Linux Kernel e TCP/IP Stack

Internals e kernel hacking

Luca De Cicco, Vittorio Palmisano

23, 30 Ottobre 2007

Outline

Outline

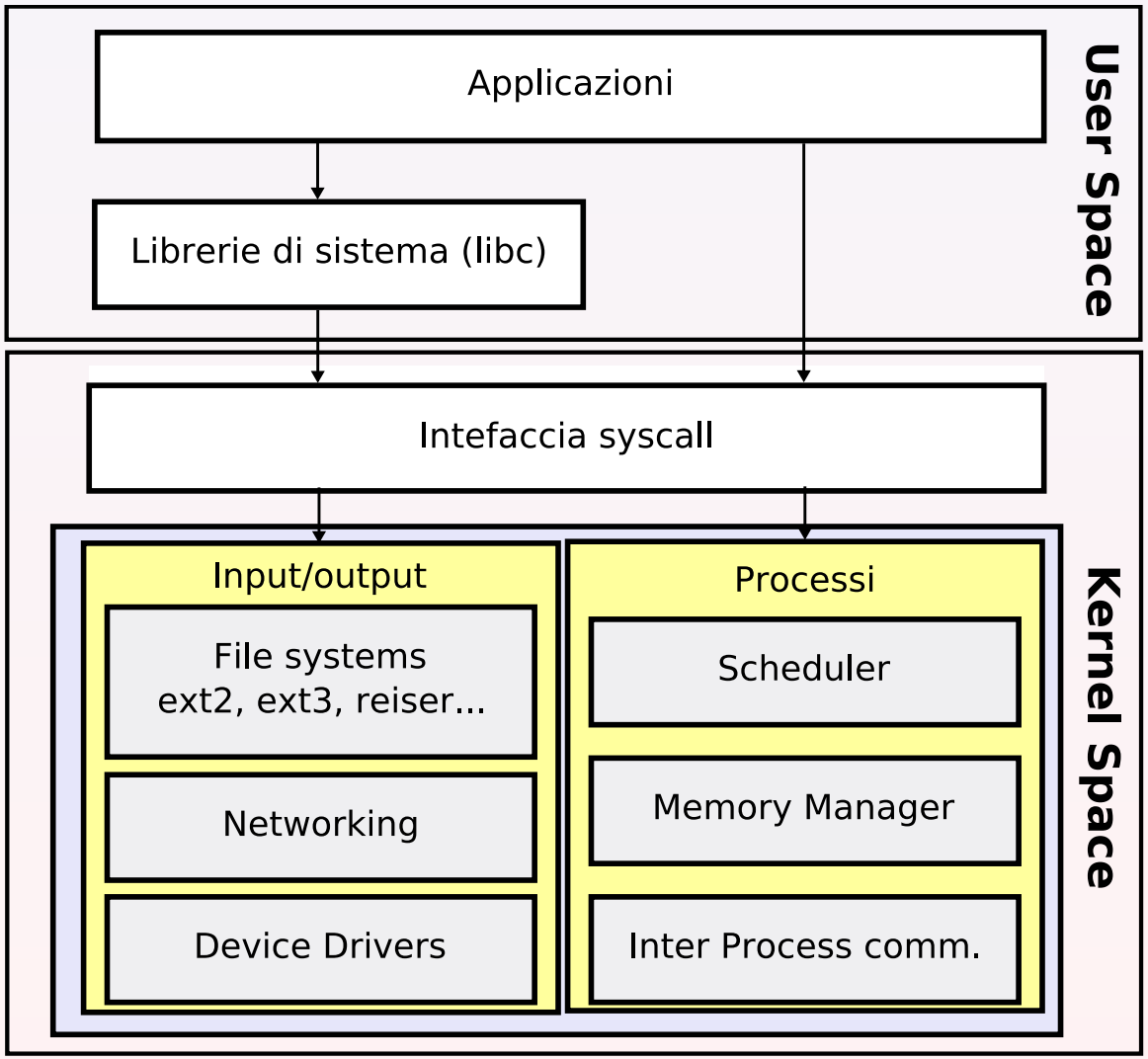
Cosa è un kernel

- Effettua il controllo e l'accesso alle risorse hardware.
- Implementa livelli di astrazione: processi, file, device
- Effettua lo scheduling ed il managment delle risorse di sistema:
  - Memoria: Virtual Memory manager
  - CPU: CPU scheduler
  - Dischi: filesystem
- Fornisce un livello di astrazione all'utente tramite le chiamate di sistema (syscall)
- **Implementa lo stack protocollare TCP/IP**

Cosa NON è

Una Shell
Una interfaccia grafica

Struttura di un kernel



"Hello GNU/Linux!"

hello.c

```
#include <stdio.h>
int main(int argc, char* argv[])
{
    printf("Hello GNU/Linux!\n");
}
```

esecuzione

```
$ gcc hello.c -o hello
$ ./hello
Hello GNU/Linux!
$
```

Dietro le quinte (1/1)

Utilizzando il comando **strace** è possibile visualizzare le chiamate di sistema:

strace ./hello

```
execve("./hello", ["./hello"], [/* 28 vars */]) = 0
/* nome_syscall(arg1,arg2, ...) = return_value */
[...]
/* Apre la libreria libc */
open("/lib/libc.so.6", O_RDONLY) = 3
read(3, "\177ELF\[...]"..., 512) = 512
[...]
/* Scrive sullo standard output (fd=1) 17 byte */
write(1, "Hello GNU/Linux!\n", 17) = 17
[...]
```

Dietro le quinte (2/2)

Guardiamo la pagina man della chiamata di sistema **write**:
`$ man 2 write`

NAME

write - write to a file descriptor

SYNOPSIS

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

DESCRIPTION

write() writes up to count bytes from the buffer pointed buf to the file referred to by the file descriptor fd.

Outline

Caratteristiche del Linux kernel

- Scritto quasi esclusivamente in C
- Kernel UNIX like
- Portato su 24 architetture diverse
- Licenza GPL
- Architettura Monolitica Modulare (Linux Kernel Module infrastructure)
- Preemptible Kernel (dalla versione 2.6)
- Supporto SMP (Simmetric Multi Processor)
- POSIX (The Portable Operating System Interface) compliant
- Supporta la paravirtualizzazione (da 2.6.20)

Outline

Cosa serve

1. Installazione di una distribuzione **GNU**/Linux sul proprio PC
2. Un compilatore C: **GCC**
3. I sorgenti del kernel (`http://www.kernel.org`)
4. Buona conoscenza del linguaggio C
5. Buona conoscenza della shell **sh**
6. Molta pazienza (mandatory)
7. Opzionale ma consigliabile, una installazione di una macchina virtuale User Mode Linux (**UML**) oppure **VMware**, XEN, KVM, ...)

Regole del Kernel Hacking

- **Non esistono operazioni in virgola mobile**: il kernel linux deve funzionare sulla maggior parte dei sistemi anche quelli non dotati di FPU (floating point unit) come ad esempio gli embedded devices.
- **Lo stack ha una dimensione fissa e ridotta**
  - utilizzare ricorsione in kernel space non è indicato
  - inizializzare array statici di grandi dimensioni non è indicato meglio usare allocazione dinamica
- **Il kernel linux è portabile**
  - il codice del kernel deve essere indipendente dalla endianness della macchina
  - Il codice in assembler va evitato a meno che non si includa in `arch/`
  - Quando si sviluppa si deve pensare di avere un sistema SMP

Configurare e compilare il kernel Linux

- Ottenere il tarball dei sorgenti del kernel da http://www.kernel.org/
- Spostare il tarball in /usr/src e decomprimerlo
- Creare un link simbolico a /usr/src/linux
- Configurare il kernel
- Compilare il kernel ed i moduli
- Installare il kernel
- Riavviare il sistema per provare il nuovo kernel

Build del kernel in /usr/src

```
# wget http://kernel.org/pub/linux/
kernel/v2.6/linux-2.6.23.1.tar.bz2
# tar xjf linux-2.6.23.1.tar.bz2
# ln -s linux-2.6.23.1 linux
# cd linux
# make menuconfig
# make && make modules_install
# cp
/usr/src/linux/arch/i386/boot/bzImage
/boot/vmlinuz-2.6.23.1
# mkinitrd -r /dev/<root_dev> -o
/boot/initrd.img-2.6.23.1 2.6.23.1
# grub-update
```

Compilare il kernel linux (Debian way)

Se si utilizza una distribuzione basata su **Debian** (es. Ubuntu, Knoppix, ...), è possibile ricompilare il kernel con il comando:

```
/usr/src/linux $ make-kpkg --initrd --rootcmd
    fakeroot --revision 1 kernel_image
```

In questo modo verrà creato un paccketto **.deb** installabile contenente l'immagine del kernel e i moduli:

```
$ sudo dpkg -i
/usr/src/linux-image-2.6.23.1_1_i386.deb
```

Il tree dei sorgenti

Le subdirectories più importanti sono:

- `arch/` Codice dipendente dall'architettura scritto in assembler
- `ipc/` Memoria condivisa, semafori, messaggi inter process...
- `kernel/` CPU scheduler, DMA ...
- `fs/` Implementazione di svariati file systems (compresi quelli virtuali)
- `mm/` Gestore della memoria
- `drivers/` Implementazione dei drivers dei dispositivi supportati ufficialmente
- `net/` Implementazione dei livelli ISO/OSI: LLC, **stack TCP/IP**, ipv6, netfilter

Configurare il kernel

Utilizzando `make menuconfig` è possibile customizzare il kernel. Ad esempio, per abilitare tutti gli algoritmi di controllo di congestione entrare nella sezione TCP Congestion Control nel seguente modo:

Networking→Networking Options →TCP: advanced congestion control

e abilitare come moduli (`M`) tutti gli algoritmi desiderati.

Outline

Introduzione ai LKM

- Un Linux Kernel Module (**LKM**) è un pezzo di codice che serve ad estendere le funzionalità del kernel.
- Un LKM ha estensione **.ko** e può essere caricato e rimosso dal kernel a *runtime* (senza dover effettuare il reboot) tramite i comandi **insmod** ed **rmmod**.
- Si possono ricavare informazioni sui moduli caricati con il comando **lsmod**
- Il comando **modprobe** serve a caricare o rimuovere i moduli installati in **/lib/modules/<versione>/**
- Il codice di un LKM viene eseguito in kernel space come qualunque altre parte del kernel e quindi può bloccare il sistema o danneggiarlo.

Hello Kernel Space

```
/* Header files da includere per un LKM */
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/tty.h> /* Serve per console_print() */
/* Inizializzazione del LKM */
int init_module()
{
    console_print("Hello Kernel Space!\n");
    /* Ritornare 0 se non ci sono errori */
    return 0;
}
/* Cleanup */
void cleanup_module()
{
    console_print("Goodbye Kernel Space!\n");
}
```

Compilazione di un LKM

Makefile

```
obj-m += hello.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Compilazione

```
# make
# insmod hello.ko
Message from syslogd@localhost at Tue Feb 28 21:50:51 2006
localhost kernel:  Hello Kernel Space!
# rmmod hello
Message from syslogd@localhost at Tue Feb 28 21:51:33 2006
localhost kernel:  Goodbye Kernel Space!
```

Outline

Il controllo di congestione nel kernel

- Fino alla versione 2.6.12 tutti gli algoritmi di controllo di congestione erano implementati in un unico file (`net/ipv4/tcp_input.c`)
- Nel 2005 Steven Hemminger ha riscritto il codice per il congestion control rendendolo modulare e a plug-in:
  - Ogni congestion control algorithm è implementato in un LKM
  - E' possibile selezionare l'algoritmo tramite l'utilizzo di una sysctl o utilizzando il file system virtuale procfs

Example

```
# modprobe tcp_westwood
# sysctl net.ipv4.tcp_congestion_control=westwood
# echo "reno" > /proc/sys/net/ipv4/tcp_congestion_control
```

Linux TCP state machine (1/3)

Al congestion control di linux e' associata una macchina a 5 stati:

- **Open**: Alla ricezione di un ACK **in sequenza** il TCP sender aumenta la congestion window di un valore che dipende se si e' in congestion avoidance o in slow start.
- **Disorder**: Quando si riceve un ACK **fuori sequenza** o **duplicato** si entra in questo stato. Cwnd non viene modificata, ma ogni qual volta si riceve un ACK si invia un nuovo pacchetto (*self-clocking*).

Linux TCP state machine (2/3)

- **CWR (congestion window reduced)**: Quando si riceve un evento esplicito di congestione (Explicit Congestion Notification - ECN) si entra in questo stato e la cwnd viene ridotta di un pacchetto per ogni ACK ricevuto finche' non si arriva a dimezzare *cwnd*. Questa fase puo' essere interrotta dalle fasi di Recovery o di Loss.
- **Loss**: Se il retransmission timeout (RTO) espira si entra in questo stato.
  - Si considerano persi tutti i pacchetti outstanding.
  - cwnd viene impostata a 1
  - la cwnd viene incrementata utilizzando l'algoritmo *slow start*
  - si permane in questo stato finche' non sono ritrasmessi con successo tutti i pacchetti considerati persi

Linux TCP state machine (3/3)

Recovery state: il Rate Halving

- Il *Rate Halving* e' una feature del CC del Linux kernel non standardizzata in RFC.
- L'algoritmo agisce nella fase di Fast Recovery (attivata alla ricezione di **3 DUPACK**):
  1. Ogni due ACK ricevuti:
     1. si decrementa *cwnd* di un pacchetto fino ad arrivare alla *ssthresh*
     2. si invia un nuovo pacchetto sulla rete (il pacchetto perso o nuovi pacchetti)
  2. Si esce dalla fase di Fast Recovery quando arrivano gli ACK di tutti i pacchetti che erano in volo all'inizio di questa fase o quando espira un RTO (e si finisce nel *Loss state*)

Outline

Implementazione

I file responsabili del congestion control in Linux (versione $\geq$ 2.6.13) sono:

- `tcp_cong.c`: Implementazione della architettura a plugin.
  - `tcp_register_congestion_control`: Inserisce un nuovo algoritmo di CC alla lista degli algoritmi di CC disponibili
  - `tcp_set_congestion_control`: Cambia il CC al socket passato come argomento
  - `tcp_reno_cong_avoid`, `tcp_reno_ssthresh`, `tcp_reno_min_cwnd`: Implementazione di Reno TCP
- `tcp_westwood.c`, `tcp_bic.c`, `tcp_vegas.c`, `tcp_hybla.c`, `tcp_scalable.c`, `tcp_highspeed.c`: Implementazione di Westdood+, BIC, Vegas, Hybla, Scalable, HSTCP.
- `tcp.h`: definizione della struttura `tcp_congestion_ops`

tcp.h (1/2)

Ad ogni congestion control algorithm e' associata una struttura `tcp_congestion_ops` cosi' fatta:

tcp.h

```
struct tcp_congestion_ops {
  struct list_head list;
  /* return slow start threshold (required) */
  u32 (*ssthresh)(struct sock *sk);
  /* do new cwnd calculation (required) */
  void (*cong_avoid)(struct sock *sk, u32 ack,u32 rtt, u32
in_flight, int good_ack);
  [...]
  char name[TCP_CA_NAME_MAX];
};
```

tcp.h (2/2)

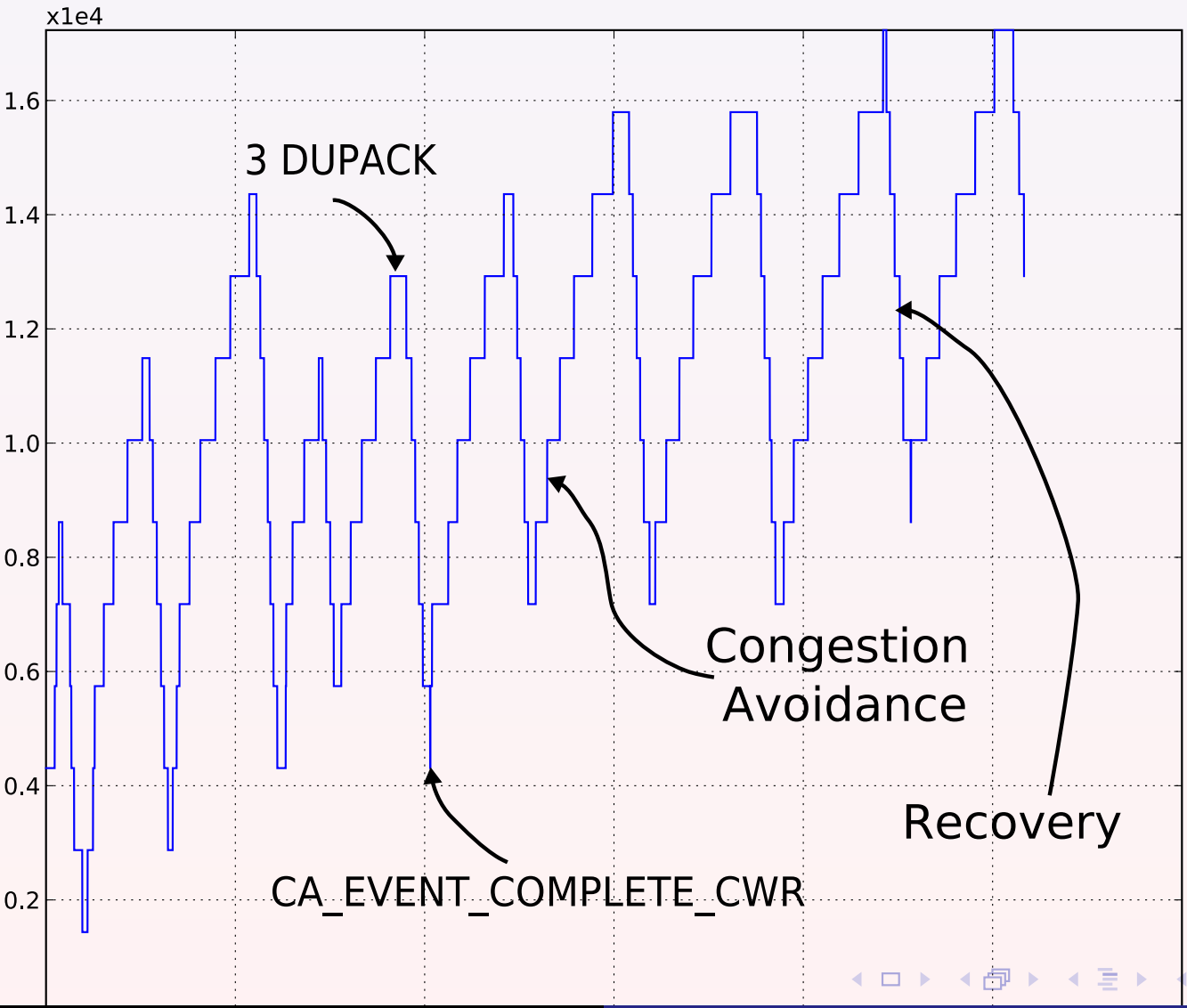
- **init/release**: inizializza/distrugge la (eventuale) sezione privata dei dati
- **ssthresh**: Ritorna il valore della Slow Start Threshold
- **min_cwnd**: Ritorna il valore della congestion window dopo un congestion episode
- **cong_avoid**: Funzionamento del CC in congestion avoidance
- **rtt_sample**: Ritorna un campione di RTT (Round Trip Time)
- **cwnd_event**: Gestisce gli eventi del Congestion Control (vedi slide successiva)

Gli eventi del Congestion Control

In tcp.h sono definiti gli eventi che vengono triggerati quando si verificano certe condizioni durante il funzionamento dell'algoritmo di Congestion Control:

- **CA_EVENT_LOSS**: E' attivato quando si rileva una perdita (in tcp_input.c funzione *tcp_enter_loss*)
- **CA_EVENT_FAST_ACK**: E' attivato quando si riceve un ACK in sequenza (in tcp_input.c funzione *tcp_ack*)
- **CA_EVENT_SLOW_ACK**: E' attivato quando si riceve un ACK fuori sequenza (in tcp_input.c funzione *tcp_ack*)
- **CA_EVENT_COMPLETE_CWR**: La fase di congestion recovery e' terminata (in tcp_input.c funzione *tcp_complete_cwr*)

Esempio


x1e4
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
3 DUPACK
Congestion
Avoidance
Recovery
CA_EVENT_COMPLETE_CWR

Outline

Scrivere un nuovo modulo

Cosa serve per scrivere un nuovo algoritmo di controllo di congestione?

1. Progettare l'algoritmo "su carta" specificando il comportamento nelle fasi di slow start, congestion avoidance e di shrinking (reazione ad un episodio di congestione)
2. Validare l'algoritmo con NS-2 (richiede la scrittura di codice...)
3. Individuare le parti dell'algoritmo diverse da quelle standard di new reno.
4. Implementare le parti dell'algoritmo che differiscono da quelle standard.
5. Validare l'implementazione

Esempio

TCP Reno Mod

Si supponga di voler modificare TCP Reno in modo da incrementare la finestra di due pacchetti per RTT quando si e' in congestion avoidance

Suggerimento

Studiare il file **net/ipv4/tcp_cong.c**, funzione: **tcp_reno_cong_avoid()**

Riferimenti

- http://c3lab.poliba.it/index.php/MetodiDiControllo
- http://kernel.org
- http://tldp.org/HOWTO/Module-HOWTO/
- http://www.tldp.org/LDP/lkmpg/2.6/html/index.html
- http://kvm.qumranet.com
- http://www.vmware.com/download/player/
  http://www.vmware.com/appliances/directory/1028
- http://gapil.truelite.it/