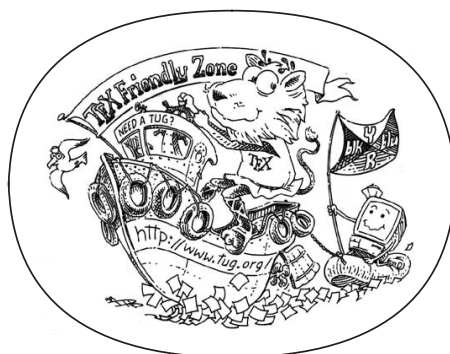


SISTEMI E TRASMISSIONE DELLE INFORMAZIONI

RICCARDO CEREGHINO



Appunti

Settembre 2019 – o.o.1classicthesis v4.6

INDICE

I RETI DI CALCOLATORI

1	ACCENNI AL FUNZIONAMENTO DI UN CALCOLATORE	3
1.1	Networking	3
1.1.1	TCP/UDP	3
1.1.2	Device driver	3
1.2	Trasmissione di dati	3
1.2.1	Invio di dati	3
1.2.2	Ricezione di dati	4
1.2.3	Trasmissione di dati su bus ring based	4
1.2.4	Commutazione	5
1.2.5	Dataflow	5
1.2.6	Routing	5
1.3	Internet protocol stack	5
1.3.1	Protocollo fisico	5
1.3.2	Network protocol	6
1.3.3	Trasporto	7
2	INTRODUZIONE AI SISTEMI OPERATIVI	9
2.1	CPU virtualizzata	9
2.2	Memoria virtualizzata	9
2.3	Periferiche	10

II APPENDIX

C	APPENDIX TEST	13
---	---------------	----

ELENCO DELLE FIGURE

Figura 1.1	indirizzi	7
Figura 1.2	Three way handshake	8
Figura 2.1	Struttura base di un calcolatore	9

ELENCO DELLE TABELLE

LISTINGS

Listing 2.1	Dimostrazione della virtualizzazione della CPU	9
Listing 2.2	Dimostrazione dell'utilizzo di indirizzi virtuali	9
Listing c.1	Esempio di MAKEFILE	13

ACRONYMS

Parte I

RETI DI CALCOLATORI

ACCENNI AL FUNZIONAMENTO DI UN CALCOLATORE

Un calcolatore è generalmente composto da:

- la *CPU*, esegue le istruzioni, tiene in memoria (*InstructionRegister*) l'istruzione corrente e la posizione dell'istruzione nella RAM (*P.C.*);
- la *RAM*, tiene in memoria le istruzioni da eseguire, certe sezioni sono riservate per specifiche funzioni (*stack*, *SP*, *BP*);
- dei moduli, come un disco fisso o altro;
- la *NIC*, che permette di inviare e ricevere pacchetti in rete.

1.1 NETWORKING

Vengono effettuate delle *syscall*, per eseguire operazioni necessario all'invio di dati. Per esempio la *syscall send*, copia dalla ram una certa sezione di memoria nel *NIC*, che verrà inviata in rete.

1.1.1 TCP/UDP

Il protocollo TCP garantisce l'invio del messaggio, dato che attende una risposta da parte del ricevente. Il protocollo UDP invia datagrammi, ma non si è certi se la ricezione è avvenuta.

1.1.2 Device driver

E' un modulo che gestisce le risorse fisiche dei vari moduli, tra cui *NIC*, gestiscono le interruzioni del dispositivo.

1.2 TRASMISSIONE DI DATI

1.2.1 Invio di dati

Il *DMA* presente nel *NIC* è il componente che accede alla memoria.

La maggior parte dei calcolatori include l'unità di gestione della memoria *MMU*, che permette al processore di gestire indirizzi virtuali per ottenere una maggiore efficienza, ma questo dispositivo è esclusivo del processore, motivo per cui il *DMA* del *NIC* utilizzerà indirizzi fisici.

Delle porzioni di memoria della RAM saranno destinate ad essere utilizzate per i *buffer*, ovvero delle aree dove NIC può immagazzinare i dati da inviare o da ricevere.

Al momento dell'invio di un datagramma, esso verrà salvato in un buffer, quindi inviato in rete appena possibile, quindi cancellato dalla memoria.

1.2.2 Ricezione di dati

Per la ricezione di datagrammi dei buffer devono essere sempre disponibili, cosicchè NIC possa utilizzarli una volta ricevuto il pacchetto.

Il datagramma ricevuto sarà inserito dal DMA in un buffer, un interrupt nel frattempo creerà nuovi buffer per possibili nuovi messaggi, quindi il sistema legge il buffer copiandolo nel buffer dell'applicazione, direzionando l'output verso l'utente corretto, quindi la memoria viene liberata.

Il comportamento standard di un'applicazione che implementa networking implementa un meccanismo bloccante per cui l'applicazione rimane in attesa fintanto che il buffer non è stato scritto al suo interno.

1.2.3 Trasmissione di dati su bus ring based

La problematica principale del DMA di tipo *bus mastering* è la necessità di dover programmare in anticipo i buffer allocati, limitandone la capacità.

Fintanto che il DMA è impegnato in un'operazione perchè in attesa del processore verranno persi i dati nel frattempo ricevuti dalla rete.

Per risolvere questo problema si implementa una struttura dati più complicata, integrando una piccola CPU all'interno della NIC (ring based).

1.2.3.1 Ring

Il ring è composto da un anello contenente più di un buffer, e due puntatori, un puntatore di inizio ed uno di fine gestiti da due processori diversi; la CPU si occupa di aggiungere elementi all'interno del ring quindi gestirà l'indice di fine, il puntatore al primo elemento viene gestito dal processore all'interno della NIC. Si ottiene una coda di buffer che si possono aggiungere o togliere a seconda se si sta gestendo un invio o ricezione di dati.

Quindi la NIC legge i buffer in memoria e usa un flag per impostare se il buffer è libero oppure se deve essere processato.

L'unico tipo di comunicazione tra processore e DMA è la trasmissione della quantità di buffer che dovranno essere utilizzati al DMA.

1.2.4 Commutazione

Si rende necessario, per una questione di efficienza, un circuito fisico R attraverso cui trasmettere i pacchetti.

I pacchetti sono inviati come datagrammi, e nella struttura includono il destinatario e le informazioni trasmesse.

1.2.5 Dataflow

L' *host* compila ed invia il buffer in locale al buffer di ricezione di $R1$ (router), il cui compito è di leggere il datagramma, ricavarne il destinatario, quindi di inviarlo: l'*host* può quindi cancellare il buffer.

La parte ricevente ($R2$) copia il datagramma nel suo buffer di ricezione, quindi $R1$ può rimuovere dal proprio buffer il datagramma. Quindi $R2$ invia il datagramma all'effettivo destinatario con un altro *store and forward*.

Ogni passaggio di trasmissione del datagramma viene chiamato *hop*.

1.2.6 Routing

Identifichiamo per il processo di routing l' *host*, gli endpoint della trasmissione di pacchetti e i *router*, calcolatori. Più router possono essere collegati tra loro.

1.3 INTERNET PROTOCOL STACK

1. Physical
2. Datalink
3. Network
4. Transport
5. Application

1.3.1 Protocollo fisico

L'unico protocollo fisico sopravvissuto ed in uso è il protocollo ethernet.

Ogni richiesta sarà composta da: header, payload e trailer.

Il protocollo definisce che ad ogni *host* venga assegnato un indirizzo *MAC*, composto da *6byte*.

L'header conterrà come dati gli indirizzi IP e le porte dei nodi comunicanti.

Il trailer è necessario per il controllo di integrità nella rete ethernet, l'algoritmo utilizzato è **CRC32**.

Il datalink utilizza le stesse modalità di funzionamento del protocollo fisico.

1.3.2 Network protocol

Il protocollo network utilizza come gestore degli indirizzi i protocolli *IPv4* e *IPv6*. *IPv4* utilizza *4byte* per il funzionamento, significa che può gestire fino a circa 4 miliardi di indirizzi, mentre *ipv6* ne può utilizzare molti di più.

I primi sviluppatori dei protocolli non avevano la possibilità di interagire direttamente con i sistemi operativi, motivo per cui si è dovuto introdurre il concetto di **porta di comunicazione**, un numero intero in *16bit*, con cui si identifica univocamente un numero di una porta ad una tipologia di applicazione.

Le porte da 0 a 1023 sono state dedicate ai protocolli internet ed alle applicazioni più comuni, un'altra serie di porte sono sempre utilizzate per il networking mentre le restanti sono le così dette porte effimere che vengono utilizzate dalle applicazioni in locale.

Quindi se la porta è la parte che un calcolatore espone ad internet, il *socket* è il corrispettivo componente nel nodo in locale, così da associare alla richiesta una macchina ed applicazione in particolare.

Sono usati i protocolli *UDP* e *text*. Il protocollo *UDP* invia datagrammi mentre il protocollo *TCP* permette di implementare una modalità di comunicazione più avanzata, di tipo *stream*, che permette di interagire con l'interfaccia di comunicazione come se fossero file.

Gli indirizzi **IP** sono utilizzati per essere associati agli indirizzi **MAC** nel network.

IL *TTL* (Time To Live) definisce da quanto tempo un pacchetto è nel network.

RFC (Remote Function Call) cerca prima di chiamare le versioni dei protocolli più recenti prima di doverne eseguire uno più vecchio.

1.3.2.1 Indirizzi

Ad ogni macchina corrisponde un IP (*IPv4* o *IPv6*)

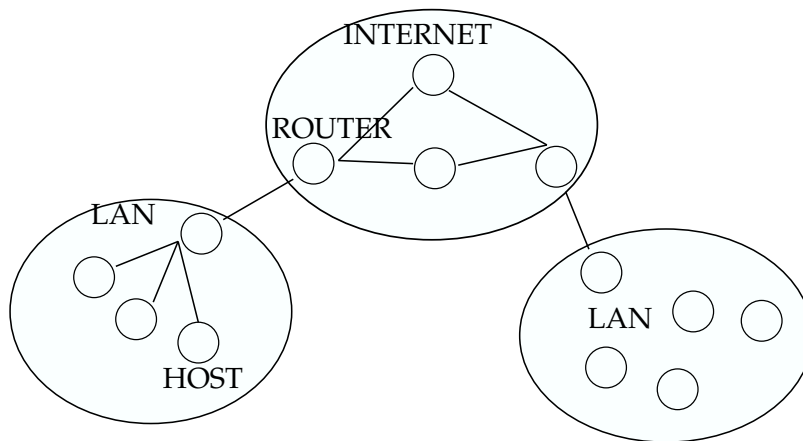


Figura 1.1: indirizzi

Un indirizzo *IPv4* in default è composto da:

- 29bit dedicati al network;
- 3bit dedicati all'host.

Per ovviare a problemi di carenza di indirizzi IP, si introduce il concetto di *NETMASK*, una parola composta da una serie di 1 ed una di 0, gli 1 rappresentano quanti bit sono dedicati al network, gli 0 per l'host.

INDIRIZZI PRIVATI Sono definiti nel protocollo *RFC 1918* dichiara che un indirizzo **IPv4** deve essere composto da 4 numeri decimali seguiti da punto, inoltre dichiara che gli indirizzi che cominciano con 10. appartengono alla *classe A*, questi indirizzi sono i cosiddetti indirizzi privati, utilizzabili solo in **LAN**.

Gli indirizzi di classe **B** cominciano da 192.168.0.0 per arrivare fino 192.168.255.255, sono sempre indirizzi privati.

Si può configurare il router per usare il **NAT** (Network Access translator), con lo scopo di assegnare due indirizzi IP allo stesso router, con lo scopo di esporne uno alla rete locale e l'altro a quella internet.

Con il NAT è possibile convertire indirizzi pubblici in indirizzi privati, direzionando i messaggi alla macchina corretta sostituendo il proprio indirizzo alla richiesta assegnando una porta libera.

1.3.3 Trasporto

Vengono utilizzati i protocolli *UDP* e *TCP* e la tecnica della *3 way handshake*.

Il trasporto comincia con l'invio di un messaggio **SYN** (accompagnato da **SEQ**), all'arrivo il server risponde con un messaggio che viene chiamato **SYN-ACK**, la **SEQ** e la **ACK**; quindi l'host invia un ulteriore messaggio chiamato **ACK**, oltre che i dati.

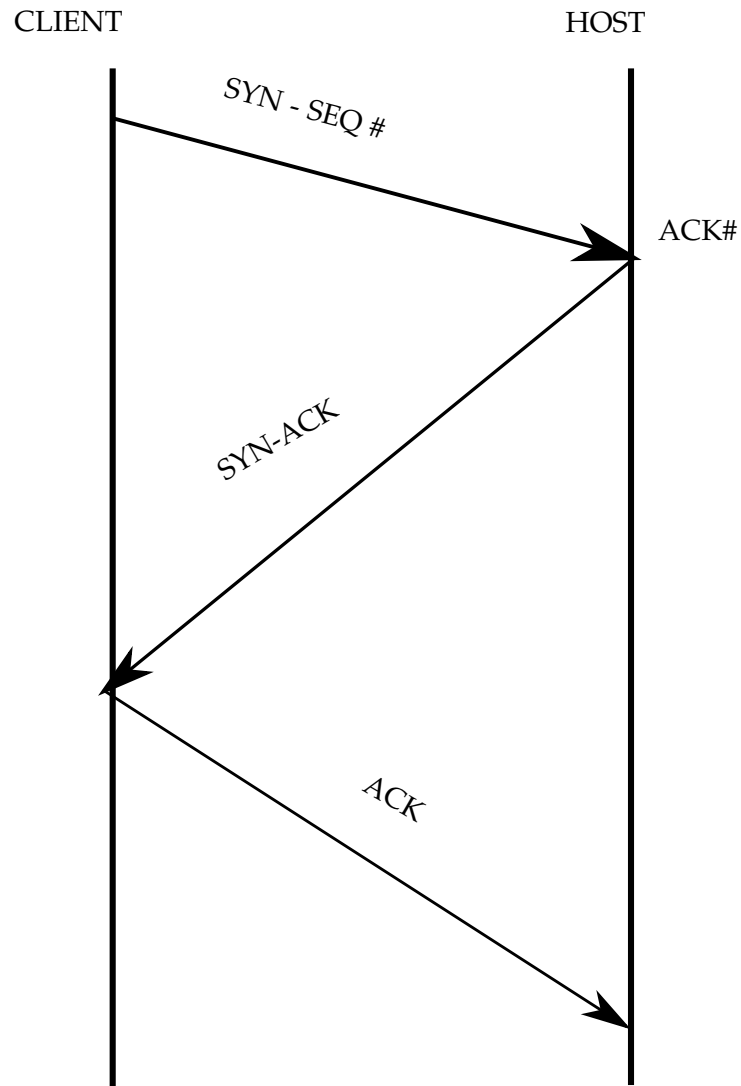


Figura 1.2: Three way handshake

L'header TCP contiene una serie di informazioni tra le quali i **FLAG** SYN e ACK oltre che la porta sorgente (**SPORT**) e di destinazione (**DPORT**).

Vengono utilizzate 2 parole, la **SEQ** e la **ACK**, permettono la di realizzare l'astrazione dello stream di byte; questi numeri per motivi di sicurezza non partono da 0, ma da un valore casuale; lo scopo della prima sequenza di passaggi è di scambiare queste informazioni.

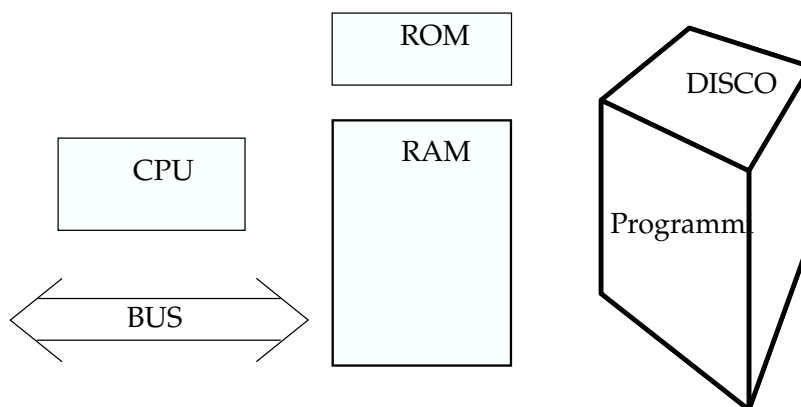


Figura 2.1: Struttura base di un calcolatore

2.1 CPU VIRTUALIZZATA

Listing 2.1: Dimostrazione della virtualizzazione della CPU

```
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: cpu <string>\n");
        exit(1);
    }
    char *str = argv[1];

    while (1) {
        printf("%s\n", str);
        Spin(1);
    }
    return 0;
}
```

2.2 MEMORIA VIRTUALIZZATA

Listing 2.2: Dimostrazione dell'utilizzo di indirizzi virtuali

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]){
    int value;
    int *p = &value;
    if(argc != 2) {
        fprintf(stderr, "usage: %s <value>\n", *argv);
        return EXIT_FAILURE;
    }
    printf("(pid:%d) addr of p: %llx\n", (int)getpid(), (unsigned
        long long)&p);

    printf("(pid:%d) addr stored in p: %llx\n", (int)getpid(), (
        unsigned long long)p);

    *p = atoi(argv[1]);
    while(1) {
        sleep(1);
        ++*p;
        printf("(pid:%d) value of p: %d\n", getpid(), *p);
    }
}

```

2.3 PERIFERICHE

Nell'interazione di un calcolatore con le periferiche, è il kernel che interagisce direttamente con l'hardware, mentre le applicazioni possono utilizzare le periferiche solo attraverso il kernel.

Vi sono 4 diversi livelli di privilegi per interagire con i dispositivi:

- 0, il livello più privilegiato, utilizzato dal kernel;
- 1 solitamente non utilizzato;
- 2 solitamente non utilizzato;
- 3 il livello meno privilegiato, utilizzato da tutti gli utenti compreso l'utente di *root*.

Se l'utente richiede un'interazione con una periferica, un comando di *TRAP* viene inviato al kernel, il quale restituisce un comando di *return from TRAP*.

Parte II

APPENDIX

APPENDIX TEST

Listing c.1: Esempio di MAKEFILE

```
CFLAGS=-Wall -ansi -pedantic -Werror -std=c11
CC=clang
EXES=cpu mem threads.v0 threads.v1 threads_101

all: $(EXES)

cpu: cpu.c
    $(CC) $(CFLAGS) -o $@ $^

cpu: mem.c
    $(CC) $(CFLAGS) -o $@ $^

threads.v0: threads.v0.c
    $(CC) $(CFLAGS) -o $@ $^ -pthread

threads.v1: threads.v0.c
    $(CC) $(CFLAGS) -o $@ $^ -pthread

threads_101: threads.v0.c
    $(CC) $(CFLAGS) -o $@ $^ -pthread

clean:
    rm -f $(EXES)

.PHONY: clean
```