Laborator 1 - Introducere

Obiectivele laboratorului

- prezentarea regulilor şi modului de desfăşurare a laboratorului de Sisteme de Operare 2
- prezentarea suportului de laborator
- prezentarea kernel-ului Linux, a kernel-ului Windows şi a resurselor aferente
- compilarea şi interacţiunea cu nucleul Linux şi Windows

Cuvinte cheie

- kernel, kernel programming
- Linux, vanilla, http://www.kernel.org
- .config, make menuconfig, make, make install
- gdb, /proc/kcore
- bzImage, vmlinuz, initrd
- LXR
- /boot/grub/grub.cfg, boot.ini
- Windows NT, Windows Driver Kit (WDK)
- Windows Research Kernel (WRK)
- Debugging Tools, WinDbg, LiveKd

Materiale ajutătoare

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

Desfăşurarea laboratorului

Laboratorul de Sisteme de Operare2 este unul de kernel programming şi driver development. Obiectivele laboratorului sunt:

- aprofundarea noţiunilor prezentate la curs
- prezentarea interfeţelor de programare la nivelul nucleului (kernel API)
- dezvoltarea deprinderilor de documentare, dezvoltare şi depanare pe un mediu freestanding
- dobândirea de cunoştinţe şi deprinderi pentru dezvoltarea driverelor

Un laborator va prezenta un anumit set de noţiuni, aplicaţii şi comenzi specifice unei problematici date. Laboratorul va debuta cu o prezentare (fiecare laborator va avea ca suport un set de slide-uri) (15 minute) urmând ca restul timpului să fie alocat rezolvării de exerciţii de laborator (80 de minute).

Pentru o desfăşurare cât mai bună a laboratorului, vă recomandăm parcurgea slide-urilor aferente şi rezolvarea quiz-ului. Pentru înţelegerea deplină a laboratorului, recomandăm parcurgerea suportului de laborator. Pentru aprofundare, folosiţi documentaţia de suport.

Suport de laborator

- Linux
  - Linux Kernel Development, 2nd Edition
  - Linux Device Drivers, 3rd Edition
  - Essential Linux Device Drivers
- Windows
  - Programming the Windows Driver Model
  - The Windows 2000 Device Driver Book, 2nd Edition
- General

- lista de discuţii (căutare în arhiva listei de dicuţii)

Compilarea kernel-ului Linux

Principalul motiv pentru care se doreşte compilarea kernel-ului este adaptarea acestuia la un anumit sistem cu scopul creşterii performanţelor, adaptare care de obicei constă în:

- utilizarea unor opţiuni de compilare prin care arhitectura procesorului este folosită la maxim (ex: alegerea tipului de procesor - implicit se foloseşte un procesor generic gen i386)
- activarea/dezactivarea suportului pentru anumite componente hardware sau software (ex: selectarea anumitor driver-e, selectarea unor anumite protocoale de comunicatie)

În cazul kernel-ului Linux, suportul pentru hardware/software poate să fie:

- **built-in** - suportul este inclus direct în imaginea (fisierul) kernel-ului
- sub forma de **module de kernel** - suportul se găseşte în fişiere separate care se încarcă on-demand

Deşi adăugarea funcţionalităţilor în kernel poate duce la creşterea eficienţei în anumite subsisteme, acest lucru poate avea ca şi consecinţă îngreunarea sistemului în ansamblu (bloated), lucru nerecomandat.

Majoritatea distribuţiilor includ în kernel numai suportul pentru componentele critice, fără de care sistemul de operare nu poate funcţiona. Restul componentelor sunt compilate sub forma de module, module care sunt încărcate în funcţie de hardware-ul/necesităţile fiecărui sistem/utilizator.

Există şi situaţii în care kernel-ul care vine cu o anumită distribuţie este mai vechi sau nu are compilat suportul pentru o anumită componentă. De asemenea, compilarea unui kernel optimizat este des întalnită pe sistemele care ruleaza server-e unde se doreşte exploatarea la maxim a resurselor sistemului.

Pentru a putea efectua modificări de tipul celor mentionate mai sus, trebuie iniţial efectuată configurarea kernel-ului. După această etapă se poate porni compilarea kernel-ului şi a modulelor de kernel.

Pregătire

Cerinţe hardware şi software

Înaintea compilării kernel-ului trebuie verificat dacă sistemul satisface cerinţele hardware şi software.

Din punct de vedere hardware:

- deşi un sistem Linux minimal poate rula în sisteme cu puţină memorie RAM, cerinţele minime pentru o distribuţie Linux, în momentul de faţă, variază în jurul valorilor de 128-256 MB RAM.
- în funcţie de opţiunile alese pentru compilare, spaţiul ocupat pe disc poate depăşi valoarea de 500 MB - în aceste condiţii, se recomandă un minim de 1 GB liber pe hard-disk-ul pe care se realizează compilarea.

Timpul de compilare variază în funcţie de numărul de componente activate pentru compilare în momentul configurării.

Compilarea se poate realiza pe un sistem mai puternic urmând ca apoi să se mute pachetul ce conţine kernel-ul pe sistemul destinaţie.

Cerinţele software pentru kernel-ul de compilat se găsesc precizate în fişierul Documentation/Changes din cadrul surselor:

```
o  Gnu C                  3.2                  # gcc --version
o  Gnu make               3.79.1               # make --version
o  binutils               2.12                 # ld -v
o  util-linux             2.10o                # fdformat --version
o  module-init-tools      0.9.10               # depmod -V
o  e2fsprogs              1.29                 # tune2fs
o  jfsutils               1.1.3                # fsck.jfs -V
o  reiserfsprogs          3.6.3                # reiserfsck -V 2>&1|grep reiserfsprogs
o  xfsprogs               2.6.0                # xfs_db -V
o  pcmciautils            004                  # pccardctl -V
o  quota-tools            3.09                 # quota -V
o  PPP                    2.4.0                # pppd --version
o  isdn4k-utils           3.1pre1              # isdnctrl 2>&1|grep version
o  nfs-utils              1.0.5                # showmount --version
o  procps                 3.2.0                # ps --version
```

```
o  oprofile              0.9                  # oprofiled --version
o  udev                  081                  # udevinfo -V
o  grub                  0.93                 # grub --version
```

Observaţii:

- cerinţele de mai sus sunt valabile pentru versiunea de kernel 2.6.31.6.
- utilitarele `e2fsprogs`, `jfsutils`, `reiserfsprogs`, `xfsprogs` sunt folosite pentru sistemul de fişiere asociat
- `PPP` şi `isdn4k-utils` sunt necesare numai dacă legăturile sistemului la Internet sunt de tip PPP sau ISDN.

Ce hardware există în sistem?

Un kernel compilat poate oferi suport doar pentru hardware-ul utilizatorului, micşorând astfel dimensiunea imaginii obţinute. De asemenea, cunoaşterea hardware-ului în sistem este necesară pentru un kernel mai rapid şi alegerea opţiunilor de compilare şi a driver-elor de dispozitiv corecte.
Informaţii despre controller-ele Ethernet, VGA, placa de sunet se află cu ajutorul comenzii `lspci` (din pachetul `pciutils`):

```
# lspci
00:00.0 Host bridge: VIA Technologies, Inc. K8M800 Host Bridge
00:00.1 Host bridge: VIA Technologies, Inc. K8M800 Host Bridge
00:00.2 Host bridge: VIA Technologies, Inc. K8M800 Host Bridge
00:00.3 Host bridge: VIA Technologies, Inc. K8M800 Host Bridge
00:00.4 Host bridge: VIA Technologies, Inc. K8M800 Host Bridge
00:00.7 Host bridge: VIA Technologies, Inc. K8M800 Host Bridge
00:01.0 PCI bridge: VIA Technologies, Inc. VT8237 PCI bridge [K8T800/K8T890 South]
00:0a.0 Ethernet controller: Linksys, A Division of Cisco Systems [AirConn] ...
```

Tipul procesorului se află cu ajutorul `procfs` (montat în `/proc`).

```
# cat /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 15
model           : 28
model name      : Mobile AMD Sempron(tm) Processor 2800+
stepping        : 0
cpu MHz         : 1601.122
cache size      : 256 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 1
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr ...
bogomips        : 3205.53
```

Obţinerea surselor

Sursele de kernel de Linux pot fi obţinute din pachete specifice distribuţiei sau pot fi descărcate sursele oficiale ale lui Linus Torvalds.

În cazul în care se alege varianta folosirii surselor oficiale, se recomandă folosirea unui mirror din Romania.

Sursele kernel-ului se găsesc în subdirectorul `/pub/linux/kernel/v2.6` (pentru versiunea 2.6). Se poate folosi http sau ftp pentru obţinerea surselor:

```
# cd /usr/src
# wget ftp://ftp.idilis.lkams.kernel.org/pub/linux/kernel/v2.6/linux-2.6.24.2.tar.gz
```

Se dezarhivează sursele. Se recomandă crearea unei legături simbolice cu numele `linux` către directorul ce contine sursele.

```
# cd /usr/src
# tar xzf linux-2.6.24.2.tar.gz
# ln -s linux-2.6.24.2 linux
# ls -l
total 40924
lrwxrwxrwx  1 root src        14 2008-02-19 13:52 linux -> linux-2.6.24.2
drwxrwxrwx 19 root root     4096 2008-02-16 19:21 linux-2.6.24.2
-rw-r--r--  1 root src  41853865 2008-02-16 19:28 linux-2.6.24.2.tar.gz
```

Kernel-ul compilat de pe maşina virtuală de Linux disponibilă este un kernel versiunea 2.6.31.6.

Configurare

Partea de configurare este partea cea mai importantă a procesului de compilare. În cadrul acesteia se decide ce caracteristici vor fi incluse în noul kernel; sunt necesare cunoştinţe ale hardware-ului sistemului şi ale facilităţilor dorite.

Procesul de configurare era unul destul de dificil de realizat la primele versiuni, însa acest lucru s-a schimbat odată cu introducerea unor interfeţe care folosesc X Window sau ncurses.
Pentru verificarea opţiunilor posibile de compilare se poate rula comanda:

```
# make help
Cleaning targets:
  clean           - remove most generated files but keep the config
  mrproper        - remove all generated files + config + various backup files

Configuration targets:
  config          - Update current config utilising a line-oriented program
  menuconfig      - Update current config utilising a menu based program
  xconfig         - Update current config utilising a QT based front-end
  gconfig         - Update current config utilising a GTK based front-end
  oldconfig       - Update current config utilising a provided .config as base
  randconfig      - New config with random answer to all options
  defconfig       - New config with default answer to all options
  allmodconfig    - New config selecting modules when possible
  allyesconfig    - New config where all options are accepted with yes
  allnoconfig     - New config where all options are answered with no
  ...
```

Pentru utilizarea unei interfeţe text-based (ncurses), va trebui instalat pachetul `libncurses-dev`. Pentru utilizarea unei interfeţe folosind front-end GTK (de obicei într-un desktop environment GNOME) vor trebui instalate pachetele `libgtk2.0-dev` şi `libglade2.0-dev`. Pentru QT va trebui instalat pachetul `libqt4-dev`.

Avantajul folosirii interfetei ncurses este faptul că este relativ uşor de utilizat şi nu necesită prezenta unui mediu grafic.

În cazul în care sursele conţineau o configuraţie anterioară care nu mai este dorită, va trebui rulată una dintre comenzile:

```
# make clean
# make mrproper
```

Rularea uneia dintre comenzile:

```
# make menuconfig
# make gconfig
# make xconfig
```

rezultă în afişarea unui meniu. Acesta conţine mai multe intrări de configurare (`General setup`, `Networking`, `Device drivers`, `File systems`, etc.) care pot fi utilizate pentru configurări ale unui subdomeniu. Acestea pot conţine, la rândul lor, alte subdomenii de configurare.

O opţiune finită de configurare (spre exemplu `Device Drivers -> Block Devices -> Normal floppy disk support`) poate prezenta utilizatorului trei optiuni de configurare:

- absenţa completă din kernel-ul finit (se apasa)
- compilarea acesteia în cadrul imaginii de kernel (**built-in**) (se apasa)
- compilarea acesteia sub forma de **modul de kernel** (se apasa)

Compilarea **built-in** înseamnă introducerea codului obiect asociat opţiunii în imaginea de kernel care va rezulta. Compilarea în forma de **modul de kernel** înseamnă că pentru activarea acelei facilităţi, kernel-ul va încărca modulul (codul obiect asociat) şi îl va descărca atunci când nu are nevoie de el. Kernel-ul este astfel extensibil şi pentru adăugarea anumitor facilităţi nu este nevoie de recompilare. Pentru lucrul cu module de kernel vor trebui adăugate opţiunile din Loadable module support (`Enable loadable module support`).

Multe opţiuni nu vor fi încorporate în cadrul kernel-ului întrucât nu sunt necesare. Altele pot fi compilate numai **built-in**. Opţiunile care suportă varianta **modul de kernel** sau **built-in** se recomandă a fi compilate ca **module de kernel** pentru a fi încărcate la nevoie. Pe procesoarele moderne, timpul în care se încarcă/descarcă module este suficient de mic încat compilarea ca **modul de kernel** sau **built-in** să nu afecteze performanţa.

O opţiune utilă este precizarea unei versiuni locale pentru kernel, astfel încât acesta să fie identificat; exista posibilitatea compilării aceleiaşi versiuni de kernel pentru scopuri distincte; oferirea unei versiuni locale generează o versiune de kernel unică.

Precizarea tipului de procesor şi a arhitecturii este un pas necesar pentru a crea un kernel eficient. Majoritatea opţiunilor ţin de preferinţele utilizatorilor (desi probabil multi vor alege suport de networking, sunet etc.) sau de hardware-ul existent.

Dacă se doreşte crearea unui kernel pentru dezvoltare (development) atunci vor trebui activate opţiunile din `Kernel hacking`; acestea oferă informaţii de debug suplimentare, cu dezavantajul unui kernel mai mare şi mai lent.

Configuraţia este salvată în cadrul fişierului `.config` din directorul rădăcină al surselor. Este indicat să se realizeze un backup al acestui fişier înainte de configurare pentru a avea o configuraţie sigură la care să se revină în cazul apariţiei de probleme.

initrd - ramdisk-ul iniţial

initrd (initial ramdisk) este un sistem de fişiere temporar având ca suport memoria RAM (ramdisk) care este folosit la pornirea sistemului (booting). Initrd este folosit pentru a încărca driver-ele necesare încărcării sistemului de fişiere rădăcina.

Motivaţia folosirii initrd este flexibilitatea. Distributiile Linux au un kernel generic Linux care trebuie să boot-eze de pe sisteme cu hardware diferit. Kernel-ul inclus trebuie să fie modular, nefiind posibilă compilarea statică a tuturor opţiunilor fără a mări semnificativ imaginea kernel-ului. Este, în consecinta, necesar să se cunoască la booting locaţia sistemului de fişiere rădăcină şi ce driver-e vor trebui încărcate în kernel. Această problemă este rezolvată prin introducerea initrd ca pas intermediar în pasul de boot-ing. Acesta actioneaza ca un sistem de fisiere rădăcină temporar. Conţinutul acestui sistem rădăcină este dat de imaginea de initrd.

De obicei, compilarea unui kernel pentru un sistem dat nu necesită utilizarea initrd, deoarece se cunoaste hardware-ul existent şi sistemul de fisiere utilizat. Pentru a se evita utilizarea initrd, vor trebui compilate în imaginea de kernel (**built-in**) driver-ele de hard-disk, SCSI (dacă există) şi de sisteme de fişiere. Daca aceste driver-e ar fi compilate ca module atunci ar trebui încărcate de pe hard-disk, fără însă a putea accesa hard-disk-ul (din lipsa driver-elor). În această situaţie se foloseste initrd. Driver-ele în cauză se găsesc în:

- `Device Drivers -> ATA/ATAPI/MFM/RLL support` - de obicei trebuie compilate **built-in** driverele IDE specifice hardware-ului folosit pentru a putea beneficia de modurile Ultra DMA. Daca nu, se pot pune driverele generice (`generic/default IDE chipset support` şi `Generic PCI IDE Chipset Support`)
- `Device Drivers -> SCSI device support`
- `File systems` (va trebui configurat ca **built-in** suportul pentru sistemul de fişiere rădăcină)

Pentru utilizarea initrd este necesar pachetul `initrd-tools`.

(re)denumirea versiunii de kernel

Înainte de compilare, este indicat să se creeze un indicator unic pentru imaginea de kernel creată. Aceasta se poate realiza completând campul EXTRAVERSION din Makefile-ul kernelului. Se poate seta acest camp la orice şir de caractere.

Compilare

Faza de compilare presupune **obţinerea** imaginii de kernel şi compilarea **modulelor** de kernel. Acest lucru se realizează prin intermediul a doua comenzi:

```
# make bzImage
# make modules
```

Prima comandă creează o imagine de kernel comprimată. Acest pas poate dura de la cateva minute până la câteva zeci, depinzând de configuraţia hardware. Dupa încheiere, imaginea comprimată se regaseşte în `arch/i386/boot/bzImage` (pentru o arhitectura x86).

A doua comandă compilează **modulele** care pot fi încărcate de kernel. Acest pas poate dura de cateva ori mai mult decat pasul precedent. Fişierele obiect ce reprezintă modulele (cu extensia `.ko`) rezidă în directoarele asociate, urmând a fi instalate.

Instalare

Instalarea presupune copierea imaginii de kernel şi a modulelor in locurile prevăzute şi configurarea bootloader-ului pentru a boot-a noul kernel. Acest pas se leagă de directorul `/boot` unde se găsesc toate fişierele importante.

Instalare imagine kernel

Imaginea de kernel şi fişierele asociate sunt copiate în directoarele necesare cu ajutorul comenzii `make install`.

Paşii executaţi prin intermediul comenzii `make install` sunt detaliaţi mai jos.

Imaginea de kernel va trebui copiată în `/boot`:

```
# cd /usr/src/linux
# cp arch/x86/boot/bzImage /boot/vmlinuz-2.6.24.2mykernel
```

(şirul *mykernel* este folosit pentru identificarea imaginii de kernel; poate coincide cu versiunea locală precizată la configurare)

În plus faţă de imaginea de kernel, se recomandă copierea fişierului de configurare şi a tabelei de simboluri.

```
# cd /usr/src/linux
# cp .config /boot/config-2.6.24.2mykernel
# cp System.map /boot/System.map-2.6.24.2mykernel
```

Instalare module de kernel

Instalarea modulelor de kernel se realizează prin intermediul comenzii:

```
# make modules_install
```

Modulele sunt instalate in `/lib/modules/2.6.24.2mykernel`.

Daca s-a configurat sistemul pentru a folosi *initrd*, va trebui creată imaginea de *ramdisk*. Pentru aceasta se ruleaza comanda:

```
# cd /boot
# mkinitrd -o /boot/initrd.img-2.6.24.2mykernel 2.6.24.2mykernel
```

Comanda va inspecta directorul `/lib/modules/2.6.24.2mykernel` si va crea imaginea de ramdisk corespunzatoare.

Configurare GRUB

Dupa instalarea imaginii de kernel şi a modulelor de kernel va trebui configurat bootloader-ul pentru a şti de unde să incarce imaginea la pornirea sistemului.

Vom presupune configurarea GRUB (GRand Unified Bootloader), în detrimentul LILO (LInux LOader), pentru că este mai răspândit. Configurări pentru LILO se pot găsi şi in link-urile de mai jos.

Există două versiuni majore de GRUB, 1 şi 2. Mai departe ne vom ocupa de configurare GRUB2.

Pentru a adăuga o intrare în meniul GRUB-ului se editează fişierul `/etc/grub.d/40_custom` şi apoi se rulează comanda:

```
# update-grub
```

Un exemplu de configurare este prezentat în continuare:

```
[...]

menuentry "Linux" {
    set root=(hd0,1)
    linux /boot/vmlinuz-2.6.24.2mykernel root=/dev/sda1
    initrd /boot/initrd.img-2.6.24.2mykernel
}
```

Opţiunea initrd poate fi omisă în cazul în care nu s-a configurat un ramdisk iniţial.

Repornire sistem

Pentru rularea noului kernel, va trebui repornit sistemul şi optat pentru noul kernel din meniul bootloader-ului.

ATENTIE: Se recomandă păstrarea fostului kernel, în cazul în care apar probleme la noul kernel compilat. Probleme pot aparea din neincluderea driver-elor necesare în cazul în care nu se foloseşte initrd, omiterea driver-ului de sistem de fişiere necesar etc.

E posibil sa avem la dispoziţie un kernel care ruleaza dar caruia ii lipsesc funcţionalitaţi (de exemplu networking), pentru ca s-a omis compilarea driver-elor pentru placa de reţea. În acest caz se poate recompila kernel-ul pentru introducerea noilor funcţionalităţi. Noua compilare va dura mai puţin în cazul în care modificările sunt minime.

O metoda de troubleshooting este compararea unei configuraţii functionale cu cea curentă prin inspecţia fişierelor `.config` asociate.

Link-uri

- initrd
- Gentoo - Configuring the kernel
- The Very Verbose Guide to Updating and Compiling Your Debian Kernel
- Debian and the kernel
- Kernel Build - HowTo
- The Linux Kernel HOWTO
- Linux Kernel
- The System.map file
- LILO mini-HOWTO
- Configuring the Bootloader
- Boot with GRUB
- README din cadrul surselor kernel-ului
- Configurare GRUB1
- Configurare GRUB2

Compilarea kernel-ului Windows

În cadrul Windows Academic Program există posibilitatea accesului la codul sursa a kernel-ului de Windows NT, prin intermediul inițiativei Windows Research Kernel (WRK).

Codul sursă prezintă componentele cele mai importante din cadrul nuclelui (managementul memoriei, procese, thread-uri, scheduling, I/O manager) şi poate fi folosit şi modificat în scopuri non-comerciale. Se pot astfel urmări diversele mecanisme de implementare şi design care stau la baza kernel-ului şi se pot testa diverse alte soluţii prin modificarea surselor. Accesul la sursele kernel-ului este limitat, anumite subdomenii (cum ar fi networking-ul) fiind absente.

Accesul la codul sursă pentru nucleul de Windows NT înseamna posibilitatea de modificare a acestuia şi, evident, de compilare a kernel-ului şi de boot-are. Sursele sunt accesibile prin intermediul imaginii de CD pusă la dispoziţie de Microsoft în cadrul inițiativei WRK.

Din păcate, kernel-ul de Windows nu vine cu opţiuni de configurare, astfel încat procesul se rezumă la rularea comenzii de compilare şi la instalarea noului kernel. Dacă se doreşte un tip special de funcţionalitate vor trebui alterate sursele.

De asemenea, kernel-ul poate fi compilat numai pe un sistem Windows 2003 SP1 sau Windows XP x64. Versiunea curentă (WRK-1.2) nu poate fi compilată pe un sistem Windows XP x86. Sursele sunt disponibile în directorul `C:\cygwin\home\Administrator\so2\WRK\` din cadrul maşinii virtuale de Windows.

Etapele de compilare sunt prezentate şi în fişierul **README.txt** din rădăcina surselor.

Compilare

Pentru compilarea surselor se parcurg următorii paşi (vom utiliza de acum inainte `%wrk%` ca rădăcina surselor de kernel de Windows):

```
C:\>set wrk=C:\cygwin\home\Administrator\so2\WRK\WRK-v1.2
C:\>set arch=x86
C:\>set path=%wrk%\tools\%arch%;%path%
C:\>cd %wrk%\base
C:\cygwin\home\Administrator\so2\WRK\WRK-v1.2\base>cd ntos
C:\cygwin\home\Administrator\so2\WRK\WRK-v1.2\base\ntos>nmake -nologo %arch%=
```

Imaginea de kernel obţinută se va regăsi în `%wrk%\base\ntos\BUILD\EXE` şi va purta numele `wrkx86.exe` pentru un sistem cu arhitectura x86. Imaginea obţinută este doar nucleul; modulele de kernel folosite vor fi cele existente în sistem în acel moment.

Instalare

Procesul de instalare presupune copierea imaginii kernel-ului în `%SystemRoot%\system32`:

```
C:\>copy %wrk%\base\ntos\BUILD\EXE\wrkx86.exe %SystemRoot%\system32\
```

Totuşi, în afara imaginii de kernel, va trebui precizată imaginea de HAL (Hardware Abstraction Layer) care va fi utilizată. Va trebui gasită imaginea corecta de HAL; există trei imagini de HAL disponibile in `%wrk%\WS03SP1HALS\x86`. Pentru a afla care imagine este cea corectă va trebui utilizat linker-ul (folosind comanda `link` şi analizată imaginea de HAL existentă în acest moment în sistem).

```
C:\>"C:\Program Files\Microsoft Visual Studio 8\VC\vcvarsall.bat"
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\>link -dump -all %SystemRoot%\system32\hal.dll | findstr pdb
  80011530: 68 61 6C 2E 70 64 62 00 00 00 00 00 00 00 00 00  hal.pdb.........
    42435B3A cv           20 00001518      918    Format: RSDS, {2ECB059A-3F06-4
285-8E30-3FDE64119692}, 1, hal.pdb
```

Motivul pentru care în ieşirea comenzii nu apare şirul `halaacpi.dll` este că a fost dezactivat ACPI pe maşina virtuală de Windows 2003.

Asocierea dintre ieşirea comenzii de mai sus şi imaginile disponibile în `%wrk%\WS03SP1HALS\x86` este (după cum este precizat şi în README.txt):

```
    halacpi.dll  -> halacpim.dll
    halaacpi.dll -> halmacpi.dll
    halapic.dll  -> halmps.dll
```

De obicei, imaginea căutată va fi `halmacpi.dll`.

Imaginea corectă de HAL va fi copiată tot în `%SystemRoot%\system32`:

```
C:\>copy %wrk%\WS03SP1HALS\x86\halmacpi\halmacpi.dll %SystemRoot%\System32\
```

Configurare boot.ini

Pentru a boota proaspătul kernel va trebui adaugată o intrare în `C:\boot.ini`, fişierul de configurare pentru loader-ul de Windows NT. Se recomandă copierea unei linii de bootare existente şi modificarea ei pentru a boota noul kernel şi noua imagine de HAL, ca mai jos:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Windows Server 2003, Standard"
/fastdetect /NoExecute=OptOut
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Windows Server 2003, Standard - WRK-1.2
compiled kernel" /kernel=wrkx86.exe /hal=halmacpi.dll /fastdetect /NoExecute=OptOut
```

Pentru vizualizarea `boot.ini` în Windows Explorer, va trebui să accesaţi `Tools -> Folder Options -> View -> Hide protected operating system files (Recommended)`. Fişierul este implicit read-only; pentru editare va trebui să anulaţi această opţiune. Alternativ puteţi face acest lucru din linia de comanda:

```
C:\>attrib -h -s -r boot.ini
```

După configurarea `boot.ini`, sistemul poate boota în noul kernel compilat.

- How to manually edit the Boot.ini file in a Windows Server 2003 environment
- Available switch options for the Windows XP and the Windows Server 2003 Boot.ini files

LXR Cross-Reference

LXR (LXR Cross-Reference) este un program care permite indexarea şi referenţierea simbolurilor din codul sursă a unui program prin intermediul unei interfeţe web. Interfaţa web prezintă link-uri către locaţiile din fişiere unde un simbol este definit sau utilizat. Site-ul de dezvoltare pentru LXR este acesta. Utilitare asemănătoare sunt OpenGrok şi Gonzui.

Deşi LXR a fost iniţial destinat surselor kernel-ului de Linux, este folosit şi la sursele utilitarelor de la Mozilla, Apache HTTP Server şi FreeBSD.

Există o serie de site-uri care folosesc LXR pentru cross-referencing la sursele kernel-ului Linux, site-ul principal fiind site-ul iniţial de dezvoltare.

LXR permite căutarea după un identificator (simbol), după un text liber sau după un nume de fişier. Principala caractateristică şi, în acelaşi timp, principalul avantaj furnizat este posibilitatea de găsire facilă a declaraţiei oricărui identificator global. Se realizează astfel foarte rapid accesul la declaraţii de funcţii, variabile, macrodefiniţii şi codul poate fi parcurs facil. De asemenea, faptul că se poate detecta ce zone de cod sunt afectate în momentul modificării unei variabile sau funcţii prezintă un real ajutor în faza de dezvoltare şi debug.

Întrucât facilităţile oferite de LXR sunt evidente, s-au indexat sursele kernel-ului Linux până la versiunea 2.6.31.6 si a celui de Windows WRK-1.2 pentru a putea fi uşor parcurse aici.

Windows Research Kernel (WRK)

Windows Research Kernel (WRK) este parte a Windows Academic Program si ofera accesul la codul sursa pentru kernel-ul de Windows in medii academice. Din aceeasi initiativa Microsoft mai fac parte si Windows OS Internals Curriculum Resource Kit si Project Oz.

WRK contine partile importante din nucleul pentru Windows XP x64 si Windows 2003 SP1 impreuna cu utilitare necesare pentru compilarea si testarea unor versiuni proprii de nucleu. Sursele incluse se refera la subsisteme precum procese, thread-uri, planificator, I/O manager, memorie virtuala.

Sursele si utilitare pot fi folosite numai in scopuri non-comerciale conform licentei.

In acest moment, sursele nucleului pus la dispozitie pot fi compilate si testare numai pe sisteme cu Windows 2003 x86/x64 sau Windows XP x64.

Puteti descarca imaginea CD-ului cu Windows Academic Program de aici. Inainte de folosirea imaginii de CD va rugam sa cititi licenţa.

gdb (Linux)

Depanarea unui kernel este un proces mult mai dificil decat depanarea unui program, pentru ca nu exista tocmai suportul sistemului de operare. De aceea, acest lucru se realizeaza de obicei prin intermediul a doua calculatoare conectate pe interfetele seriale.

Alternativ, o metoda de debug mai simpla, dar cu multe lipsuri este depanarea locala folosind gdb, imaginea de kernel nearhivata (`vmlinux`) si `/proc/kcore` (imaginea in timp real a kernel-ului). Aceasta metoda este folosita de obicei pentru inspectia kernel-ului si detectarea anumitor inconsistente in timp ce acesta ruleaza. Metoda este utila mai ales daca s-a compilat kernel-ul cu optiunea `-g` de pastrare a informatiilor de debug. Nu pot fi folosite facilitatile de debug cunoscute cum sunt stabilirea de breakpoint-uri sau modificarea datelor.

Imaginea de kernel nearhivata ofera informatii pretioase despre structurile de date si simbolurile existente:

```
# cd /usr/src/linux
# file vmlinux
vmlinux: ELF 32-bit LSB executable, Intel 80386, ...
# nm vmlinux | grep sys_call_table
c02e535c R sys_call_table
# cat System.map | grep sys_call_table
c02e535c R sys_call_table
```

Utilitarul `nm` este folosit pentru afisarea simbolurilor dintr-un cod obiect sau executabil. In cazul nostru vmlinux este un fisier ELF. Alternativ se poate folosi `System.map` pentru afisarea informatiilor despre simbolurile din kernel.

Apoi folosim `gdb` pentru a inspecta simbolurile folosind imaginea nearhivata de kernel. O sesiune simpla de `gdb` este urmatoarea:

```
# cd /usr/src/linux
# gdb --quiet vmlinux
Using host libthread_db library "/lib/tls/libthread_db.so.1".
(gdb) x/x 0xc02e535c
0xc02e535c <sys_call_table>:    0xc011bc58
(gdb) x/16 0xc02e535c
0xc02e535c <sys_call_table>:    0xc011bc58      0xc011482a      0xc01013d3      0xc014363d
0xc02e536c <sys_call_table+16>: 0xc014369f      0xc0142d4e      0xc0142de5      0xc011548b
0xc02e537c <sys_call_table+32>: 0xc0142d7d      0xc01507a1      0xc015042c      0xc0101431
0xc02e538c <sys_call_table+48>: 0xc014249e      0xc0115c6c      0xc014fee7      0xc0142725
(gdb) x/x sys_call_table
0xc011bc58 <sys_restart_syscall>:       0xffe000ba
(gdb) x/x &sys_call_table
0xc02e535c <sys_call_table>:    0xc011bc58
(gdb) x/16 &sys_call_table
0xc02e535c <sys_call_table>:    0xc011bc58      0xc011482a      0xc01013d3      0xc014363d
0xc02e536c <sys_call_table+16>: 0xc014369f      0xc0142d4e      0xc0142de5      0xc011548b
0xc02e537c <sys_call_table+32>: 0xc0142d7d      0xc01507a1      0xc015042c      0xc0101431
0xc02e538c <sys_call_table+48>: 0xc014249e      0xc0115c6c      0xc014fee7      0xc0142725
(gdb) x/x sys_fork
0xc01013d3 <sys_fork>:  0x3824548b
```

```
(gdb) disass sys_fork
Dump of assembler code for function sys_fork:
0xc01013d3 <sys_fork+0>:        mov    0x38(%esp),%edx
0xc01013d7 <sys_fork+4>:        mov
```

```
# gdb /usr/src/linux/vmlinux /proc/kcore
Core was generated by `root=/dev/hda3 ro'.
#0  0x00000000 in ?? ()
(gdb) p sys_call_table
 class="code" = -1072579496
(gdb) p /x sys_call_table
# cd /usr/src/linux # gdb --quiet vmlinux Using host libthread_db library "/lib/tls/libthread_db.so.1". (gdb) x/x
0xc02e535c 0xc02e535c : 0xc011bc58 (gdb) x/16 0xc02e535c 0xc02e535c : 0xc011bc58 0xc011482a 0xc01013d3 0xc014363d
0xc02e536c : 0xc014369f 0xc0142d4e 0xc0142de5 0xc011548b 0xc02e537c : 0xc0142d7d 0xc01507a1 0xc015042c 0xc0101431
0xc02e538c : 0xc014249e 0xc0115c6c 0xc014fee7 0xc0142725 (gdb) x/x sys_call_table 0xc011bc58 : 0xffe000ba (gdb) x/x
&sys_call_table 0xc02e535c : 0xc011bc58 (gdb) x/16 &sys_call_table 0xc02e535c : 0xc011bc58 0xc011482a 0xc01013d3
0xc014363d 0xc02e536c : 0xc014369f 0xc0142d4e 0xc0142de5 0xc011548b 0xc02e537c : 0xc0142d7d 0xc01507a1 0xc015042c
0xc0101431 0xc02e538c : 0xc014249e 0xc0115c6c 0xc014fee7 0xc0142725 (gdb) x/x sys_fork 0xc01013d3 : 0x3824548b (gdb)
disass sys_fork Dump of assembler code for function sys_fork: 0xc01013d3 : mov 0x38(%esp),%edx 0xc01013d7 : mov
$0x11,%eax 0xc01013dc : push $0x0 0xc01013de : push $0x0 0xc01013e0 : push $0x0 0xc01013e2 : lea 0x10(%esp),%ecx
0xc01013e6 : call 0xc0111aab  0xc01013eb : add $0xc,%esp 0xc01013ee : ret End of assembler dump. = 0xc011bc58
(gdb) p /x &sys_call_table
 = 0xc02e535c
(gdb) x/16 &sys_call_table
0xc02e535c <sys_call_table>:    0xc011bc58      0xc011482a      0xc01013d3      0xc014363d
0xc02e536c <sys_call_table+16>: 0xc014369f      0xc0142d4e      0xc0142de5      0xc011548b
0xc02e537c <sys_call_table+32>: 0xc0142d7d      0xc01507a1      0xc015042c      0xc0101431
0xc02e538c <sys_call_table+48>: 0xc014249e      0xc0115c6c      0xc014fee7      0xc0142725
```

```
x11,%eax
0xc01013dc <sys_fork+9>:        push
```

```
C:\Documents and Settings\Administrator>livekd
LiveKd v3.0 - Execute i386kd/windbg/dumpchk on a live system
Sysinternals - www.sysinternals.com
Copyright (C) 2000-2005 Mark Russinovich

Launching C:\program files\Debugging Tools for Windows\kd.exe:
...
kd> dd nt!ExAllocatePool
80809627  8b55ff8b 6f4e68ec 75ff656e 0875ff0c
80809637  08be3ee8 08c25d00 50535300 01f08fe8
80809647  5a09e900 c0330000 0059b8e9 90909000
80809657  ffffff90 94d861ff 94d87480 90909080
80809667  ffffff90 96410cff 96411f80 90909080
80809677  ffffff90 96427cff 96429280 90909080
80809687  ffffff90 9642d9ff 9642ec80 90909080
80809697  ff8b9090 80ec8b55 89f7cc3d 840f0080
kd> u nt!ExAllocatePool
nt!ExAllocatePool:
80809627 8bff            mov     edi,edi
80809629 55              push    ebp
8080962a 8bec            mov     ebp,esp
8080962c 684e6f6e65      push    656E6F4Eh
80809631 ff750c          push    dword ptr [ebp+0Ch]
80809634 ff7508          push    dword ptr [ebp+8]
80809637 e83ebe0800      call    nt!ExAllocatePoolWithTag (8089547a)
8080963c 5d              pop     ebp
kd> dd nt!KiTimerExpireDpc
808a81c0  00200113 00000000 00000000 8081fc62
808a81d0  00000000 00000000 00000000 00000000
808a81e0  808a81e0 808a81e0 00000000 00000000
808a81f0  808a81f0 808a81f0 00000000 00000000
808a8200  00000000 00000000 808a8208 808a8208
808a8210  817a62c0 00000000 00000000 00000000
808a8220  00040001 00000000 817a6368 817a6368
808a8230  00000000 00000000 00000000 00000000
kd> dt nt!_KDPC

   +0x000 Type             : UChar
   +0x001 Importance       : UChar
   +0x002 Number           : UChar
   +0x003 Expedite         : UChar
   +0x004 DpcListEntry     : _LIST_ENTRY
   +0x00c DeferredRoutine  : Ptr32
   +0x010 DeferredContext  : Ptr32 Void
   +0x014 SystemArgument1  : Ptr32 Void
   +0x018 SystemArgument2  : Ptr32 Void
   +0x01c DpcData          : Ptr32 Void
kd> dt nt!_KDPC 808a81c0

   +0x000 Type             : 0x13 ''
   +0x001 Importance       : 0x1 ''
```

```
   +0x002 Number           : 0x20 ' '
   +0x003 Expedite         : 0 ''
   +0x004 DpcListEntry     : _LIST_ENTRY [ 0x0 - 0x0 ]
   +0x00c DeferredRoutine  : 0x8081fc62     nt!KiTimerExpiration+0
   +0x010 DeferredContext  : (null)
   +0x014 SystemArgument1  : (null)
   +0x018 SystemArgument2  : (null)
   +0x01c DpcData          : (null)
kd> dd nt!KiTimerExpiration
8081fc62  8b55ff8b 94ec81ec fa000000 18a100eb
8081fc72  8bffdf00 df00140d 1c053bff 89ffdf00
8081fc82  4d89e445 07850fe0 a10000c7 ffdf000c
8081fc92  00080d8b 053bffdf ffdf0010 89ec4589
8081fca2  850fe84d 0000c6f1 e7fc0d8b 8bfb8089
8081fcb2  d18b1045 fa81d02b 00000200 5a3d830f
8081fcc2  65830003 565300f4 01ffe181 8d570000
8081fcd2  4d89ff70 f845c7cc 00000018 04fc45c7
kd> u nt!KiTimerExpiration
nt!KiTimerExpiration:
8081fc62 8bff            mov     edi,edi
8081fc64 55              push    ebp
8081fc65 8bec            mov     ebp,esp
8081fc67 81ec94000000    sub     esp,94h
8081fc6d fa              cli
8081fc6e eb00            jmp     nt!KiTimerExpiration+0x10 (8081fc70)
8081fc70 a11800dfff      mov     eax,dword ptr ds:[FFDF0018h]
8081fc75 8b0d1400dfff    mov     ecx,dword ptr ds:[0FFDF0014h]
kd> q
quit:
Execute Kd again? (y/n) n
Exiting LiveKd.
C:\Documents and Settings\Administrator>
x0
0xc01013de <sys_fork+11>:       push
printk(KERN_NOTICE "my new kernel is booting\n");
x0
0xc01013e0 <sys_fork+13>:       push
 root=/dev/sda1 rw init=/bin/bash
x0
0xc01013e2 <sys_fork+15>:       lea    0x10(%esp),%ecx
0xc01013e6 <sys_fork+19>:       call   0xc0111aab <do_fork>
0xc01013eb <sys_fork+24>:       add
mount -t proc proc /proc
xc,%esp
0xc01013ee <sys_fork+27>:       ret
End of assembler dump.
```

Se observa ca s-a folosit ca parametru pentru `gdb` imaginea de kernel nearhivata care rezida in radacina surselor dupa compilare.

Cateva comenzi utilizate pentru debugging cu gdb sunt:

- x - este folosita pentru afisarea continutului zonei de memorie a carei adresa este primita ca parametru (aceasta adresa poate fi valoarea unei adrese fizice, un simbol sau adresa unui simbol); poate primi ca parametri (precedati de `/`): formatul in care afiseaza datele (`x` pentru hexazecimal, `d` pentru zecimal, etc.), cate unitati de memorie se afiseaza si dimensiunea unei unitati de memorie.
- disassemble - este folosita pentru dezasamblarea unei functii.
- p - este folosita pentru evaluarea si afisarea valorii unei expresii; se poate specifica formatul in care se afiseaza datele (`/x` pentru hexazecimal, `/d` pentru zecimal, etc.).

Analiza imaginii de kernel este o analiza statica. Daca dorim o analiza dinamica (o analiza a kernel-ului asa cum ruleaza el) vom folosi `/proc/kcore`; acesta este o imagine dinamica (in memorie) a kernel-ului.

```
# gdb /usr/src/linux/vmlinux /proc/kcore Core was generated by `root=/dev/hda3 ro'. #0 0x00000000 in ?? () (gdb) p sys_call_table $1 = -1072579496 (gdb) p /x sys_call_table $2 = 0xc011bc58 (gdb) p /x &sys_call_table $3 = 0xc02e535c (gdb) x/16 &sys_call_table 0xc02e535c : 0xc011bc58 0xc011482a 0xc01013d3 0xc014363d 0xc02e536c : 0xc014369f 0xc0142d4e 0xc0142de5 0xc011548b 0xc02e537c : 0xc0142d7d 0xc01507a1 0xc015042c 0xc0101431 0xc02e538c : 0xc014249e 0xc0115c6c 0xc014fee7 0xc0142725
```

Folosirea imaginii dinamice a kernel-ului este utila pentru detectarea de rootkit-uri.

- Linux Device Drivers 3rd Edition - Debuggers and Related Tools
- Kernel Debugging with DDD/GDB Run Locally
- Detecting Rootkits and Kernel-level Compromises in Linux

- User-Mode Linux
- GDB User Manual

LiveKd (Windows)

Asemanator cu gdb, pe Windows exista posibilitatea analizei dinamice a kernel-ului folosind Live Kernel Debugger - LiveKd. Pachetul este oferit de SysInternals. Pentru lucrul cu acesta aveţi nevoie de pachetul Debugging Tools de la Microsoft. Acesta conţine debugger-e peste care lucrează livekd, cum ar fi **kd** sau **WinDbg**. Aceste debugger-e sunt, de obicei, folosite în cadrul unui sistem de două calculatoare legate printr-un cablu serial: un calculator pe care se face debug iar altul care realizeaza debug-ul. LiveKd permite inspecţia kernel-ului în timp ce acesta rulează.

LiveKd foloseşte ca backend `kd`, `WinDbg` sau `Dumpchk` (`kd` este implicit). Inainte de pornire, Live Kd va inspecta configuraţia curentă pentru a detecta prezenţa simbolurilor de debug ale sistemului de operare; dacă acestea nu sunt prezente se va cere să fie descărcate. Deosebirea între `kd` şi `WinDbg` este aceea că `WinDbg` permite configurare prin interfaţa grafică; setul de comenzi de depanare este, însă, apropiat. Un exemplu de sesiune LiveKd este prezentată mai jos:

```
C:\Documents and Settings\Administrator>livekd LiveKd v3.0 - Execute i386kd/windbg/dumpchk on a live system Sysinternals - www.sysinternals.com Copyright (C) 2000-2005 Mark Russinovich Launching C:\program files\Debugging Tools for Windows\kd.exe: ... kd> dd nt!ExAllocatePool 80809627 8b55ff8b 6f4e68ec 75ff656e 0875ff0c 80809637 08be3ee8 08c25d00
50535300 01f08fe8 80809647 5a09e900 c0330000 0059b8e9 90909000 80809657 ffffff90 94d861ff 94d87480 90909080 80809667
ffffff90 96410cff 96411f80 90909080 80809677 ffffff90 96427cff 96429280 90909080 80809687 ffffff90 9642d9ff 9642ec80
90909080 80809697 ff8b9090 80ec8b55 89f7cc3d 840f0080 kd> u nt!ExAllocatePool nt!ExAllocatePool: 80809627 8bff mov edi,edi 80809629 55 push ebp 8080962a 8bec mov ebp,esp 8080962c 684e6f6e65 push 656E6F4Eh 80809631 ff750c push dword ptr [ebp+0Ch] 80809634 ff7508 push dword ptr [ebp+8] 80809637 e83ebe0800 call nt!ExAllocatePoolWithTag (8089547a) 8080963c 5d pop ebp kd> dd nt!KiTimerExpireDpc 808a81c0 00200113 00000000 00000000 8081fc62 808a81d0 00000000 00000000 00000000
00000000 808a81e0 808a81e0 808a81e0 00000000 00000000 808a81f0 808a81f0 808a81f0 00000000 00000000 808a8200 00000000
00000000 808a8208 808a8208 808a8210 817a62c0 00000000 00000000 00000000 808a8220 00040001 00000000 817a6368 817a6368
808a8230 00000000 00000000 00000000 00000000 kd> dt nt!_KDPC +0x000 Type : UChar +0x001 Importance : UChar +0x002 Number : UChar +0x003 Expedite : UChar +0x004 DpcListEntry : _LIST_ENTRY +0x00c DeferredRoutine : Ptr32 +0x010 DeferredContext : Ptr32 Void +0x014 SystemArgument1 : Ptr32 Void +0x018 SystemArgument2 : Ptr32 Void +0x01c DpcData : Ptr32 Void kd> dt nt!_KDPC 808a81c0 +0x000 Type : 0x13 '' +0x001 Importance : 0x1 '' +0x002 Number : 0x20 ' ' +0x003 Expedite : 0 '' +0x004 DpcListEntry : _LIST_ENTRY [ 0x0 - 0x0 ] +0x00c DeferredRoutine : 0x8081fc62 nt!KiTimerExpiration+0 +0x010 DeferredContext : (null) +0x014 SystemArgument1 : (null) +0x018 SystemArgument2 : (null) +0x01c DpcData : (null) kd> dd nt!KiTimerExpiration 8081fc62 8b55ff8b 94ec81ec fa000000 18a100eb 8081fc72 8bffdf00 df00140d 1c053bff 89ffdf00 8081fc82
4d89e445 07850fe0 a10000c7 ffdf000c 8081fc92 00080d8b 053bffdf ffdf0010 89ec4589 8081fca2 850fe84d 0000c6f1 e7fc0d8b
8bfb8089 8081fcb2 d18b1045 fa81d02b 00000200 5a3d830f 8081fcc2 65830003 565300f4 01ffe181 8d570000 8081fcd2 4d89ff70
f845c7cc 00000018 04fc45c7 kd> u nt!KiTimerExpiration nt!KiTimerExpiration: 8081fc62 8bff mov edi,edi 8081fc64 55 push ebp 8081fc65 8bec mov ebp,esp 8081fc67 81ec94000000 sub esp,94h 8081fc6d fa cli 8081fc6e eb00 jmp nt!KiTimerExpiration+0x10 (8081fc70) 8081fc70 a11800dfff mov eax,dword ptr ds:[FFDF0018h] 8081fc75 8b0d1400dfff mov ecx,dword ptr ds:[0FFDF0014h] kd> q quit: Execute Kd again? (y/n) n Exiting LiveKd. C:\Documents and Settings\Administrator>
```

Comenzile utilizate in cadrul acestei sesiuni de debug sunt:

- `dt` (dump using type information) - permite analiza tipurilor de date din sistem (cum este spre exemplu `KDPC`)
- `dd` (dump memory) - permite analiza diverselor variabile, precizând conţinutul unei anumite locaţii de memorie; poate primi ca parametru o adresă sau un simbol (asemănător cu comanda `x` de la `gdb`)
- `u` (unassemble) - permite dezasamblarea codului unei funcţii

Un simbol poate fi parte a unui domeniu. Simbolurile de kernel sunt precedate de `nt!` pentru a se specifica faptul că sunt simboluri de kernel.

Informaţii complete despre funcţionarea depanatorului şi comenzile utilizate găsiţi în documentaţia asociată (`Debugging Help.chm`).

- Navigating The Kernel Debugger

Integrare LiveKd în WinDbg

Una dintre cele mai uşoare forme de utilizare a LiveKd este prin intermediul WinDbg, care oferă interfaţă grafică pentru depanarea nucleului Windows.

Folosirea LiveKd în WinDbg înseamnă activarea opţiunii `Local Debugging`. Pentru aceasta se accesează secvenţa de meniuri: `File -> Kernel Debug (CTRL+K) -> Local`.

Documentatie

Dezvoltarea kernel-ului are un grad sporit de dificultate raportat la programarea din user space. API-ul diferit, necesitatea cunoasterii amanuntite a sistemului pe care se lucreaza si a structurii kernel-ului necesita o etapa de pregatire suplimentara. Documentatia asociata este destul de eterogena, fiind nevoie de inspectia mai multor surse pentru a avea o intelegere completa a unui aspect.

Documentatie Linux

Principalele avantaje ale kernel-ului Linux sunt accesul la surse si sistemul deschis de dezvoltare. Drept urmare, Internet-ul ofera un numar mult mai mare de resurse de documentare a kernel-ului.

Cateva link-uri utile sunt prezentate mai jos:

- KernelNewbies
- KernelNewbies - Kernel Hacking
- Kernel Analysis - HOWTO
- Linux Kernel Programming
- Linux kernel - Wikibooks

Link-urile nu sunt nicidecum exhaustive. Folosirea Internet-ului si a surselor este esentiala.

Documentatie Windows

Documentatia principala de Windows este cea care vine cu Windows DDK (Driver Development Kit). Alternativ ea poate fi accesata de pe site-ul Microsoft. Informatii si utilitare se pot regasi si la SysInternals, site creat de Mark Russinovich (unul din autorii Inside Windows, 4th Edition) sau la Code Project. Desi cu mai putine resurse de documentatie decat kernel-ul Linux, Internet-ul ofera multe informatii.

In mod evident, nu trebuie ignorat accesul la codul sursa, care poate oferi informatii acolo unde documentatia nu este suficienta.

Quiz

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

Exerciţii

Precizări

- Exerciţiile legate de LXR nu necesită folosirea maşinilor virtuale (doar accesul la un browser).
- În rest, exerciţiile de Linux vor fi realizate în cadrul **maşinii virtuale** de Linux, iar cele de Windows în cadrul **maşinii virtuale** de Windows.

- 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. Conturile maşinilor virtuale sunt:
    - Linux: root/student, student/student (`ssh student@spook.local`)
    - Windows: Administrator/student, student/student (`ssh student@chooch.local`)

Linux

1. Bootaţi sistemul de pe imaginile de kernel de aici.
   1. Descărcaţi imaginea în directorul `/boot/`.
      - **Hints**:
        - Se recomandă folosirea `wget` direct din maşina virtuală de Linux.
          - Alternativa este descărcarea imaginii în sistemul gazdă şi copierea prin SSH pe maşina virtuală. Alternativa

durează mai mult
- Vă puteţi conecta la maşina virtuală de Linux prin SSH folosind comanda `ssh -l root spook.local` (parola `student`)

2. Ce nucleu conţine fiecare dintre cele două imagini?
   - **Hints**:
     - Comanda `file`.
3. Adaugaţi **o** nouă intrare în fişierul de configurare al GRUB-ului care să permită bootarea de pe noul kernel `mybzImage.bad`. Denumirea care trebuie să apară în ecranul de boot GRUB să fie *MyKernel*.
   - **Hints**:
     - `/etc/grub.d/40_custom`
     - `# update-grub`
     - nu aveţi nevoie de o imagine `initrd`
     - partiţia root este `/dev/sda1`
     - Configurare GRUB
4. Reporniţi sistemul.
   - În ecranul GRUB selectaţi opţiunea *MyKernel*.
5. Care este cauza erorii de bootare? (`Kernel panic`)
   - **Hints**:
     - Urmăriţi mesajul afişat. Ce sistem de fişiere se găseşte pe partiţia rădăcină?
     - Dacă nu obţineţi eroarea este posibil să fi bootat altă imagine, imaginea deja existentă sau `mybzImage.good`.
6. Reporniţi sistemul.
   - În ecranul GRUB selectaţi şi editaţi opţiunea *MyKernel* pentru a boota `mybzImage.good`
   - **Hints**:
     - Apăsaţi `e`, editaţi apoi `Ctrl-X`.
     - **NU** ieşiţi din meniul de editare.
7. Sistemul ar trebui să funcţioneze
   - Care este diferenţa dintre cele două imagini?
   - **Hints**:
     - `diff`

2. Pregătiţi nucleul Linux pentru compilare.
   1. Descărcaţi arhiva cu sursele nucleului Linux versiunea 2.6.37 în `/usr/src/`.
      - Recomandăm `wget` (fie FTP, fie HTTP).
      - Alternativa este descărcarea imaginii în sistemul gazdă şi copierea prin SSH pe maşina virtuală. Alternativa durează mai mult
        - Vă puteţi conecta la maşina virtuală de Linux prin SSH folosind comanda `ssh -l root spook.local` (parola `student`)
   2. Dezarhivaţi sursele nucleului.
      - **Hints**:
        - NU folosiţi opţiunea `v` (verbose) la dezarhivare. Va afişa la consolă informaţii despre dezarhivare care nu sunt neapărat necesare.
   3. Creaţi legătura simbolică `/usr/src/linux` către sursele dezarhivate ale nucleului.
   - **Hints**:
     - Citiţi secţiunea Obtinerea surselor din laborator.
3. Personalizaţi nucleul Linux înainte de compilare.
   1. Adăugaţi, în cadrul funcţiei `start_kernel` din `init/main.c`, un mesaj de formaprintk(KERN_NOTICE "my new kernel is booting\n");
      - **Hints**:
        - NU este virgulă între `KERN_NOTICE` şi `"my new kernel is booting\n"`. Pentru mai multe informaţii consultaţi laboratorul 2 şi laboratorul 3.
        - Adaugaţi linia în corpul funcţiei `start_kernel` înainte de apelul funcţiei `rest_init`.
4. Compilaţi sursele kernel-ului de Linux.
   1. Rulaţi `ARCH=i386 make allnoconfig`
      - Ce face comanda de mai sus?
        - **Hint**: `make help`
   2. Rulaţi `make menuconfig`
   3. Configuraţi după cum urmează:
      1. *Enable the Block Layer* ⇒ *Support for large (2TB+) block devices and files* **y**
      2. *Bus options (PCI etc.)* ⇒ { *PCI support* **=y** şi *Support for PCI Hotplug* **=y** }
      3. *Executable File formats* ⇒ *Kernel support for ELF files* **=y**
      4. *Device Drivers* ⇒ *Serial ATA and Parallel ATA drivers* **y** ⇒ *Intel ESB, ICH, PIIX3, PIIX4 PATA/SATA support* **y**
      5. *Device Drivers* ⇒ *SCSI device support* ⇒ *SCSI disk support* **y**
      6. *File systems* ⇒ { *The extentended 4 (ext4)* **y**, *Ext 4 POSIX Access Control List* **y**, *Ext 4 Security Labels* **y**}
      7. *File systems* ⇒ *Pseudo filesystems* ⇒ */proc/kcore support* **y**
      8. *Kernel hacking* ⇒ {*Kernel debugging* **y**, *Compile the kernel with debug info* **y**, *Compile the kernel with frame pointers* **y**}
   4. Compilaţi nucleul pentru obţinerea unei imagini tip `bzImage`.

- **Hint:**
  - Compilarea va dura aproxamativ **8 minute**, puteţi continua cu exerciţiile de pe Windows
5. Compilaţi modulele nucleului.
   - De ce aţi obţinut eroare?
   - **Ignoraţi** eroarea.
   - Citiţi secţiunile Configurare şi Compilare din laborator.

5. Bootaţi de pe nucleul proaspăt compilat.
   1. Instalaţi imaginea kernel-ului.
      - **Hints**:
        - Folosiţi `make install`.
   2. Configuraţi GRUB pentru a permite boot-area de pe noul nucleu. Titlul asociat noului nucleu în ecranul de boot al GRUB trebuie să fie *SO2-rules Linux kernel 2.6.37 compilat de #your name here#*
   3. Adăugaţi următoarele opţiuni la kernel: root=/dev/sda1 rw init=/bin/bash Ce face opţiunea `init`?
      - **Hints**:
        - Puteţi porni de la fişierul `40_custom`, configurat la punctul **1**:
        - `cp /etc/grub.d/40_custom /etc/grub.d/50_custom`
        - Editaţi fişierul şi actualizaţi configuraţia GRUB-ului.
   4. Reporniţi sistemul.
   5. În ecranul GRUB optaţi pentru bootare de pe nucleul proaspăt compilat.
   6. După bootare verificaţi versiunea nucleului şi apariţia mesajului personalizat de la exerciţiul **3**.
      - **Hints**:
        - Folosiţi comanda `uname -a` pentru a verfica versiunea nucleului.
        - Folosiţi comanda `dmesg` (eventual înlănţuită cu `grep`) pentru a vizualiza mesajele afişate de kernel la boot.
   - **Hints**:
     - Citiţi secţiunea Instalare din laborator.

Windows

1. Folosiţi Windows Live Debugger (`livekd`) prin intermediul WinDbg pentru:
   - a afla adresa funcţiei `PsCreateSystemThread` şi a dezasambla această funcţie;
   - a afla adresa structurilor `KeServiceDescriptorTable` şi `KeServiceDescriptorTableShadow`.
   - **Hints**:
     - Citiţi secţiunea LiveKd (Windows) din laborator.
2. Folosiţi LXR pe Windows pentru a determina locul în care sunt definite/declarate anumite structuri/funcţii.
   1. Determinaţi fişierul în care sunt declarate următoarele tipuri de date:
      - `EPROCESS`
      - `DRIVER_OBJECT`
      - `FAST_MUTEX`
      - `IRP`
   2. Determinaţi fişierul în care sunt declarate structurile:
      - `KeServiceDescriptorTable`
      - `KeServiceDescriptorTableShadow`
   3. Determinaţi fişierul în care sunt declarate funcţiile:
      - `InterlockedExchange`
      - `ExAllocatePoolWithTag`
      - `IoRequestDpc`
      - `PsSetCreateProcessNotifyRoutine`
   - **Hints**:
     - Citiţi secţiunea Windows Research Kernel (WRK) din laborator.

Extra

Pentru putea folosi `/proc/kcore` şi pentru a avea informaţiile de debug în nucleu, bootaţi nucleul compilat la punctul **4** folosind intrarea din meniul GRUB adăugată la punctul **5**.

În consola apărută montaţi sistemul de fişiere `procfs`:

```
mount -t proc proc /proc
```

1. Folosiţi `gdb` şi `/proc/kcore` pentru a obţine urmatoarele informaţii:
   - adresa variabilei `jiffies` în memorie şi conţinutul acesteia;
   - dezasamblarea funcţiei care creează thread-uri de kernel (`kernel_thread`)
   - **Hints**:
     - Citiţi secţiunea gdb (Linux) din laborator.
2. Folosiţi LXR pe Linux pentru a determina locul anumitor informaţii.
   1. Determinaţi fişierul în care sunt declarate următoarele tipuri de date:
      - `struct task_struct`
      - `struct semaphore`
      - `struct list_head`
      - `spinlock_t`
      - **Hints**:
        - Pentru o structură se caută doar numele ei. Spre exemplu, în cazul `struct task_struct` se caută şirul `task_struct`.
   2. Determinaţi fişierul în care sunt declarate următoarele simboluri globale:
      - `sys_call_table`
      - `jiffies`
      - `current`
   3. Determinaţi fişierul în care sunt declarate următoarele funcţii:
      - `copy_from_user`
      - `vmalloc`
      - `schedule_timeout`
      - `add_timer`
   - **Hints**:
     - Citiţi secţiunea LXR Cross-Reference din laborator.
3. Compilaţi sursele nucleului de Windows din cadrul proiectului WRK şi boot-aţi de pe noul kernel.
   - **Notă**: Acest exerciţiu nu este realizabil pe maşina virtuală de SO2, întrucât foloseşte Visual Studio 9; pentru compilarea nucleului WRK este nevoie de Visual Studio 8.
   1. Compilaţi nucleul pentru obţinerea unei imagini pentru arhitectura x86.
      - **Hints**:
        - Sursele sunt disponibile în cadrul imaginii VMware în directorul `C:\cygwin\home\Administrator\so2\WRK\WRK-v1.2`.
        - Citiţi secţiunea Compilare din laborator.
   2. Instalaţi imaginea de kernel.
      - **Hints**:
        - Copiaţi imaginea de nucleu în `C:\WINDOWS\system32` cu denumirea `wrk-so2.exe`.
        - Găsiţi imaginea corectă de HAL şi copiaţi-o în `C:\WINDOWS\system32`.
        - **ATENŢIE**: numele pentru imaginea de nucleu şi imaginea de HAL sunt nume de fisier scurte.
        - Citiţi secţiunea Instalare din laborator.
   3. Bootaţi de pe nucleul proaspăt compilat.
      - **Hints**:
        - Configuraţi `boot.ini` pentru a permite bootarea de pe noul nucleu. Titlul asociat noului nucleu în ecranul de boot trebuie să fie *SO2-rules Windows Server 2003 WRK 1.2 compilat de #your name here#*.
        - Reporniţi sistemul.
        - În ecranul de boot optaţi pentru bootare de pe nucleul proaspăt compilat.
        - După bootare verificaţi versiunea nucleului cu ajutorul comenzii ver.
        - Citiţi secţiunea Configurare boot.ini din laborator.

Soluţii

- Soluţii exerciţii laborator 1

Observaţii

- `make` = `make bzImage && make modules`