2) VMware Virtual Platform
EIP: 0060:[<c89c3016>] EFLAGS: 00010292 CPU: 0
EIP is at my_oops_init+0x6/0x20 [oops]
EAX: 00000000 EBX: fffffffc ECX: c89c3380 EDX: 00000001
ESI: c89c3010 EDI: 00000000 EBP: c57cbe24 ESP: c57cbe1c
 DS: 007b ES: 007b FS: 0000 GS: 0033 SS: 0068
Process insmod (pid: 2754, ti=c57cb000 task=c66ec780 task.ti=c57cb000)
Stack:
 c57cbe34 00000282 c57cbf8c c010102d c57b9280 0000000c c57cbe58 c01708e4
 00000124 00000000 c89c3380 c57cbe58 c5db1d38 00000001 c89c3380 c57cbe60
 c0170981 c57cbf8c c014b698 00000000 00000000 c57cbf78 c57cbf20 00000580
```

```
Call Trace:
 [<c010102d>] ? _stext+0x2d/0x170
 [<c01708e4>] ? __vunmap+0xa4/0xf0
 [<c0170981>] ? vfree+0x21/0x30
 [<c014b698>] ? load_module+0x19b8/0x1a40
 [<c035d083>] ? printk+0x0/0x1a
 [<c035e965>] ? __mutex_unlock_slowpath+0xd5/0x140
 [<c0140da6>] ? trace_hardirqs_on_caller+0x106/0x150
 [<c014b7aa>] ? sys_init_module+0x8a/0x1b0
 [<c0140da6>] ? trace_hardirqs_on_caller+0x106/0x150
 [<c0240a08>] ? trace_hardirqs_on_thunk+0xc/0x10
 [<c0103407>] ? sysenter_do_call+0x12/0x43
Code: <a1> 34 12 00 00 c7 04 24 54 30 9c c8 89 44 24 04 e8 58 a0 99 f7 31
EIP: [<c89c3016>] my_oops_init+0x6/0x20 [oops] SS:ESP 0068:c57cbe1c
---[ end trace 45eeb3d6ea8ff1ed ]---
```

objdump

Informaţii detaliate despre instrucţiunea care a generat oops-ul pot fi aflate folosind utilitarul `objdump` de inspecţie a unui cod obiect. Opţiunile utile de folosit sunt `-d` pentru dezasamblarea codului şi `-S` pentru intercalarea codului C în cod în limbaj de asamblare. Pentru o decodificare eficientă avem însă nevoie de adresa unde a fost încarcat modulul de kernel. Aceasta poate fi regasită în `/proc/modules`.

Prezentăm în continuare un exemplu de utilizare a objdump pe modulul de mai sus pentru a identifica instrucţiunea care a generat oops-ul:

```
faust:~/lab-01/modul-oops# cat /proc/modules
oops 1280 1 - Loading 0xc89d4000
netconsole 8352 0 - Live 0xc89ad000
pcnet32 33412 0 - Live 0xc895a000
ide_cd_mod 34952 0 - Live 0xc8903000
crc32 4224 1 pcnet32, Live 0xc888a000
cdrom 34848 1 ide_cd_mod, Live 0xc886d000

faust:~/lab-01/modul-oops# objdump -dS --adjust-vma=0xc89d4000 oops.ko

oops.ko:     file format elf32-i386


Disassembly of section .text:

c89d4000 <init_module>:
#define OP_READ         0
#define OP_WRITE        1
#define OP_OOPS         OP_WRITE

static int my_oops_init (void)
{
c89d4000:       55                      push   %ebp
#else
#error "Unknown op for oops!"
#endif

        return 0;
}
c89d4001:       31 c0                   xor    %eax,%eax
#define OP_READ         0
#define OP_WRITE        1
#define OP_OOPS         OP_WRITE

static int my_oops_init (void)
{
c89d4003:       89 e5                   mov    %esp,%ebp
        int *a;

        a = (int *) 0x00001234;
#if OP_OOPS == OP_WRITE
        *a = 3;
c89d4005:       c7 05 34 12 00 00 03    movl
c89d4005:       c7 05 34 12 00 00 03    movl
faust:~/lab-01/modul-oops# addr2line -e oops.ko 0x5
/root/lab-01/modul-oops/oops.c:23
x3,0x1234
x3,0x1234
c89d400c:       00 00 00
#else
#error "Unknown op for oops!"
```

```
#endif

return 0;
}
c89d400f:       5d                      pop    %ebp
c89d4010:       c3                      ret
c89d4011:       eb 0d                   jmp    c89c3020 <cleanup_module>
c89d4013:       90                      nop
c89d4014:       90                      nop
c89d4015:       90                      nop
c89d4016:       90                      nop
c89d4017:       90                      nop
c89d4018:       90                      nop
c89d4019:       90                      nop
c89d401a:       90                      nop
c89d401b:       90                      nop
c89d401c:       90                      nop
c89d401d:       90                      nop
c89d401e:       90                      nop
c89d401f:       90                      nop

c89d4020 <cleanup_module>:

static void my_oops_exit (void)
{
c89d4020:       55                      push   %ebp
c89d4021:       89 e5                   mov    %esp,%ebp
}
c89d4023:       5d                      pop    %ebp
c89d4024:       c3                      ret
c89d4025:       90                      nop
c89d4026:       90                      nop
c89d4027:       90                      nop
```

Se observă că instrucţiunea care a generat oops-ul (cea de la adresa `c89d4005` identificată anterior) este:

```
alice:~# modprobe netconsole\
> netconsole="6666@192.168.191.130/eth0,30000@192.168.191.1/00:50:56:c0:00:08"
```

adică exact cum era de aşteptat - stocarea valorii `3` la adresa `0x0001234`.

Fişierul `/proc/modules` este folosit pentru a afla adresa unde este încarcat un modul de kernel. Opţiunea `?adjust-vma` permite afişarea instrucţiunilor relativ la adresa `0xc89d4000`. Opţiunea `-l` afişează numărul fiecărei linii din codul sursă intercalat cu codul în limbaj de asamblare.

addr2line

O modalitate mai simplistă de a găsi codul care a generat un oops este de a folosi utilitarul addr2line:

```
bob:~# nc -l -p 30000 -u
```

unde `0x5` este valoarea contorului program (EIP = `c89d4005`) la care s-a generat kernel oops, minus adresa de bază a modulului (`0xc89c4000`), conform `/proc/modules`.

netconsole

Netconsole este un utilitar care permite logarea mesajelor de debug din kernel prin intermediul reţelei. Acest lucru este folositor atunci când sistemul de logging pe disk nu funcţionează, când nu sunt disponibile porturi seriale sau când terminalul nu răspunde la comenzi. Netconsole vine sub forma unui modul de kernel.

Pentru a funcţiona, acesta are nevoie de următorii parametri:

- portul,adresa IP şi numele interfeţei sursă ale staţiei pe care se face debug
- portul, adresa MAC şi adresa IP a maşinii către care vor fi trimise mesajele de debug

Aceşti parametri pot fi configuraţi atunci când modulul este inserat în kernel, sau, chiar în timp ce modulul este inserat dacă acesta a fost compilat cu opţiunea CONFIG_NETCONSOLE_DYNAMIC.

Un exemplu de configurare în momentul inserării este următorul:

```
#include <ntddk.h>

void DriverUnload(PDRIVER_OBJECT driver)
{
```

```
	DbgPrint("driver unload");
	return;
}

NTSTATUS DriverEntry(PDRIVER_OBJECT driver, PUNICODE_STRING registry)
{
	DbgPrint("driver entry");
	driver->DriverUnload = DriverUnload;
	return STATUS_SUCCESS;
}
```

Astfel, mesajele de debug de pe staţia ce are adresa 192.168.191.130 vor fi trimise pe interfaţa eth0, având ca port sursă 6666. Mesajele vor fi trimise către 192.168.191.1, ce are adresa MAC 00:50:56:c0:00:08 , pe portul 30000.

Mesajele pot fi ascultate pe staţia destinaţie folosind `netcat`:

```
#include <ntddk.h>
#include "eventlog.h"

void putk(PDRIVER_OBJECT driver, long error, const char *str)
{
	UNICODE_STRING ustr;
	ANSI_STRING astr;
	IO_ERROR_LOG_PACKET *lp;

	RtlInitAnsiString(&astr, str);
	lp=IoAllocateErrorLogEntry(driver, sizeof(IO_ERROR_LOG_PACKET)+
		(unsigned short)RtlAnsiStringToUnicodeSize(&astr));

	if (!lp)
		return;

	lp->ErrorCode = error;
	lp->NumberOfStrings = 1;
	lp->StringOffset = sizeof(IO_ERROR_LOG_PACKET);
	ustr.Buffer = (void*) ((char*)lp+lp->StringOffset);
	ustr.MaximumLength = (unsigned short) RtlAnsiStringToUnicodeSize(&astr);
	RtlAnsiStringToUnicodeString(&ustr, &astr, FALSE);
	IoWriteErrorLogEntry(lp);
}

void DriverUnload(PDRIVER_OBJECT driver)
{
	putk(driver, EVENTLOG_INFO, "driver unload");
	return;
}

NTSTATUS DriverEntry(PDRIVER_OBJECT driver, PUNICODE_STRING registry)
{
	putk(driver, EVENTLOG_INFO, "driver entry");
	driver->DriverUnload = DriverUnload;
	return STATUS_SUCCESS;
}
```

Alternativ, staţia destinaţie poate configura `syslogd` pentru a intercepta aceste mesaje. Mai multe informaţii puteţi găsi aici.

Module kernel in Windows

Pentru dezvoltarea de device drivere în Windows este necesară instalarea pachetului Windows DDK. Acest pachet conţine header-ele pentru compilare, exemple cât si utilitare pentru compilarea device driver-elor. Daca se pastrează configuraţia implicită, pachetul DDK va fi instalat în directorul `C:\WINDDK`. Acolo se găsesc header-ele, sursele şi restul utilitarelor.

Un exemplu de modul kernel

Probabil cel mai simplu modul kernel de Windows este cel prezentat mai jos:

hello_win.c

```
	MessageIdTypedef=NTSTATUS

	SeverityNames=(Informational=0x1:STATUS_SEVERITY_INFORMATIONAL
		       Warning=0x2:STATUS_SEVERITY_WARNING
		       Error=0x3:STATUS_SEVERITY_ERROR
		      )
```

```
        FacilityNames=(System=0x0
                      )

        MessageId=0x0001 Facility=System Severity=Informational
        SymbolicName=EVENTLOG_INFO Language=English
        %2.
        .

        MessageId=0x0002 Facility=System Severity=Warning
        SymbolicName=EVENTLOG_WARNING Language=English
        %2.
        .

        MessageId=0x0003 Facility=System Severity=Error
        SymbolicName=EVENTLOG_ERROR Language=English
        %2.
        .
```

Funcţia `DriverEntry` va fi rulată la inserarea modulului în kernel. Funcţia `DriverUnload` va fi apelată atunci când se scoate modulul din kernel. Funcţia `DbgPrint` va trimite şirul primit la debugger doar dacă modulul este compilat în mod checked.

Un exemplu mai complicat, care trimite mesaje ce sunt logate în userspace, este prezentat mai jos:

```
c:\so\tema1\objchk_w2k_x86\i386> driver mesg tema1.sys
...
info: driver entry
info: driver unload
Reached the end of the file.
```

Funcţia `putk` este folosită aici pentru a trimite un mesaj în userspace. A se remarca faptul că se folosesc şiruri de caractere Unicode şi nu ASCII. ATENŢIE: această funcţie o puteţi folosi numai la un nivel de întrerupere pasiv.

Mesajele pot fi vizualizate cu ajutorul utilitarului `EventViewer`, dar datorită modului în care se formează aceste mesaje, doar atât timp cât modulul este prezent în kernel. Aceasta datorită faptului că mesajele sunt formate de fapt dintr-un template, descris într-un fişier resursă ce este linkeditat în modulul kernel, şi şiruri ce sunt inserate în template în anumite locaţii. Mesajul template este identificat, în cazul nostru, de `EVENTLOG_INFO`, iar şirurile sunt inserate de `DriverEntry` şi `DriverUnload`.

Fişierul template ce defineşte tipurile de mesaje de log este prezentat mai jos (acest fişier este link-editat împreună cu modulul de kernel).

```
TARGETNAME = modul
TARGETPATH = obj
TARGETTYPE = DRIVER

SOURCES = modul.c
```

În cadrul fişierului template prezentat, `%1` reprezintă numele modulului, de la `%2` mai departe folosindu-se şirurile de inserat în mesaj. După cum am spus, acest fişier este compilat într-un fişier resursă şi apoi linkeditat împreună cu restul modulului în imaginea acestuia. Pentru ca EventViewer-ul să poată citi fişierul template el trebuie să ştie unde se află imaginea modulului. Din această cauză, la instalarea unui driver trebuie creată o cheie în registry, care să indice tocmai imaginea modulului. Ca o consecinţă, pentru că modulele dezvoltate de noi nu au caracter permanent, la scoaterea unui modul din kernel, se va şterge cheia din registry, astfel încât EventViewer-ul nu va mai putea afişa corect mesajele din loguri.

Pentru a ocoli acest neajuns, dar şi pentru a simplifica procesul de colectare a mesajelor, a fost dezvoltat un utilitar ce primeşte ca parametru imaginea modulului şi afişează toate mesajele prezente în log generate de acesta. Un exemplu de rulare este dat mai jos:

```
!ifdef NTMAKEENV
!include $(NTMAKEENV)\makefile.def
!else

DDK=C:\WinDDK00.16385.0
kbuild:
        -del $(DDK)\build.dat
        pushd . & $(DDK)\bin\setenv.bat $(DDK) chk wnet & @echo on & popd & build -cZ

!endif
```

Compilarea modulelor de kernel

Pentru compilarea modulelor de kernel se recomandă folosirea utilitarului build din DDK. Acest utilitar are nevoie de două fişiere care trebuie să fie prezente în directorul ce conţine sursele modulului: `sources` şi `makefile`.

Un exemplu de fişier `sources` este prezentat in continuare:

```
C:\pso\lab-01\modul-win> driver load objchk_wnet_x86\i386\modul.sys
C:\pso\lab-01\modul-win> driver unload modul
C:\pso\lab-01\modul-win> driver list
name                                    type       status
Abiosdsk                                driver     stopped
ACPI                                    driver     running
....
```

După cum se observă fişierul `sources` conţine informaţii despre sursele modulului, tipul acestuia (în `TARGETTYPE` - poate fi driver normal sau sistem de fişiere), numele modulului (în `TARGETNAME` ? fără extensia implicita `.sys`) şi directorul unde se vor plasa obiectele temporare şi modulul (în `TARGETPATH`). Fişierele sursă trebuie descrise în variabila `SOURCES`. În cazul nostru s-a inclus un singur fişier sursă (`.c`). În cazul în care modulul trebuie compilat din mai multe fişiere sursă, acestea vor fi specificate toate în variabila `SOURCES`, separate prin spaţii.

Fişierul `makefile` are un format standard, prin care se apelează utilitarul build, după ce a fost setat mediul de compilare dorit:

```
C:\pso\lab-01\modul-win>dir
 Volume in drive C has no label.
 Volume Serial Number is 3017-31C0

 Directory of C:\pso\lab-01\modul-win

03/02/2009  06:37 AM    <DIR>          .
03/02/2009  06:37 AM    <DIR>          ..
03/02/2009  06:13 AM               213 Makefile
03/02/2009  06:09 AM               303 modul.c
03/02/2009  06:22 AM                78 sources
               3 File(s)            594 bytes
               2 Dir(s)   3,419,484,160 bytes free

C:\pso\lab-01\modul-win>nmake

Microsoft (R) Program Maintenance Utility Version 9.00.21022.08
Copyright (C) Microsoft Corporation.  All rights reserved.

        del C:\WinDDK00.16385.0\build.dat
Could Not Find C:\WinDDK00.16385.0\build.dat
        pushd . & C:\WinDDK00.16385.0\bin\setenv.bat C:\WinDDK00.16385.0 chk w
net & @echo on & popd & build -cZ
BUILD: Compile and Link for x86
BUILD: Start time: Mon Mar 02 06:37:35 2009
BUILD: Examining c:\pso\lab-01\modul-win directory for files to compile.
BUILD: Compiling and Linking c:\pso\lab-01\modul-win directory
_NT_TARGET_VERSION SET TO WS03
Compiling - modul.c
Linking Executable - objchk_wnet_x86\i386\modul.sys
BUILD: Finish time: Mon Mar 02 06:37:36 2009
BUILD: Done

    3 files compiled
    1 executable built

C:\pso\lab-01\modul-win>driver load objchk_wnet_x86\i386\modul.sys

C:\pso\lab-01\modul-win>driver list
name                                    type       status
Abiosdsk                                driver     stopped
ACPI                                    driver     running
ACPIEC                                  driver     stopped
[...]
modul                                   driver     running

C:\pso\lab-01\modul-win>driver unload modul
```

Fişierul makefile este folosit în cadrul procesului de build, din această cauză este protejat de construcţia `ifdef`. Pentru rularea targetului `kbuild`, fişierul makefile va fi aplelat rulând utilitarul `nmake`.
Pentru compilarea unui modul kernel trebuie să se selecteze sistemul de operare (`w2k` pentru Windows 2000, `wxp` pentru Windows XP şi `wnet` pentru Windows 2003) dar şi dacă sistemul de operare trebuie să facă în plus verificări de consistenţă (`chk`) sau nu (`fre`). Pentru dezvoltare se recomandă modul `chk` întrucât bugurile vor fi detectate mai devreme. Pentru producţie se recomandă modul `fre`, în modul `chk` existând penalizări de performanţă.

Încărcarea/descărcarea unui modul de kernel

Spre deosebire de UNIX, în Windows nu există utilitare de încărcare/descărcare a unui modul în/din kernel. În mod normal, un modul este încărcat de către sistem la bootare. Din această cauză, uzual, un modul kernel trebuie instalat în directorul sistem, trebuie sa cream un set de chei în registry (aşa cum a fost specificat şi mai sus), care vor fi apoi scanate de "Service Control Manager". Acestea definesc de fapt servicii speciale (de tip driver sau sistem de fişier) care vor fi pornite de `SCMAN` şi astfel driverul va fi încărcat. Dacă, totuşi, se doreşte încărcarea unui modul în kernel la cerere, trebuiesc efectuate unele din aceste operaţii: crearea unui serviciu, crearea unui set de chei în registry şi pornirea serviciului.

driver - utilitar de încărcare/descărcare a modulelor de kernel

Pentru a face dezvoltarea şi testarea unui driver în Windows cât de cât facilă, am creat un utilitar care se ocupă cu toate aceste operaţii şi care primeşte ca parametru `load` şi imaginea modulului pentru încărcarea modulului în kernel. Pentru scoaterea modulului din kernel se foloseşte `unload` cu numele modulului, iar pentru listarea modulelor (active sau inactive, dar înregistrate) se foloseste `list`.

```
/*
 * Oops generating kernel module
 */

#include <ntddk.h>

#define OP_READ         0
#define OP_WRITE        1
#define OP_OOPS         OP_WRITE
#define POISON          0x00001234


void DriverUnload (PDRIVER_OBJECT driver)
{
}

NTSTATUS DriverEntry (PDRIVER_OBJECT driver, PUNICODE_STRING registry)
{
        int *a;

        a = (int *) POISON;
#if OP_OOPS == OP_WRITE
        *a = 3;
#elif OP_OOPS == OP_READ
        DbgPrint ("value = %d\n", *a);
#else
#error "Unknown op for oops!"
#endif

        driver->DriverUnload = DriverUnload;

        return STATUS_SUCCESS;
}
```

Un exemplu complet de compilare şi încărcare/descărcare a unui modul de kernel este prezentat în continuare:

```
C:\Program Files\Debugging Tools for Windows (x86)>kd -z C:\Windows\MEMORY.DMP
[...]
Loading Dump File [C:\Windows\MEMORY.DMP]
Kernel Complete Dump File: Full address space is available
[...]
Use !analyze -v to get detailed debugging information.

BugCheck 7E, {c0000005, f9ffc030, fa0cebbc, fa0ce8b8}
[...]
Probably caused by : bsod.sys ( bsod!DriverEntry+10 )

Followup: MachineOwner
---------

kd> ln f9ffc030
(f9ffc020)   bsod!DriverEntry+0x10   |  (f9ffd000)   bsod!KeTickCount

kd> u f9ffc030
bsod!DriverEntry+0x10:
f9ffc030 c70003000000    mov     dword ptr [eax],3
f9ffc036 8b4d08          mov     ecx,dword ptr [ebp+8]
f9ffc039 c7413410c0fff9  mov     dword ptr [ecx+34h],offset bsod!DriverUnload (f
9ffc010)
f9ffc040 33c0            xor     eax,eax
```

```
f9ffc042 8be5            mov     esp,ebp
f9ffc044 5d              pop     ebp
f9ffc045 c20800          ret     8
f9ffc048 0000            add     byte ptr [eax],al

kd> u bsod!DriverEntry
bsod!DriverEntry:
f9ffc020 8bff            mov     edi,edi
f9ffc022 55              push    ebp
f9ffc023 8bec            mov     ebp,esp
f9ffc025 51              push    ecx
f9ffc026 c745fc34120000  mov     dword ptr [ebp-4],1234h
f9ffc02d 8b45fc          mov     eax,dword ptr [ebp-4]
f9ffc030 c70003000000    mov     dword ptr [eax],3
f9ffc036 8b4d08          mov     ecx,dword ptr [ebp+8]
```

Este vorba de primul modul de Windows prezentat în laborator. Întrucât în cadrul modulului s-a folosit DbgPrint, se recomandă folosirea DbgView pentru afişarea mesajelor (utilitarul trebuie pornit înainte de încărcarea modulului în kernel).

ATENŢIE: calea către directorul cu sursele ce implementează modulul nu trebuie sa conţină spaţii; în caz contrar, compilarea modulului va eşua.

OSR Driver Loader

Un utilitar cu interfaţă grafică pentru lucrul cu module de kernel pe Windows este OSR Driver Loader. Îl puteţi găsi la OSR Online (va trebui să vă înregistraţi).

Utilitarul este foarte intuitiv. În primă fază va trebui precizată calea catre modul (fisierul .sys). Pentru încărcarea modulului va trebui înregistrat şi apoi pornit (`Register Service` şi `Start Service`) - echivalent cu `driver load`. Pentru descărcare va trebui deînregistrat şi apoi oprit (`Unregister Service` şi `Stop Service`) - echivalent cu `driver unload`. Butonul `Active Services` este folosit pentru listarea modulelor încărcate la un moment dat în kernel - echivalent cu `driver list`.

Mai multe informaţii se găsesc în fişierul de documentaţie (`OSR Driver Loader.chm`).

Depanarea unui modul de kernel

Pentru depanarea modulelor, programatorul are la dispoziţie câteva mecanisme: compilare checked, mesaje trimise pe serială către alt calculator sau un debugger.

Când intenţionaţi să folosiţi debuggerul ţineţi cont de faptul că sunt necesare două maşini: cea care este inspectată, şi cea pe care rulează debuggerul. Acestea vor să fie conectate printr-o legătură serială. Pentru a porni procesul de debug pe maşina ce se doreşte a fi inspectată trebuie pornit kernelul cu opţiunile `/bootport` şi `/baudrate` (de exemplu `/bootport=COM1` şi `/baudrate=9600`). Ţineţi cont de faptul că legătura serială o să limiteze cantitatea de informaţii schimbată între maşini. Pe cealaltă maşină trebuie să porniţi debugger-ul (`kd` sau `WinDbg`).
Pentru a folosi mesaje de debug utilizaţi funcţia `DbgPrint`, ce are aceiaşi parametri şi comportare ca şi funcţia `printf` (sau echivalentul `printk` pe Linux). Pentru a vizualiza aceste mesaje puteţi fie folosi debugger-ul - dar sistemul de operare trebuie boot-at cu opţiunea `/debug`, fie folosi utilitarul DbgView de la SysInternals.

De obicei apariţia unei erori la nivelul kernel-ului Windows duce la apariţia ecranului albastru. O informaţie preţioasă pe care o oferă acesta (echivalent cu mesajul decodificat al oops-ului) este adresa care a cauzat eroarea (valoarea EIP).

Exemplu de modul de kernel pentru depanare

În continuare este prezentat un modul de kernel care generează o eroare la nivelul nucleului prin accesarea unei regiuni invalide de memorie:

debug_win.c

```
f9ffc030 c70003000000    mov     dword ptr [eax],3
```

Încarcarea acestui modul în kernel va duce la apariţia unei erori şi a ecranului albastru. Informaţiile afişate pot fi recuperate după repornirea sistemului din log-uri sau din dump-ul de memorie creat după crash. Dump-ul creat (`%SystemRoot%\Memory.dmp`) este foarte important întrucât dă posibilitatea analizei locaţiei ce a generat eroarea.

Analiza folosind memory dump

Mai jos este prezentată o secvenţă de rulare a kd, folosind crash dump-ul generat (localizat în `%SystemRoot%\Memory.dmp`). Locaţia de memorie care a generat crash-ul (aparută şi pe ecranul albastru şi în informaţiile prezentate de debugger) a fost `0xf9ffc030`:

```
$ ssh -l root spook.local
$ ssh -l Administrator chooch.local
```

Folosind comanda `ln` se află care sunt simbolurile apropiate acelei adrese, urmând ca, ulterior, să se dezasambleze acea adresă şi funcţia-simbol asociată.

La fel ca în Linux, instrucţiunea ce a generat eroarea a fost:

```
$ ml
```

adică scrierea valorii `3` la o locaţie invalidă indicată de `eax` (adresa `1234h`), definită de macroul `POISON`.

Quiz

Pentru auto-evaluare (de preferat înainte de laborator) răspundeţi la întrebările de aici.

Exerciţii

- Folosiţi arhiva de sarcini a laboratorului.
- Pe maşina virtuală de Linux recomandăm folosirea `wget` pentru descărcarea arhivei.
- Maşinile virtuale pot fi accesate, respectiv, prin Multicast DNS, folosind numele `spook.local` (Linux) şi `chooch.local` Windows.
  - Pentru accesarea maşinilor virtuale puteţi folosi SSH (are mai mult sens pentru Linux). Autentificarea se realizează folosind chei (fără parolă):$ mw
  - Conturile maşinilor virtuale sunt:
    - Linux: root/student, student/student
    - Windows: Administrator/student, student/student
    - fiind vorba de kernel programming/driver development veţi folosi preponderent conturile privilegiate (`root` respectiv `Administrator`)
- Fiecare maşină virtuală are un director partajat prin SMB/CIFS. Un set de intrări în `/etc/fstab` pe sistemul gazdă permit montarea facilă a acestora. Share-urile sunt:
  - (**Linux**) directorul `/root` de pe maşina virtuală Linux se poate monta în sistemul gazdă folosind comanda: grep -r 'struct task_struct {' /usr/src/linux
    - directorul `/home/student/spook-share` sau icon-ul proaspăt creat pe Desktop permite accesarea share-ului
  - (**Windows**) directorul `C:\Cygwin\home\Administrator\share` se poate monta în sistemul gazdă folosind comanda:$ mw
    - directorul `/home/student/chooch-share` sau icon-ul proaspăt creat pe Desktop permite accesarea share-ului
- Daca nu sunt recunoscute numele spook.local sau chooch.local este posibil ca daemon-ul Avahi sa fie oprit. Reporniti folosind comanda `/etc/init.d/avahi-daemon start`.

Linux

- Folosiţi directorul `lin/` din arhiva de sarcini a laboratorului.
- Punctaj total: **6 puncte**
- **IMPORTANT** Pentru primele exercitii alegeti **prima** optiune de boot din GRUB (**nu** este cea implicita).

1. (**0.5 puncte**) Intraţi în directorul `1-test-mod/`.
   - Ce fişiere sunt prezente în acest director?
   - Inspectaţi fişierul C.
   - Obţineţi modulul kernel asociat.
   - **Hint**:
     - Citiţi secţiunea Compilarea modulelor de kernel din laborator.
   - Inseraţi modulul în kernel.
   - **Hint**:
     - Citiţi secţiunea Încărcarea/descarcarea unui modul de kernel din laborator.

- Listaţi modulele din kernel şi verificaţi existenţa modulului curent.
- Eliminaţi modulul din kernel.
- **Hint**:
  - La eliminare trebuie precizat doar numele modulului (fără extensie).
- Vizualizaţi mesajele afişate de modul la intrarea, respectiv ieşirea din nucleu.
- **Hint**:
  - Citiţi secţiunea Un exemplu de modul kernel din laborator.
- Curăţaţi directorul curent. Trebuie să rămână doar fişierele iniţiale.

2. (**0.5 puncte**) De ce mesajele nu au fost afişate direct la consolă?
   - Inspectaţi fişierul cu codul sursă. Modificaţi fişierul astfel încât mesajele să fie afişate direct la consolă.
   - **Hint**:
     - Citiţi secţiunea Printk Debugging din laborator.
   - Compilaţi modulul.
   - Inseraţi şi apoi eliminaţi modulul din kernel.
   - Mesajele sunt afişate la consolă (care consolă?).
   - Curăţaţi directorul curent.
3. (**0.5 puncte**) Intraţi în directorul `3-error-mod/`.
   - Obţineţi modulul de kernel asociat.
   - De ce au apărut acele erori?
   - **Hint**:
     - Cu ce diferă acest modul de modulul precedent?
   - Modificaţi modulul pentru a rezolva cauza apariţiei acelor erori.
   - Compilaţi, încărcaţi şi descărcaţi modulul.
4. (**1 punct**) Intraţi în directorul `4-multi-mod/`
   - Inspectaţi fişierele sursă C.
   - Creaţi un fişier Kbuild care să conducă la crearea fişierului-modul `multi_mod.ko` pornind de la cele două fişiere sursă C.
   - Exemplul este unul academic (modulul 2 conţine doar definiţia unei funcţii folosite de modulul 1).
   - **Hint**:
     - Citiţi secţiunea Compilarea modulelor de kernel din laborator.
   - Compilaţi, încărcaţi şi descărcaţi modulul.
   - Mesajele sunt afişate corespunzător la consolă.
5. (**1 punct**) Intraţi în directorul `5-oops-mod/`.
   - Inspectaţi fişierul sursă C. Observaţi unde va apărea problema.
   - Compilaţi modulul asociat.
   - **Hint**:
     - adăugaţi opţiunea -g pentru compilare
   - **Faceţi snapshot la maşina virtuală**.
   - **Hint**:
     - De acum încolo obişnuiţi-vă să faceţi snapshot la maşina virtuală înainte de încărcarea unui modul.
   - Încărcaţi modulul în kernel.
   - La ce adresă de memorie a apărut oops-ul?
   - Ce instrucţiune a dus la apariţia oops-ului?
   - **Hints**:
     - Folosiţi informaţia din `/proc/modules`.
     - Folosiţi `objdump` şi/sau `addr2line`.
     - `objdump` are nevoie de suport de debugging la compilare!
     - Citiţi secţiunea Exemplu de modul de kernel pentru depanare din laborator.
     - Citiţi secţiunile objdump şi addr2line din laborator.
   - Descărcaţi modulul din kernel. De ce nu funcţionează operaţia?
   - Reveniţi la snapshot-ul maşinii virtuale.
6. (**1 punct**) Reporniti masina virtuala si selectati optiunea din GRUB marcata cu `(debug)`.
   - Intraţi în directorul `6-cmd-mod/`
   - Inspectaţi fişierul sursă C.
   - Compilaţi modulul asociat.
   - Încărcaţi modulul pentru a vedea mesajul. Descărcaţi modulul.
   - Folosind `netconsole`, faceţi ca mesajul să fie trimis către maşina fizică
   - **Hints**:
     - Modulul `netconsole` este deja configurat si incarcat (puteti vedea folosind `lsmod`). Nu este nevoie de o operatie specializata pe masina virtuala.
     - Mesajele sunt transmise în mod implicit pe portul UDP 6000 al sistemului gazdă.
     - Puteţi porni o sesiune de captură `netcat` pe sistemul gazdă folosind icon-ul de pe `Desktop` marcat cu `Netconsole Terminal`.
   - Fără a modifica sursele, încărcaţi modulul în kernel, astfel încât mesajul afişat să fie *Early bird gets tired*
   - **Hint**:
     - Variabila `str` poate fi modificată ca parametru transmis modulului.
     - Accesati acest link.
7. (**1.5 puncte**) Intraţi în directorul `7-list-proc/`.

- Creaţi un modul care să afişeze informaţii despre procesul curent.
- Numele modulului trebuie sa fie `list_proc.ko`.
- Afişaţi process id-ul (pid-ul procesului) şi numele executabilului.
- Informaţiile vor fi afişate atât la încărcarea cât şi la descărcarea modulului.
- **Hints**:
  - Nu începeţi de la zero. Folosiţi fişierele `Makefile` şi `Kbuild` şi fişierele sursă C din directoarele anterioare.
  - În kernel-ul Linux un proces este descris de structura `struct task_struct`.
  - Folositi LXR pentru a afla conţinutul unei structuri din nucleu. Vă autentificaţi folosind numele de utilizator şi parola de pe curs.cs.pub.ro. **SAU** grep -r 'struct task_struct {' /usr/src/linux
  - Pointer-ul la structura procesului ce rulează la un moment dat în kernel este dat de variabila `current` (de tipul `struct task_struct *`).
  - Pentru a folosi variabila `current` va trebui să includeţi header-ul în care este definită structura `struct task_struct`.
- Compilaţi şi încărcaţi modulul obţinut.
- **Hint**:
  - Este indicat să faceţi snapshot la maşina virtuală înainte de încărcarea modulului în kernel.
- Descarcati modului de kernel.
- Repetati operatia incarcare/descarcare.
- De ce diferă rezultatele obţinute? (PID-urile proceselor)

Pentru acasă

1. Modificaţi modulul creat anterior pentru a afişa informaţii despre toate procesele din sistem la încărcarea modulului.
   - Comparaţi rezultatul obţinut cu ieşirea comenzii `ps`.
   - **Hints**:
     - Procesele din sistem sunt structurate într-o listă circulară.
     - Macrourile de tipul `for_each_?` (ca de exemplu `for_each_process`) sunt utile în situaţiile în care se doreşte parcurgerea elementelor dintr-o listă.
     - Pentru înţelegerea modului de folosire a unei funcţii sau a unui macro folosiţi LXR şi căutaţi un caz de utilizare.
2. Creaţi un modul de kernel care să afişeze zonele de memorie virtuală a procesului curent.
   - **Hints:**
     - Porniţi de la un modul existent.
     - Investigaţi structura `struct task_struct` şi structura `struct mm_struct`.
     - Aveţi grijă la header-ele care trebuie incluse pentru a avea acces la toate structurile necesare.

Windows

- Folosiţi directorul `win/` din arhiva de sarcini a laboratorului.
- Punctaj total: **5 puncte**
- **IMPORTANT**: **Nu** lucrati in directoare care contin spatii (blank).
  - Recomandam sa lucrati in `C:\Cygwin\home\Administrator\so2` sau `C:\Cygwin\home\Administrator\share`.

1. (**0.3 puncte**) Intraţi în directorul `1-test-mod/`.
   - Ce fişiere sunt prezente în acest director?
   - Inspectaţi fişierul C.
   - Obţineţi modulul kernel asociat.
   - Porniţi utilitarul `DbgView`.
   - Inseraţi modulul în kernel.
   - Listaţi modulele din kernel şi verificaţi existenţa modulului curent.
   - Eliminaţi modulul din kernel.
   - **Hint**:
     - Citiţi secţiunea Compilarea modulelor de kernel din laborator.
     - Citiţi secţiunea driver - utilitar de incarcare/descarcare a modulelor de kernel din laborator.
     - Lucrul cu module de kernel în Windows se realizează din consola DDK (`Windows Server 2003 Checked x86 Build Environment`)
     - Folosiţi icon-ul `x86 Checked Build Environment` din partea dreaptă a Desktop-ului.
   - Folosiţi pentru încărcarea/descărcarea modulului utilitarul `OSR Driver Loader` (partea dreaptă a Desktop-ului).
     - Succesiunea de operaţii este `Browse`, `Register Service`, `Start Service`, `Stop Service`, `Unregister Service`.
   - Curăţaţi directorul curent. Trebuie să rămână doar fişierele iniţiale.

2. (**0.2 puncte**) Configuraţi utilitarul `DbgView` pentru a stoca mesajele afişate într-un fişier.
   - Compilaţi (din nou, dacă aţi şters modulul `.sys` anterior) modulul de kernel `hello_mod`.
   - Porniţi şi configuraţi `DbgView`.
   - Inseraţi şi eliminaţi modulul din kernel.
   - Verificaţi stocarea mesajelor în fişierul specificat.
   - Curăţaţi directorul curent.
3. (**0.5 puncte**) Intraţi în directorul `3-multi-mod/`.
   - Compilaţi fişierele sursă C din director în modulul `multi_mod.sys`.
   - **Hints**:
     - Trebuie creat doar fişierul `source` necesar.
     - Citiţi secţiunea Compilarea modulelor de kernel din laborator.
   - Inserati si descarcati modulul din kernel.
   - Curatati directorul curent.
4. (**1.5 puncte**) Intraţi în directorul `4-bsod-mod/`.
   - Inspectaţi fişierul sursă C. Observaţi unde va apărea problema.
   - Compilaţi modulul asociat.
   - Încărcaţi modulul în kernel.
   - Reţineţi din BSOD adresa la care s-a produs eroarea.
   - Reporniţi maşina virtuală.
   - Folosiţi `WinDbg` pentru a afla ce instrucţiune a dus la apariţia oops-ului.
   - **Hint**:
     - Citiţi secţiunea Analiza folosind memory dump din laborator.
   - Accesaţi submeniul `FileOpen Crash Dump` din WinDbg. Dump-ul se găseşte în `C:\Windows\MEMORY.DMP`
5. (**1 punct**) Intraţi în directorul `5-proc-info/`.
   - Creaţi un modul de kernel care să afişeze id-ul procesului curent şi id-ul thread-ului curent.
   - Efectuaţi operaţia de afişare atât la încărcarea cât şi la descărcarea modulului.
   - Ce proces este folosit pentru încărcarea/descărcarea modulului?
     - **Hints**:
       - Nu începeţi de la zero. Folosiţi fişierul `makefile` şi fişierele sursă C din directoarele anterioare.
       - Funcţiile pe care trebuie să le folosiţi sunt definite în header-ul ntos/inc/ps.h din WRK.
         - Folositi sirul "PsGet" pentru cautare.
       - Nu trebuie inclus header-ul `ps.h`. Este automat inclus de `ntddk.h`
6. (**1.5 puncte**) Rămâneţi în directorul `proc-info/`.
   - Completaţi modulul anterior astfel încât să afişeze numele imaginii procesului.
   - **Hints**:
     - Structura reprezentativă pentru procese în kernel-ul Windows este `EPROCESS`.
     - Va trebui să aflaţi pointer-ul la procesul curent folosind tot o funcţie din header-ul base/ntos/inc/ps.h
     - Investigaţi header-ul ce conţine definiţia structurii `EPROCESS` folosind LXR.
     - Structura `EPROCESS` este opacă pentru dezvoltatorul de module de kernel (nu se pot referenţia elemente ale structurii).
     - Pentru accesarea câmpului asociat imaginii procesului va trebui să cunoaşteţi offset-ul acestuia în cadrul structurii.
     - Pentru a afla offset-ul unui câmp în cadrul unei structuri kernel folosiţi `livekd` şi comanda `dt`.
     - **Convenţia** pentru tipuri şi typedef-uri în Windows este: structura e cu _ şi typedef-ul fără
     - Comanda `dt` acceptă o structură ca parametru
     - Consultaţi secţiunea `livekd` din laboratorul 1.
     - Folositi `WinDbg` si optiunea `Local Kernel Debug` ca wrapper peste `livekd`.
     - Nu trebui inclus header-ul `ps.h`. Este automat inclus de `ntddk.h`

Soluţii

- Soluţii exerciţii laborator 2

Resurse utile

Linux

1. Linux Device Drivers, 3rd edition

  - Chapter 2. Building and Running Modules
  - Chapter 4. Debugging Techniques
2. The Linux Kernel Module Programming Guide
3. (nearly) Complete Linux Loadable Kernel Modules
4. www.urbanmyth.org: OOPS!
5. Linux Kernel Debugging

Windows

1. Utilitar de incarcat/descarcat/listat module in Windows
2. The Windows 2000 Device Driver Book, Second Edition ? Chapter 17. Testing and Debugging Drivers
3. The Code Project - Device Drivers
4. Windows Kernel-Mode Driver Components
5. Introduction to Device Drivers
6. OSR Online - The Home Page for Windows Driver Developers
7. Windows Device Drivers Articles
8. Getting Started with Windows Drivers

[1] Pentru mai multe informaţii despre printk consultaţi laboratorul 3

[2] Nivelul implicit setat pe consolă se poate configura din /proc/sys/kernel/printk; spre exemplu, comanda `echo 8 > /proc/sys/kernel/printk` va face ca toate mesajele din kernel să fie afişate la consolă

[3] Pentru a verifica / modifica ce fişiere sunt folosite pentru logarea erorilor fiecărui nivel (loglevel), inspectaţi fişierul de configurare pentru syslogd, `/etc/syslog.conf`