La memoria virtuale

Problema: è spesso necessario eseguire programmi che richiedono più memoria di quanta non sia disponibile sul calcolatore.

Soluzione: utilizzare la memoria secondaria (disco)

Una delle prime tecniche utilizzate: *overlaying*

Il programmatore divideva il programma in un certo numero di parti dette overlay di dimensione tale da poter essere caricate in memoria.

Ogni overlay, una volta terminata la propria esecuzione, caricava in memoria l'overlay successivo e lo mandava in esecuzione.

1

La memoria virtuale

Il programmatore si doveva preoccupare di:

- dividere il programma
- decidere la locazione in memoria secondaria degli overlay
- gestire il caricamento in memoria principale degli overlay

⇩

Virtualizzazione della memoria

Si separano i concetti di spazio di indirizzamento e di memoria fisica

Le due tecniche principali di virtualizzazione della memoria sono:

- la paginazione
- la segmentazione

2

La memoria virtuale: paginazione

Un calcolatore con istruzioni con un campo indirizzi a 16 bit è in grado di indirizzare 64K locazioni di memoria.

Lo spazio degli indirizzi consiste in questo caso dei numeri 0,1,…, 65535

Il calcolatore può avere una memoria fisica costituita da un numero di locazioni molto minore (supponiamo 4K)

Nel caso in cui non ci sia un meccanismo di virtualizzazione della memoria e` necessario distinguere gli indirizzi da 0 a 4K-1 (spazio di indirizzamento utilizzabile) da quelli superiori ed uguali a 4K (spazio degli indirizzi senza una corrispondenza in memoria fisica)

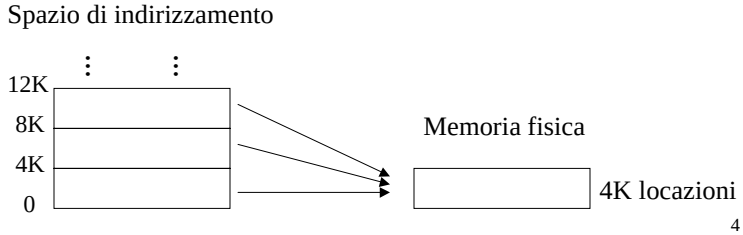
3

La memoria virtuale: paginazione

Nel caso in cui la memoria sia virtualizzata il programma utilizza indirizzi virtuali che vengono tradotti in indirizzi fisici da un apposito meccanismo. Ad es:

agli indirizzi 0, 4K, 8K, … potrebbe corrispondere la locazione con indirizzo 0 in memoria fisica

agli indirizzi 1, 4K+1, 8K+1, … potrebbe corrispondere la locazione con indirizzo 1 in memoria fisica



4

La memoria virtuale: paginazione

Se si prova ad accedere ad una locazione con indirizzo maggiore di 4K-1 in un sistema senza virtualizzazione della memoria viene generata una segnalazione di errore.

Nel caso del sistema con virtualizzazione:

1 la memoria principale viene salvata sul disco

2 viene caricato dal disco il blocco al quale l'indirizzo appartiene

3 l'indirizzo virtuale viene tradotto in modo da accedere alla opportuna locazione in memoria fisica

5

La memoria virtuale: paginazione

L'insieme degli indirizzi a cui i programmi possono fare riferimento è detto *spazio di indirizzamento virtuale*

L'insieme degli indirizzi effettivamente cablati è detto *spazio di indirizzamento fisico*

I blocchi in cui è diviso lo spazio di indirizzamento virtuale sono detti pagine (in questo caso la memoria fisica può contenere una sola pagina)

Le pagine hanno una dimensione fissa

Oltre al meccanismo di traduzione degli indirizzi e` necessario un sistema che gestisca lo spostamento delle pagine dalla memoria secondaria alla memoria principale e viceversa.

6

La memoria virtuale: paginazione

Alcuni vantaggi:

•I programmi possono essere scritti come se il sistema disponesse di una quantità di memoria principale pari allo spazio di indirizzamento

•La dimensione dei programmi è limitata dalla memoria secondaria e non da quella principale

Il meccanismo di paginazione è *trasparente* in quanto il programmatore può programmare come se tale meccanismo non esistesse

7

La memoria virtuale: meccanismi per la paginazione

Lo spazio di indirizzamento virtuale è diviso in un certo numero di pagine di dimensione fissa (da 512 a 64k byte)

Lo spazio di memoria fisica è diviso in blocchi della stessa dimensione e detti *page frame*

Supponiamo di avere un sistema con indirizzi virtuali a 32 bit, 32k locazioni di memoria fisica e pagine di dimensione 4k.

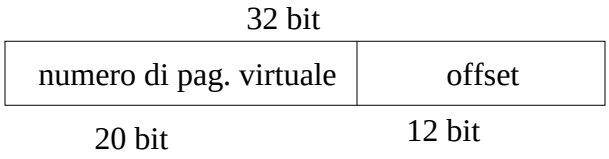
L'indirizzo virtuale a 32 bit deve essere tradotto in un indirizzo fisico a 15 bit.

Il dispositivo che opera la traduzione degli indirizzi è detto Memory Management Unit (MMU)

8

La memoria virtuale: meccanismi per la paginazione

L'indirizzo virtuale può essere scomposto in:



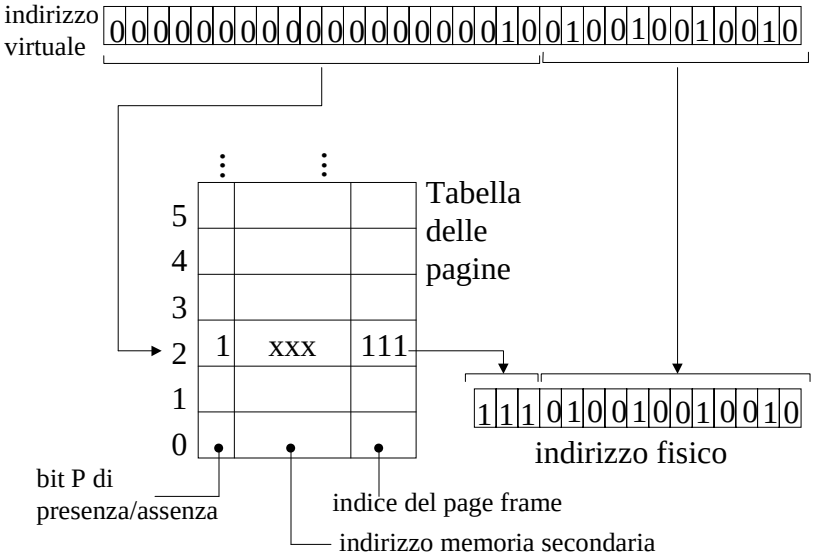
I 12 bit meno significativi sono utilizzati come indirizzo interno alla pagina

I restanti bit sono utilizzati come numero di pagina virtuale

Il numero di pagina virtuale è adoperato come indice nella *tabella delle pagine*

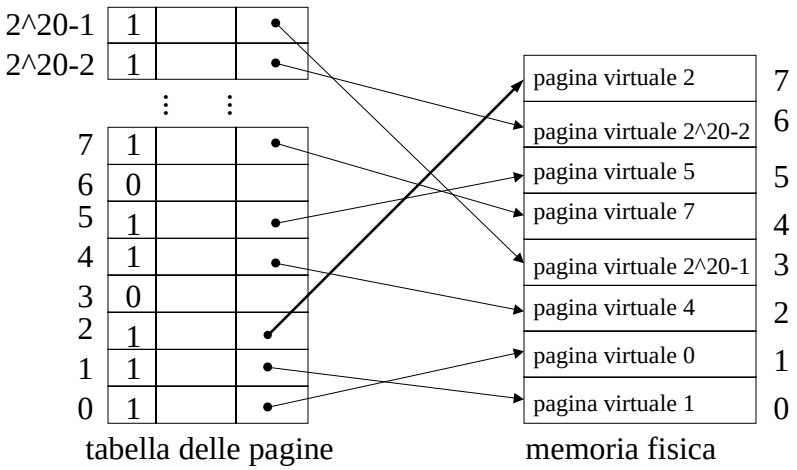
9

La memoria virtuale: meccanismi per la paginazione



10

La memoria virtuale: meccanismi per la paginazione



11

La memoria virtuale: meccanismi per la paginazione

Ovviamente non tutte le pagine della memoria virtuale possono essere in memoria fisica

Quando viene usato un indirizzo di una pagina virtuale non presente in memoria fisica viene generato un *page fault.*
In tale caso:

1 la pagina richiesta viene caricata dalla memoria secondaria

2 la tabella delle pagine viene aggiornata

3 viene ripetuta l'istruzione che ha generato il page fault

Questo modo di operare è detto *paginazione su domanda*

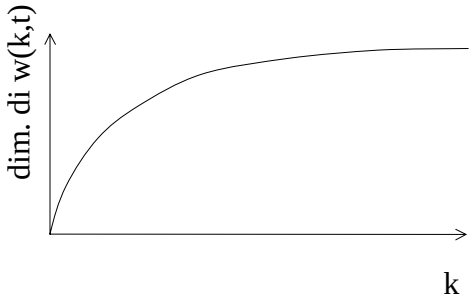
12

La memoria virtuale: meccanismi per la paginazione

E' quindi possibile far partire un programma anche quando neanche una porzione del programma è in memoria fisica

Dopo un certo periodo di tempo le pagine necessarie all'esecuzione sono quasi tutte in memoria centrale ed i riferimenti alla memoria tendono a concentrarsi su un piccolo numero di pagine

All'istante *t* l'insieme costituito dalle pagine usate dai *k* più recenti riferimenti alla memoria è detto **working set** w(k,t)



13

La memoria virtuale: politiche di rimpiazzamento

Quando si verifica un page fault e non c'è spazio in memoria fisica per caricare la pagina richiesta è necessario spostare un'altra pagina dalla memoria principale al disco

Come scegliere la pagina da spostare sul disco in modo tale da minimizzare la probabilità che si verifichi a breve un altro page fault?

Algoritmi più usati:

•**LRU** (Least Recently Used) rimuove la pagina che non viene riferita da più tempo

•**FIFO** (First In First Out) rimuove la pagina che è in memoria da più tempo

14

La memoria virtuale: politiche di rimpiazzamento

Se un programma genera page fault in continuazione (working set molto più grande della memoria fisica) si dice che il programma è in una condizione di *thrashing*

Nel caso in cui la pagina in memoria fisica coinvolta nel rimpiazzamento non sia stata modificata, è inutile riscriverla sul disco

La MMU utilizza un bit per pagina (dirty/clean) che è a 0 quando la pagina viene caricata in memoria, e viene posto ad 1 quando la pagina è acceduta in scrittura

15

La memoria virtuale: dimensione delle pagine

•Se la pagina ha dimensione *n* lo spazio sprecato nell'ultima pagina sarà in media *n*/2 (*frammentazione interna*)

•Pagine di piccola dimensione si comportano meglio quando il working set è costituito da un grosso numero di piccole regioni dello spazio degli indirizzi virtuali

•Le pagine di dimensione maggiore fanno uso migliore del disco. Ad es:
seek+rotational delay=10msec *transfer rate*=10 MB/sec

pagina da 1KB: 10msec+0,1msec

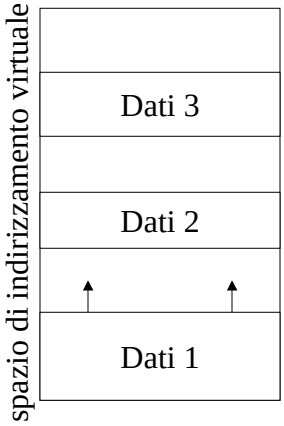
pagina da 8KB: 10msec+0,8msec

•Le pagine di grossa dimensione richiedono una tabella delle pagine più piccola

16

La memoria virtuale: segmentazione

Una memoria virtuale unidimensionale si presta male a risolvere alcuni problemi:



se Dati1 cresce troppo va in collisione con Dati2 nonostante sia ancora disponibile dello spazio di indirizzamento virtuale libero

⇩

è necesario un meccanismo che gestisca queste situazioni in modo automatico

⇩

Più spazi di indirizzamento indipendenti detti **segmenti**

17

La memoria virtuale: segmentazione

Avere più spazi di indirizzamento indipendenti richiede che la memoria virtuale sia bidimensionale: l'indirizzo virtuale è costituito da due componenti

- **selettore** individua uno specifico segmento
- **offset** individua la locazione all'interno di un dato segmento

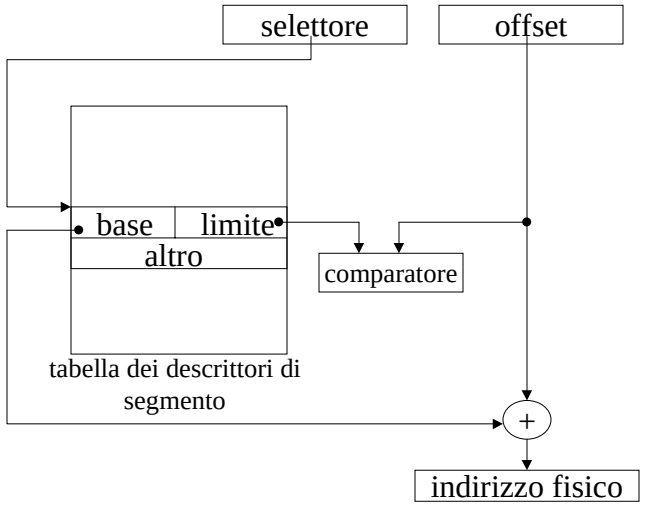
Il numero di bit utilizzati per il campo selettore determina il numero massimo di segmenti**,** mentre il numero di bit del campo offset determina la massima dimensione di un segmento

Segmenti diversi avranno in genere lunghezze diverse

18

La memoria virtuale: segmentazione

Traduzione degli indirizzi da virtuali a fisici:



19

La memoria virtuale: segmentazione

I descrittori di segmento contengono, oltre ai campi base e limite, un bit che indica se il segmento è in memoria o meno

Quando si tenta di accedere ad un segmento non presente in memoria si verifica un *segment fault*

In tal caso:

1 si deve trovare in memoria fisica spazio per il nuovo segmento eventualmente ricopiando su disco uno o più segmenti

2 si trasferisce il segmento dal disco alla memoria pincipale

3 si aggiorna la tabella dei descrittori di segmento

20

La memoria virtuale: segmentazione

Se Data1, Data2 e Data3 vengono poste in segmenti diversi le strutture dati possono crescere senza causare conflitti dal momento che ognuna è locata in uno spazio di indirizzamento separato

Poiché è il programmatore a decidere cosa mettere nei vari segmenti, il meccanismo della segmentazione è *non trasparente*

I segmenti possono avere diversi tipi di protezione, ad es:

•lettura/scrittura

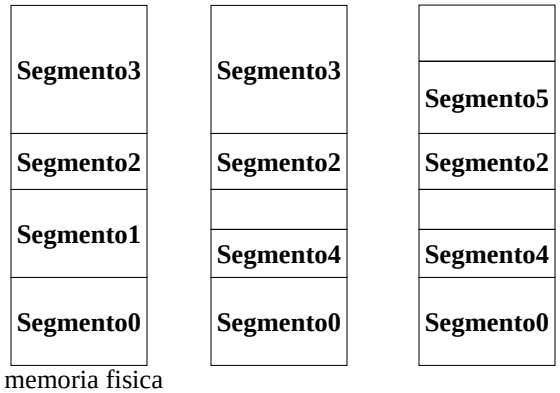
•esecuzione

questo perché il programmatore è consapevole di cosa contiene un segmento

21

La memoria virtuale: segmentazione

Le principali difficoltà implementative di un meccanismo di segmentazione derivano dal fatto che i segmenti non hanno una dimensione fissa



memoria fisica

nella memoria principale si formano dei "buchi" di dimensione variabile

frammentazione esterna

22

La memoria virtuale: segmentazione

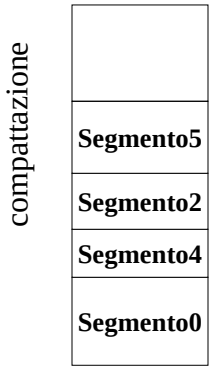
Può succedere che si debba caricare in memoria un segmento la cui dimensione è maggiore della dimensione dei singoli buchi, ma inferiore della loro somma

O si elimina dalla memoria principale un altro segmento o si compatta la memoria

La compattazione può essere effettuata:

•ogni volta che si crea un buco

•quando una certa percentuale della memoria è coperta da buchi



23

La memoria virtuale: segmentazione

E' necessario gestire una lista con le dimensioni e gli indirizzi delle aree di memoria libere

Nel caso in cui non si ricompatti sempre la memoria, l'area di memoria libera da usare per il nuovo segmento può essere scelta secondo due strategie:

•**best fit** sceglie l'area di memoria più piccola in grado di contenere il segmento

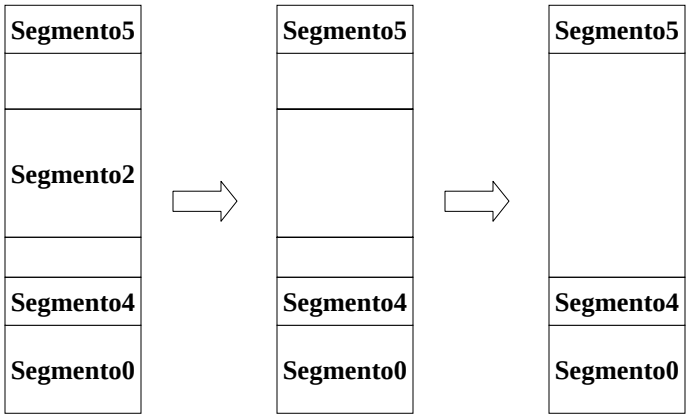
•**first fit** esamina in maniera circolare la lista delle aree libere e sceglie la prima in grado di contenere il segmento

Entrambe le tecniche fanno diminuire la dimensione media delle aree di memoria libera (soprattutto best fit)

24

La memoria virtuale: segmentazione

Quando un segmento viene tolto dalla memoria se le aree adiacenti sono libere possono essere fuse in una sola area di dimensioni maggiori



25

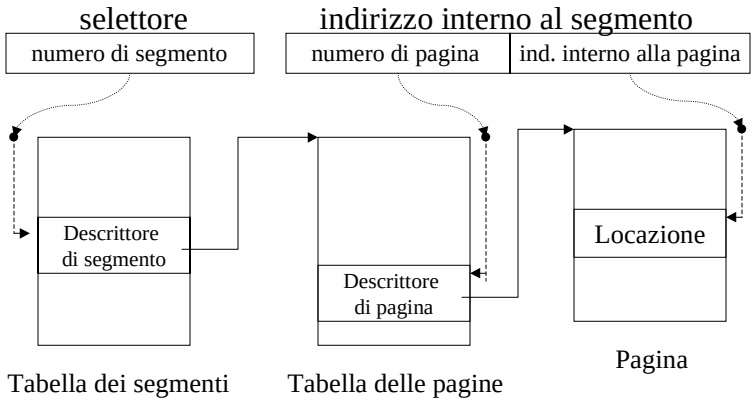
La memoria virtuale

	Paginazione	**Segmentazione**
Il programmatore ne è consapevole?	no	sì
Quanti spazi di indirizzamento lineari ci sono?	1	molti
Lo spazio di indirizzamento totale può superare la memoria fisica?	sì	sì
Prevede forme di protezione?	no	sì
E' possibile gestire semplicemente strutture dati di dimensione variabile?	no	sì
Per quale motivo viene usata?	simulare una memoria più grande	fornire più spazi di indirizzamento

26

La memoria virtuale

E’ possibile ottenere i vantaggi della memoria segmentata e quelli della memoria paginata trattando ogni segmento come una memoria virtuale ed impaginandolo



27

La memoria virtuale nel 386 Intel

Il 386 supporta:

•paginazione su domanda

•segmentazione pura

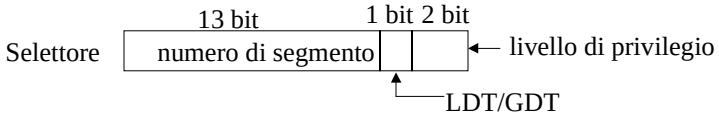
•segmentazione con paginazione

Il sistema è dotato di 2 tabelle:

•GDT (Global Descriptor Table) è condivisa da tutti i processi e descrive i segmenti di sistema

•LDT (Local Descriptor Table) ogni processo ne ha una e descrive i suoi segmenti locali (codice, dati, …)

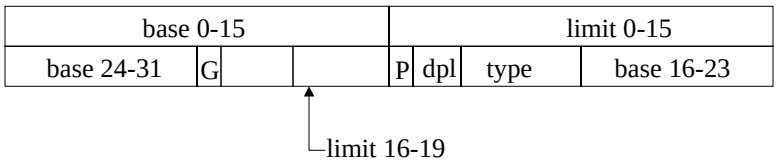
Fino a 16K segmenti distinti ognuno con indirizzi a 32bit



28

La memoria virtuale nel 386 Intel

descrittore di segmento



G (granularità) Se G=0 limit è espresso in byte, se G=1 limit è espresso in pagine

P Indica se il segmento è presente in memoria

DPL Livello di privilegio

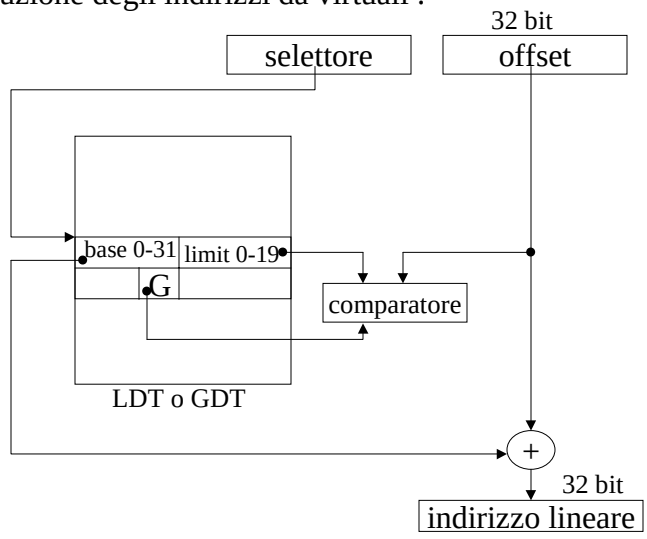
TYPE Tipo di segmento e protezione

Per individuare il descrittore di segmento è sufficiente sommare l'indirizzo di partenza di GDT o di LDT al valore contenuto nel selettore con i 3 bit meno significativi posti a 0.

29

La memoria virtuale nel 386 Intel

Traduzione degli indirizzi da virtuali :



30

La memoria virtuale nel 386 Intel

Se la paginazione è disabilitata l'indirizzo lineare viene interpretato come indirizzo fisico

Se invece la paginazione è abilitata l'indirizzo lineare viene interpretato come un indirizzo virtuale

indirizzo lineare

dir	page	offset
10 bit	10 bit	12 bit

Le pagine hanno una dimensione di 4K quindi sono necesari 12 bit per individuare una locazione all'interno della pagina

I restanti 20 bit individuano la pagina all'interno del segmento

Un segmento può contenere 1M pagine. Per limitare la dimensione della tabella delle pagine si usa una corrispondenza a due livelli

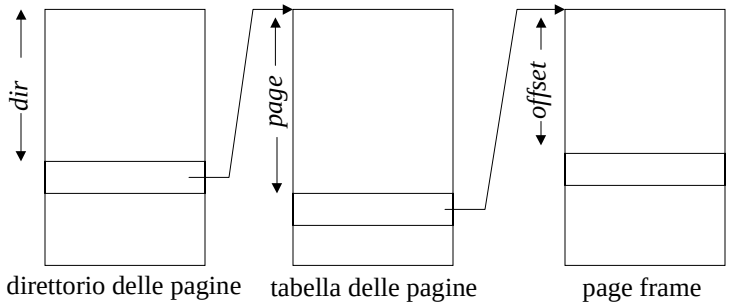
31

La memoria virtuale nel 386 Intel

Ad ogni programma in esecuzione è associato un *direttorio delle pagine*

Il direttorio delle pagine è composto da 1024 elementi ognuno dei quali punta ad una tabella delle pagine

Ogni tabella delle pagine contiene 1024 elementi ognuno dei quali punta ad un page frame (nel caso in cui la pagina sia in memoria fisica) ed alla locazione sul disco



32

L’I/O virtuale

La forma di organizzazione più comune dell’I/O virtuale è quella basata su un’astrazione detta **File**

Un file può essere definito come una sequenza di byte memorizzati su un dispositivo di I/O

L’ingresso-uscita dei dati è realizzato mediante chiamate di sistema che consentono di

•aprire/chiudere un file

•leggere/scrivere dei dati in un file

La chiamata di sistema che consente di leggere (scrivere) dei dati in un file avrà in genere i seguenti parametri:

•quale file aperto deve essere letto (scritto)

•un puntatore all’area di memoria dove immettere (da cui prelevare) i dati

•il numero di byte da leggere (scrivere)

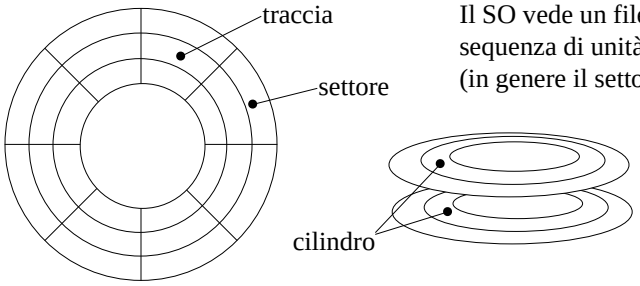
33

L’I/O virtuale

In genere l’operazione di lettura (scrittura) restituisce un valore che indica il numero effettivo di byte letti (scritti)

Ad ogni file aperto è associato un puntatore che indica il prossimo byte da leggere (scrivere). Il puntatore avanza automaticamente in funzione del numero di byte letti (scritti).

In alcuni casi un file non è una sequenza di byte ma una sequenza di record (informazioni strutturate), che costituiscono l’unità di lettura (scrittura)



Il SO vede un file come una sequenza di unità di allocazione (in genere il settore)

34

L’I/O virtuale

Il file può essere memorizzato come

•una sequenza di settori consecutivi (CD-ROM)

•un insieme di settori non consecutivi

Nel caso in cui il file sia costituito da un insieme di settori non consecutivi il SO deve essere in grado di localizzarli:

•le unità di allocazione di un file sono organizzate come una lista

•una tabella contiene tutte le coordinate delle unità di allocazione del file

Il sistema può tenere traccia dello spazio libero su disco in due modi:

•una lista che indica il punto di partenza e la dimensione di un’area libera

•una bit-map che contiene uno 0 se l’unità di allocazione è libera, 1 se occupata

35

L’I/O virtuale

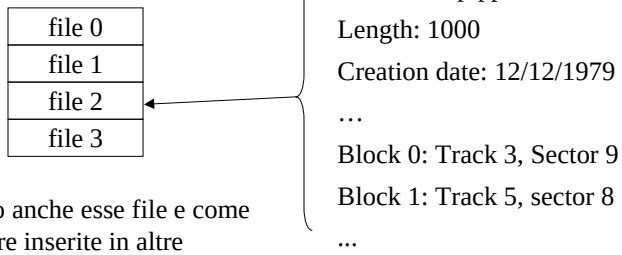
I SO organizzano i file raggruppandoli in directory

Chiamate di sistema permettono di

•creare un file ed inserirlo in una directory

•cancellare un file da una directory

•rinominare un file



Le directory sono anche esse file e come tali possono essere inserite in altre directory dando origine ad un albero

36

Unix virtual I/O

Il file system di Unix è gerarchico e comprende tre tipi di file

•*file ordinari* sequenze lineari di byte (max 2^32-1 byte)

•*directory* gruppi di file. Viene implementato come un file contenente l'elenco dei file appartenenti alla directory

•*file speciali* rappresentano dispositivi di ingresso/uscita ed altre risorse

Per accedere ad un file si deve conoscerne la posizione nel file system che viene indicata da un *path*

Un path è una espressione formata da nomi di file separati da '/'

L'ultimo nome di un path può essere un file di qualunque tipo, gli altri devono essere nomi di directory

37

Unix virtual I/O

In ogni istante, ad ogni processo è associata una *directory corrente*

I path usati da un processo sono interpretati a partire dalla directory corrente a meno che non comincino con '/'

Se il path comincia con '/' viene interpretato a partire dalla directory radice

I path che cominciano con '/' sono detti *assoluti*, gli altri sono detti *relativi*

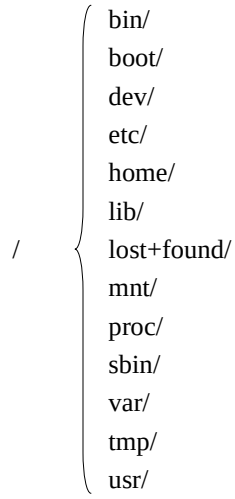
Il simbolo della directory radice è '/'

Il simbolo '.' rappresenta la directory corrente

Il simbolo '..' rappresenta la directory padre (livello immediatamente superiore)

38

Unix: il filesystem



39

Unix: accesso al sistema e protezione

Un sistema operativo multiutente deve

•permettere la condivisione di informazioni tra più utenti

•evitare che un utente danneggi le informazioni di un altro utente

Ad ogni utente del sistema sono associati:

•un nome utente (nome di login)

•una password

•un ID di utente (identificatore numerico)

•un gruppo (identificato da un nome di gruppo e da un ID di gruppo)

Gli utenti possono accedere al sistema mediante la procedura di login:

```
login:
password:
```

40

Unix: protezione dei file

Esiste un utente privilegiato detto superuser il cui nome di login è root ed il cui ID è 0. Il superuser è onnipotente.

Quando un utente crea un file è proprietario (owner) del file

Ad ogni file sono associati dei permessi di accesso nei confronti del proprietario, dei membri dello stesso gruppo del proprietario, di tutti gli altri utenti

L'insieme dei permessi di accesso viene rappresentato con una sequenza di 10 caratteri:

```
-rw-r--r--
```

`r` permesso di lettura

`w` permesso di scrittura

`x` permesso di esecuzione

41

Unix: il file system

I dati contenuti in un file ordinario o in una directory sono memorizzati in un numero adeguato di unità di allocazione (blocchi) generalmente non contigue

Ad ogni file è associata una struttura dati detta *inode* che contiene:

- identificatore del proprietario
- identificatore del gruppo
- tipo di file
- permessi di accesso
- dimensione del file in byte (su 32 bit)
- contatore dei link
- 13 indirizzi di blocchi su disco

Gli inode sono memorizzati in una tabella memorizzata su disco detta *ilist*

42

Unix: il file system

I 13 indirizzi di blocchi su disco sono organizzati come segue (supponiamo 1 blocco=1Kbyte):

•per file con una dimensione inferiore a 10Kbyte, gli indirizzi indicano la locazione dei blocchi che contengono i dati

•per file con una dimensione superiore a 10Kbyte, l'undicesimo indirizzo punta ad un *blocco di indirezione* il quale contiene 256 indirizzi. Questi indirizzi puntano alle locazioni su disco dei blocchi dati del file. (file fino a 10Kbyte+256Kbyte)

•per file con una dimensione superiore a 266Kbyte, il dodicesimo indirizzo punta ad un *blocco di indirezione doppia* il quale contiene gli indirizzi di 256 blocchi di indirezione. (file fino a 10Kbyte+256Kbyte+64Mbyte)

•per file con dimensione superiore a ….

Il limite pratico è però di 4Gigabyte (dimensione su 32 bit)

43

Unix: il file system

Un file directory contiene una sequenza di strutture dette link formate da un nome di file e dal numero di inode corrispondente

Più nomi, eventualmente in directory diverse, possono riferire lo stesso inode (e quindi lo stesso file)

Il contatore dei link contenuto nell'inode indica il numero di nomi con cui ci si può riferire al file

Alcuni comandi che operano sui link:

ln aggiunge ad una directory un nuovo link per un file

mv rimuove un link e ne crea uno nuovo

rm rimuove un link. Se il link rimosso era l'ultimo a riferire un file, l'inode corrispondente e le unità di allocazione vengono liberati

44

Unix virtual I/O

Alcune chiamate di sistema:

```
int creat(char *path, int perms)
int open(char *path, int flags, int perms)
int close(int fd)
int read(int fd, char *buf, unsigned nbytes)
int write(int fd, char *buf, unsigned nbytes)
long lseek(int fd, long offset, int base)
int mknod(char *path, int mode, int device)
```

`creat` ed `open` restituiscono un intero detto descrittore di file che viene poi utilizzato per identificare il file nelle primitive `close`, `read`, `write`, `lseek`

45

Unix:il file system

I descrittori di file con numeri 0,1,2 sono speciali e si riferiscono rispettivamente a

- lo *standard input* (in genere la tastiera)
- lo *standard output* (in genere il display)
- lo *standard error* (in genere il display)

Molti programmi (detti *filtri*) prendono i dati di ingresso dallo standard input e forniscono i dati di uscita sullo standard output

46

Unix: l'interprete di comandi

Dopo aver completato la procedura di login l'utente interagisce con l'interprete di comandi (detto shell)

La shell attende che l'utente inserisca un comando, lo esegue e torna in attesa di un nuovo comando

Per terminare una sessione di lavoro l'utente deve scrivere il comando `exit` o `ctrl+d`

Una semplice forma di comando è un path di un file eseguibile, eventualmente seguito dagli argomenti

Se il path è costituito dal solo nome del file, la shell lo cerca in un insieme di directory predefinite

Alcuni argomenti sono detti opzioni e cominciano con il carattere '-'

47

Unix: l'interprete di comandi

La shell riconosce la tastiera ed il il video del terminale da cui è stata attivata come l'ingresso standard e l'uscita standard

E' possibile redirigere l'ingresso o l'uscita di un comando:

`>` redirezione dell'uscita standard

`<` redirezione dell'ingresso standard

`>>` redirezione dell'uscita standard concatenandola al contenuto di un file

`2>` redirezione dell'errore standard

Ad esempio:

```
$ ls > pippo
```

48

Unix: l'interprete di comandi

L'uscita standard di un comando può essere utilizzata come ingresso standard di un altro comando

Questa modalità viene detta pipe o pipeline e si indica interponendo il carattere '|' tra i due comandi

I due comandi vengono sincronizzati in modo tale che il secondo aspetti i dati prodotti dal primo, mentre il primo attende che il secondo li abbia letti

Ad esempio:

```
$ ls | sort
```

49

Unix: l'interprete di comandi

E' possibile riferirsi a più file usando maschere formate da caratteri normali e wildcard:

* sostituisce un numero qualunque di caratteri (anche 0)

? sostituisce un solo carattere

[,] sostituisce caratteri appartenenti ad una certa classe

Ad es:

```
$ ls *.txt
```

Il carattere '.' è trattato in modo speciale quando è il primo carattere di un nome di file e non può essere usato per espandere le wildcard

I file il cui nome comincia con '.' non vengono visualizzati dal comando `ls` a meno che non venga usata l'opzione `-a`

50

Unix: l'interprete di comandi

Se il comando viene fatto seguire dal carattere '&', l'interprete dei comandi lo esegue in background (la shell esegue il comando in parallelo ai comandi successivi)

```
$ gcc prova.c &
1416
```

La shell restituisce il numero del processo associato a tale comando

Per vedere quali processi sono in esecuzione si usa il comando `ps`

```
PID   TTY     TIME   CMD
1390  ttyS0   0:00   bash
1416  ttyS0   0:07   gcc prova.c
```

Per terminare un processo in background: `kill -9 PID`

Per terminare il processo attualmente eseguito dalla shell: `ctrl+c`

51

Unix: comandi principali

adduser viene usato dal superuser, o da un altro utente autorizzato, per aggiungere un nuovo utente

alias consente di creare nomi alternativi per i comandi ad esempio

```
alias ll='ls -l'
```

cat legge in sequenza uno o piu' file e li stampa sullo standard output ad es:
`cat file1` stampa sul video il contenuto di file1 (anche se e' binario)
`cat file1 file2 > file3` file3 viene prodotto concatenando file1 e file2

cd (change directory) cambia la directory corrente
`cd ..` permette di spostarsi nella directory padre
`cd` o `cd ~` permette di spostarsi nella directory home

chgrp consente di cambiare il gruppo associato ai permessi di un file o di una directory. Puo` essere usato dal proprietario del file o dal superuser. `chgrp <nuovo gruppo> <file>`

52

Unix: comandi principali

chmod cambia i permessi associati ad un file o una directory. Ci sono due modi per specificare i nuovi permessi:
codifica alfabetica

`u` proprietario	`r` lettura	`+` aggiunge
`g` gruppo	`w` scrittura	`-` rimuove
`o` gli altri utenti	`x` esecuzione	
`a` (all) tutti gli utenti		

ad esempio `chmod ug+rx file1`

codifica numerica
4 equivale a lettura, 2 a scrittura e 1 a esecuzione. Sommare i valore in modo da ottenere i permessi desiderati per ciascuna categoria di utenti. Lo stato precedente dei permessi non viene considerato

ad es: `chmod 700 file1` (lettura scrittura ed esecuzione al proprietario, nessun permesso agli altri)

chown consente di cambiare il proprietario di un file o di una directory.
`chown <nuovo proprietario> <file>`

53

Unix: comandi principali

cp (copy) consente di copiare un file `cp file1 file2`

find cerca un file nelle directory specificate come argomenti e nelle loro sottodirectory.
`find dir1 dir2 -name "*.h"` cerca in dir1, dir2, e nelle loro sottodirectory, tutti i file il cui nome termina con .h

grep cerca una data stringa nel contenuto di un insieme di file

grep <testo> <file> ad es: `grep root /etc/passwd`

gzip comprime il file passato come parametro

hostname restituisce il nome dell'host

kill invia un segnale ad un dato processo chiedendogli di terminare l'esecuzione. kill -9 <PID> termina un processo

54

Unix: comandi principali

lpr stampa il contenuto di un file (sulla stampante)

ls elenca il contenuto di una directory. Opzioni piu` comuni:

`ls -l` visualizza l'output nel formato esteso

`ls -a` elenca il contenuto di una directory includendo anche i file nascosti (quelli il cui nome comincia per punto). Esempio di `ls -la:`

```
total 1708
drwxr-xr-x    37 root     root         4096 Nov 24 13:30 .
drwxr-xr-x    17 root     root         4096 Nov 24 03:26 ..
-rw-------     1 root     root            0 Oct 20 18:02 .pwd.lock
drwxr-xr-x     3 root     root         4096 Aug 30 21:19 CORBA
-rw-r--r--     1 root     root         2434 Aug 25 16:24 DIR_COLORS
-rw-r--r--     1 root     root           21 Oct 20 17:28 HOSTNAME
drwxr-xr-x    13 root     root         4096 Oct 20 17:12 X11
-rw-r--r--     1 root     root         2516 Feb 10  2000 a2ps-site.cfg
-rw-r--r--     1 root     root        12812 Feb 10  2000 a2ps.cfg
-rw-r--r--     1 root     root           12 Aug 24 04:41 adjtime
```

55

Unix: comandi principali

man formatta e visualizza le pagine del manuale in linea relative al parametro passato come argomento. `man <nomecomando>`

mkdir crea una nuova directory `mkdir <nomedirectory>`

more visualizza un file di testo una pagina alla volta

mount monta il filesystem indicato dal file speciale (primo parametro) sulla directory specificata (secondo parametro). Quando e` usato senza parametri mostra tutti i filesystem montati. Es:

```
/dev/hda8 on / type ext2 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/hda1 on /boot type ext2 (rw)
/dev/hda5 on /home type ext2 (rw)
/dev/hda7 on /var type ext2 (rw)
```

mv (move) `mv <file1> <file2>` Se <file2> non e` una directory cambia il nome di <file1> in <file2>. Se <file2> e` una directory sposta <file1> all'interno di <file2>

56

Unix: comandi principali

passwd permette di cambiare la password

ps mostra i processi in esecuzione. Opzioni:

-a mostra anche i processi degli altri utenti

-u mostra delle informazioni aggiuntive (user, pid, %cpu, %mem, ...)

-x visualizza anche le informazioni dei processi che non sono associati a nessun terminale

pwd visualizza il nome della directory corrente

rm (remove) rimuove uno o piu` file. L'opzione `-r` fa si` che elimini i file in modo ricorsivo

rmdir elimina una directory vuota

su consente ad un utente di divenire temporaneamente un altro utente. Quando e` invocato senza argomenti permette all'utente di divenire il superuser (richiede l'immissione di una password)

57

Unix: comandi principali

tail visualizza le ultime righe di un file di testo

talk permette a due utenti di colloquiare `talk username@hostname`

tar consente di creare un archivio, di aggiungere file ad un archivio o di estrarre file da un archivio. Ad es:

```
tar cf archivio.tar /etc/
```

umount permette di smontare un file speciale dall'albero del filesystem

58