Percorso 7:
- Linux Advanced Administration

Fulvio Corno
fulvio.corno@polito.it

Bartolomeo Montrucchio
bartolomeo.montrucchio@polito.it

Comandi di base UNIX

- cd, cp, rm, mkdir, mv
- tar, gzip, kill, bg, fg
- chmod, chown
- sudo, su
- redirezione, >, >>, |
- pagine di manuale, man

Esercizio

- Copiare una directory da un punto ad un altro del filesystem utilizzando tar prima per unire i file e poi per dividerli (comando su di una unica riga)
- Realizzare un breve script bash in grado di individuare i processi (in corso) di proprietà dell'utente ed il cui nome inizia con prova e terminarli
  - realizzare in precedenza dei processi adeguati, preferibilmente compilando un piccolo programma C con una fork()

VI

- È l'editor di base sui sistemi Unix
- È di fatto una macchina a (2) stati:
  - stato modifica (inserimento)
  - stato comandi (cancellazione, movimento)
- Il VIM (versione improved) è più comodo
- Il VI è indispensabile per un sistemista
  - in taluni casi è l'unico editor disponibile

Esercizio

- Provare a creare un file con vi
- Provare il copia incolla
- Provare a cancellare
- Infine provare a salvare ed uscire

Linux kernel

- Ultima versione stabile 3.16.3 (20 settembre 2014)
  - 3.16 major release number
  - 3 minor release number
- Il kernel di Linux è costruito in modo tale da essere indipendente dalla piattaforma
  - Nel codice del kernel non si possono utilizzare numeri in floating point in quanto alcuni processori non l'hanno
    - la configurazione dell'emulazione floating point nel kernel riguarda solo lo user space
  - sono presenti memory barriers per garantire il corretto ordine di esecuzione delle operazioni
- L'idea è che:
  - Kernel: livello hardware
  - Shell: livello testuale
  - X Window: livello GUI

Kernel

- Gestisce tutte le risorse hardware (CPU, memoria, I/O)
  - in concorrenza (si pensi ad una interfaccia di rete usata da molteplici applicazioni nello spazio utente)
- Fornisce un insieme di system call (circa 300) per permettere all'utente l'utilizzo delle risorse hardware
  - le system call sono spesso utilizzate tramite (wrapped) la libreria C
- Il linguaggio usato è prevalentemente il C, con piccole parti in Assembly
- È possibile effettuare il download del kernel da www.kernel.org (a noi interessa la versione 3.3.0 in quanto compatibile con [1])
- Il kernel nel caso di Ubuntu è in /boot (dove è in OpenBSD?)

Esercizio

- Preparare una nuova macchina virtuale
- Effettuare il download del kernel 3.3.0
- Decomprimerlo mediante bunzip2 (o altro) in una directory quale /usr/src/3.3.0
  - anche se in questa fase non sarebbe necessario, lavorare come root o utilizzare sudo su; usando sudo su – si potrebbero avere problemi con il display X
- Andare a guardare la directory arch/
  - Quali sono le principali implementazioni?
    - arm, alpha, powerpc, x86
- Quali sono le dimensioni del kernel? Quali quelle dei sorgenti (usare du)?

Funzioni di libreria

- Lo spazio utente (user space) è implementato al di sopra dei servizi del kernel, non viceversa [2]
- Macchina a stati (292-298 di [2])
- Il kernel deve pertanto avere proprie implementazioni di libreria (gestione stringhe, crittografia, decompressione)
- Ad esempio NON si possono usare nel codice del kernel funzioni della libreria C standard quali printf(), memset(), malloc()
- Le funzioni equivalenti a livello kernel sono per esempio printk(), memset(), kmalloc()
- Le funzioni interne al kernel (API) sono soggette a cambiamenti tra le release del kernel
  - chi propone il cambiamento aggiorna anche il codice dei driver coinvolti
  - se però il codice del driver non è GPL ciò NON accade!
- Le system call a livello user space invece sono estremamente stabili

Memoria

- Non vi è protezione della memoria a livello di kernel
  - accessi illegali alla memoria producono blocchi del kernel
  - lo stack ha dimensione fissa (8 o 4KB) e non c'è modo di incrementarlo
  - non è possibile effettuare lo swap della memoria del kernel

Driver

- Driver:
  - Il codice può essere riutilizzato
  - Una volta accettato nell'albero di sviluppo, il driver viene mantenuto direttamente dagli sviluppatori del kernel
  - Per driver proprietari (dunque non GPL) è vietato effettuare il link statico nel kernel
    - si possono aggiungere come moduli (potrebbe comunque essere illegale, varia da caso a caso)
- Driver in User Space (in alcuni casi):
  - I driver possono essere scritti in numerosi linguaggi
  - Possono essere mantenuti proprietari (anche se è sconsigliato)
  - Il loro codice può essere fermato e sottoposto a debug, senza fermare il kernel
- Il kernel può essere considerato come una applicazione che gira sopra i device driver
- Si noti che per quanto la maggior parte dei moduli siano device drivers, essi (i moduli) non lo sono per forza
- Con una certa approssimazione un modulo utilizza il kernel come una libreria condivisa, utilizzando solo una lista di simboli e funzioni che sono stati esportati
- I moduli possono dipendere gli uni dagli altri e formare uno stack

cscope

- È un tool per la gestione del codice (soprattutto C)
- In tempi dell'ordine del minuto indicizza i sorgenti del kernel
  - make cscope (nella directory del kernel)
  - cscope
    - TAB per spostarsi tra risultati della ricerca e comandi
    - CTRL-D per uscire

Ricompilazione kernel (12.04) (I)

- Partendo dalla directory del kernel appena scaricato 3.3.0:

  make mrproper (se richiesto da precedenti compilazioni)
- installare i pacchetti eventualmente necessari (per esempio: apt-get install g++ o apt-get install libqt4-dev o apt-cache search qt4 per cercare i pacchetti mancanti)

  make xconfig
- scegliere le opzioni di configurazioni ritenute più opportune (considerando che è in uso una macchina virtuale)
- il file .config contiene la configurazione, salvarne una copia con altro nome (esiste comunque un .config.old ed esiste anche make oldconfig)
- Prestare attenzione a non sovrascrivere mai il kernel in uso
  - peraltro il procedimento dovrebbe essere automatico

Ricompilazione kernel (12.04) (II)

- make -j 4 (-j suggerito sui multicore) compila il kernel
  - ora il kernel è in arch/x86/boot/bzImage
- make modules_install provvede ad installare tutti i moduli compilati nella directory /lib/modules/3.3.0 (crea anche la directory)
- make install
- solo se richiesto:
  - copiare il kernel (arch/x86/boot/bzImage) e System.map in /boot con un nome appropriato
  - update-initramfs -c –k 3.3.0 (a cosa serve?)
  - update-grub aggiorna il file /boot/grub/grub.cfg (richiede la riscrittura dell'MBR?)
  - Modificare la configurazione grub in /etc/default/grub (poi lanciare update-grub)
    - ✓ GRUB_TIMEOUT=10 #timeout before start
    - ✓ #GRUB_HIDDEN_TIMEOUT=0 # to see the menu
- Fare reboot con la corretta versione di kernel (usare ESC)

Esercizio

- Configurare il nuovo kernel, giustificando le scelte
  - salvare il file di configurazione
- Compilare ed installare il nuovo kernel, prestando attenzione a preservare il precedente kernel
  - se necessario installare i pacchetti richiesti
  - verificare che la configurazione di grub sia corretta
- Utilizzando cscope, analizzare il codice
- Poi effettuare minime modifiche (per esempio un commento) e ricompilare il kernel
- Provare anche http://lxr.free-electrons.com (sistema di gestione del codice più sofisticato)

Esercizio

- Scaricare gli esercizi del libro [1] “Writing Linux Device Drivers” Jerry Cooperstein ISBN 978-1448672387, da: http://www.coopj.com/LDD. Copiare e verificare il file di configurazione (32 bit, PAE per il kernel 3.3.0), facendo eventuali aggiustamenti se richiesti
  - usare il file di configurazione fornito con gli esercizi (fare una copia del precedente)
- Infine compilare ed installare il nuovo kernel

Driver

- Ci sono tre tipi principali di device driver:
- Device a caratteri:
  - Accesso sequenziale, un byte alla volta, stream
  - Di fatto dei file (essi implementano open, close, read, write)
  - /dev/tty0
- Device a blocchi
  - Accesso casuale, in blocchi (con cache) (in Linux si possono usare anche come fossero a caratteri)
  - Si possono montare filesystem su questi device
  - /dev/hda1
- Device di rete
  - Trasferisce pacchetti, non stream (per lo più via socket BSD)
  - Le interfacce di rete non sono mappate sul filesystem (hanno un nome, tipo eth0, ppp0)
  - Ill kernel non usa read/write, ma funzioni per ricezione e trasmissione dei pacchetti
- SCSI, USB e User-space drivers non ricadono in questa tassonomia (ad es. per la presenza di protocolli)

printk()

- È simile alla printf(), ma lavora a livello kernel

printk(KERN_INFO “Esempio di messaggio\n”)

- KERN_INFO è uno dei livelli di log
- Non scrive su video, ma su /var/log/syslog
- Il \n è necessario se si vuole essere sicuri di vedere subito il messaggio

Installare un device driver

- Esempio a caratteri (più semplice)
- Creare un Makefile (esempio capitolo 2 di [1]) dove si ha il codice del modulo:

obj-m += lab1_chrdrv.o

- Si compila con make

make -C/lib/modules/$(uname -r)/build M=$PWD modules

- Si carica e scarica con insmod lab1_chrdrv.ko e rmmod lab1_chrdrv
- Si crea un device node (c per caratteri, 700 e 0 sono major e minor number):

mknod /dev/mycdrv c 700 0

il nome è irrilevante, il kernel usa il major number; il minor number è usato nel driver stesso

Installare un device driver

- Per testare il driver:

echo Messaggio prova device > /dev/mycdrv

- Per leggere i dati scritti:

dd if=/dev/mycdrv count=1

Esercizio

- Utilizzando l'esempio del capitolo 2 del libro "Writing Linux Device Drivers" Jerry Cooperstein ISBN 978-1448672387:
  - preparare un Makefile per il driver lab1_chrdrv
  - compilare il modulo e installarlo
  - creare il device node
  - cercare di usare il modulo scrivendo e leggendo dal device
  - apportare modifiche al codice, inserendo una printk e ricompilare
  - verificare su syslog
- Provare ad inserire il modulo nell'albero del kernel
  - Creare una directory ad es. in drivers/ con Makefile e sorgenti, inserendovi un Kconfig opportuno (ad es. simile a quello di mca)
  - Modificare il Kconfig globale e il Makefile dei driver
  - Rifare make xconfig e ricompilare

Bibliografia

- [1] Writing Linux Device Drivers, Jerry Cooperstein, Jerry Cooperstein editore
- [2] Linux Kernel and Driver Development Training, Free Electrons, http://free-electrons.com