Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2022-2023

Pietro Frasca

Lezione 23

Martedì 10-01-2023

Protezione di file e directory

- Fondamentale requisito per i SO multiutente: consente di stabilire regole di accesso ai file e directory.
- Le soluzioni più comuni per l'accesso ai file si basano sui concetti di ***risorsa*** e ***dominio di protezione.***
- Le **risorse** sono gli oggetti da proteggere (file, directory, dispositivi …).
- I **domini di protezione** sono definiti come un insieme di coppie ***<risorsa, diritti>*** e generalmente sono associati ad un utente. I diritti specificano il tipo di operazione che è possibile eseguire per la risorsa. Ad esempio per i file i diritti specificano se si ha diritto di lettura, scrittura o esecuzione.
- Il sistema operativo mantiene in opportune strutture dati le risorse e i domini. Concettualmente si può pensare ad una **matrice di protezione**.

Utente\risorsa	**file1**	**file2**	**file3**	**dir1**	**stampante1**
Lino	rw	r	rw	rw	w
Mario	r	rwx		r	w
Eva				rw	

Matrice di protezione

- La matrice non è una struttura dati adatta in pratica per realizzare la corrispondenza tra risorsa e diritti, per via dell'elevato numero di utenti e soprattutto di risorse che sono presenti in un sistema. Esse sarebbero matrici sparse e di enormi dimensioni.
- Le due soluzioni più comuni consistono in:
  - **Liste di controllo degli accessi (Access Control List – ACL)**
  - **Liste di capacità (Capability List o C-list)**

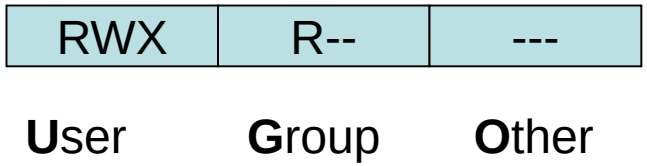
strategie

- Una **ACL** esprime la politica di protezione associata ad una risorsa, rappresenta quindi una **colonna della matrice di protezione**: per ogni risorsa **R** si elencano i permessi che ciascun utente possiede per essa. Ad esempio per la **risorsa file2** l'ACL è:

Utente\risorsa	file1	file2	file3	dir1	stampante1
Lino	rw	r	rw	rw	w
Mario	r	rwx		r	w
Eva				rw	

[lino:r,mario:rwx]

- Sia Windows che Unix adottano tecniche di protezione basate su **ACL**.
- Unix usa una tecnica di ACL più semplice poiché per ogni risorsa si specificano tre classi in cui gli utenti sono raggruppati: ***proprietario***, ***gruppo*** e tutti gli ***altri*** utenti. Pertanto, ogni ACL occupa solo 9 bit:



- Una **C-List**, invece corrisponde ad una riga della matrice di protezione. Quindi, il sistema mantiene una lista di permessi relativi alle risorse che il processo P ha diritto ad accedere.
  In relazione alla matrice di protezione dell'esempio si ha per un processo **P** dell'utente **lino**:

  [file1:rw, file2:r, file3:rwx, dir1:rw, stampante1:w]

Utente\risorsa	**file1**	**file2**	**file3**	**dir1**	**stampante1**
Lino	rw	r	rwx	rw	w
Mario	r	rw		r	w
Eva				rw	

Protezione

- La protezione in unix/linux avviene a livello di:
  - **autenticazione degli utenti**
  - **controllo dell'accesso alle risorse**

- L'accesso degli utenti al sistema consiste nella verifica di due parametri: **username** e **password**. Ogni utente è identificato nel sistema mediante un **nome utente** a cui corrisponde uno **user-id** (dato di tipo intero) e una **password** (dato di tipo string).
- L'utente **root** (detto super-user) è l'utente che ha i privilegi di amministratore di sistema, il suo **user-id** ha valore **0**.
- Gli utenti sono aggregati in gruppi. Ogni gruppo è identificato da un **nome logico** a cui è associato un numero intero: il **group-id**. Ogni utente deve appartenere almeno ad un gruppo, detto **gruppo di default**, e può appartenere a più gruppi.

Il file /etc/passwd

- L'elenco degli utenti è contenuto nel file di sistema **/etc/passwd**.
- Il file /etc/passwd ha un record composto da 7 campi; il carattere : (due punti) è il separatore di campo:
  1. User-name dell'utente
  2. Password crittografata
  3. User-id dell'utente
  4. Group-id a cui appartiene l'utente
  5. Descrizione dell'utente
  6. Home directory dell'utente
  7. Programma che viene eseguito al login

Esempio:

```
root:x:0:0:root:/root:/bin/bash
rossi:x:510:501:rossi lino:/home/rossi:/bin/sh
bianchi:x:511:501:bianchi eva:/home/bianchi:/bin/bash
```

Il file /etc/group

- definisce i gruppi ai quali appartengono gli utenti. Ogni gruppo è specificato da un record, avente il carattere : (due punti) come separatore di campo, che ha il seguente formato:

  nome_gruppo:passwd:GID:lista_utenti

- **nome_gruppo** indica il nome logico del gruppo;
- **password** se specificata (non è obbligatoria) indica la password (criptata) del gruppo;
- **GID (G**roup **ID**entifier**)** è l'identificativo numerico del gruppo.
- **lista_utenti** elenco dei nomi logici degli utenti, appartenenti ad un diverso gruppo di default, separati da virgole.

Esempio di record di /etc/group:

studenti::501

tesisti::502,bianchi, rossi

- L’accesso al sistema avviene tramite **il login**. L’utente deve digitare al prompt username e password che vengono confrontate con i rispettivi valori presenti nel file **/etc/passwd e /etc/shadow**.
- Per motivi di sicurezza le password cifrate sono memorizzate nel file shadow che è accessibile in lettura e scrittura solo al super-user (o altri utenti ai quali sono stati assegnati i diritti di accesso)
- In unix l’accesso alle risorse è basato sulle liste di controllo degli accessi **ACL** (**A**ccess **C**ontrol **L**ist): per ogni risorsa sono definiti i diritti di accesso associati agli utenti. In unix le risorse sono viste come file.

- In unix gli utenti, per quanto riguarda la sicurezza, sono distinti in tre gruppi:
  1. **Utente** (**User**) (costituito dal solo proprietario del file)
  2. **Gruppo** (**Group**) (costituito da tutti gli utenti appartenenti al gruppo del proprietario)
  3. **Altri** (**Other**) (costituito da tutti gli altri utenti del sistema che non appartengono agli altri due gruppi precedentemente descritti)
- Il **proprietario** del file stabilisce i diritti di accesso per i vari gruppi, mediante il comando **chmod**. Ad esempio il comando **chmod 755 test.c**.

- Per ogni file sono ammesse tre modalità di accesso:
  - **Lettura:** il contenuto del file può essere solo letto
  - **Scrittura:** il file può essere cancellato e il suo contenuto può essere modificato.
  - **Esecuzione**: il file può essere eseguito: in questo caso il file deve essere o un programma eseguibile o uno script. Nel caso di directory la **x** ha il significato di permettere la visualizzazione del listato della directory.
- L'ACL di ogni file è composta da 9 bit, che indica i diritti di accesso per gli utenti del sistema, visti come appartenenti ai tre gruppi: **U**ser, **G**roup e **O**ther (UGO).

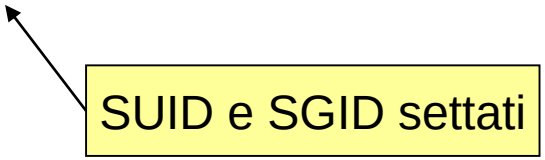
r	w	x	r	w	x	r	w	x
1	1	1	1	0	0	0	0	0
User			**G**roup			**O**ther		

- Per ognuno dei tre gruppi i diritti si esprimono con tre bit nel ordine lettura, scrittura, esecuzione (read, write, execute) i cui simboli sono rispettivamente r, w e x.
- Oltre ai 9 bit che stabiliscono i permessi di accesso, per ogni file sono specificati altri 3 bit, che hanno significato solo nel caso in cui i file sono eseguibili (programmi o script):
  - Il bit SUID (**S**et **U**ser **ID**)
  - Il bit SGID (**S**et **G**roup **ID**)
  - Il bit STIcky (**S**ave **T**ext **I**mage)
- Infatti, quando si avvia un programma il sistema inserisce nel descrittore (PCB) del nuovo processo che si crea lo **user-id** e il **group-id**, appartenenti all'utente che lancia il programma.
- E' possibile cambiare il proprietario del file settando i bit **SUID** e **SGID**. Infatti se il bit SUID è posto ad 1, al processo che esegue il file sarà assegnato lo **user-id** del proprietario del file (analogamente per SGID). In tal modo l'utente che lancia il file eseguibile assume, temporaneamente, l'identità del proprietario del file.

- Un esempio d'uso di SUID e SGID si ha nel programma **passwd**.
- Il comando passwd consente a un utente di cambiare la propria password, modificando il proprio record all'interno del file /etc/passwd (e in /etc/shadow), che per sicurezza appartiene all'utente root (superuser) e può essere modificato solo da questo. Il file eseguibile /bin/passwd ha entrambi i bit SUID e SGID settati ad 1 consentendo così ad un qualsiasi utente di assumere l'identità di root per la durata dell'esecuzione del programma e quindi di modificare il file /etc/passwd.
- Se si esegue il list del file si può vedere che è presente il carattere **s** minuscolo al posto della **x**.

- **`[frasca@localhost]$ ls -l /usr/bin/passwd`**

`-r-s--x--x  1 root root 16336 13 feb  2003 /usr/bin/passwd`



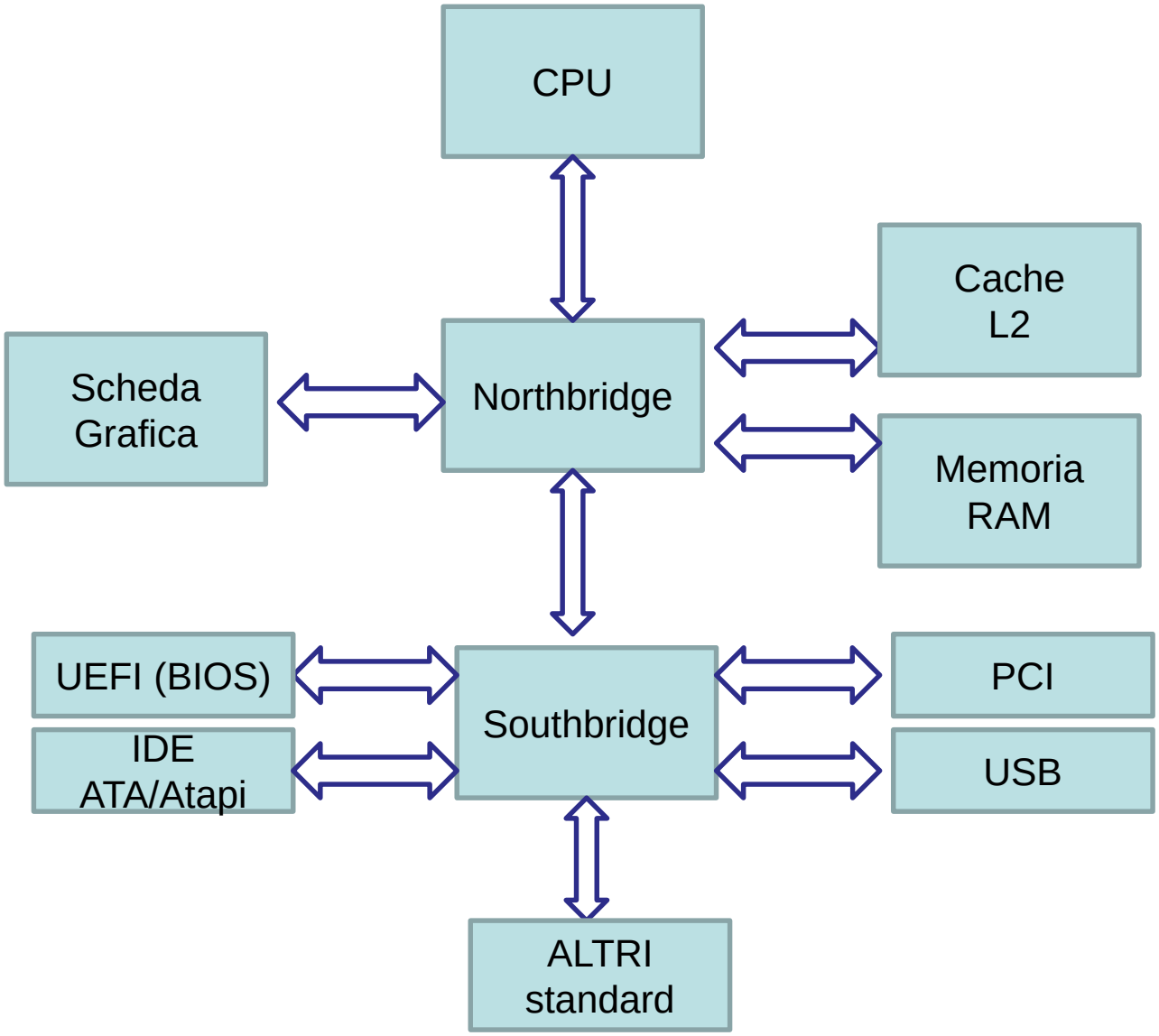
```
[frasca@localhost]$ ls -l /etc/passwd
-rw-r--r--  1 root root 21907 Dec  1 13:36 /etc/passwd
```

- Sui vecchi sistemi unix, se un file ha lo **sticky bit** settato ad 1, consente di mantenere memorizzato il codice del processo nell'area di swap, anche dopo la sua terminazione. Questo consentiva al programma di avviarsi più rapidamente. Questa caratteristica non è più usata nei moderni sistemi. Linux, ad esempio, ignora lo sticky bit sui file. Altri sistemi unix possono usare lo sticky bit sui file per scopi particolari. In alcuni sistemi, solo il superuser può settare lo sticky bit sui file. Quando lo sticky bit è usato su una directory, i file in quella directory possono essere eliminati o rinominati solo dal proprietario o da root. Senza lo sticky bit anche gli utenti che hanno accesso in scrittura a quei file possono cancellarli e rinominarli.

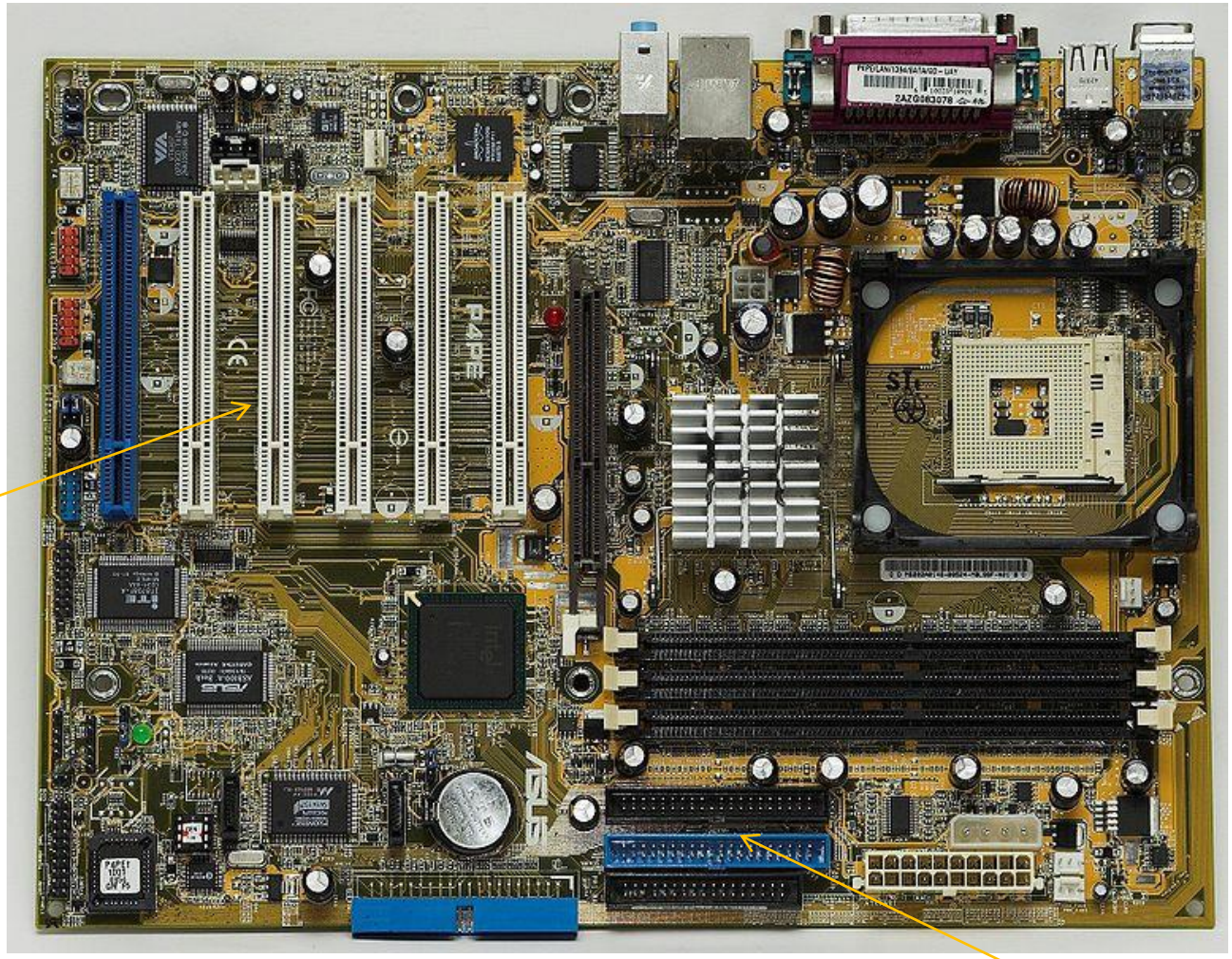
Gestione dell'I/O

- Le operazioni svolte da un processo possono essere classificate in due classi:
  - **Operazioni di I/O**
  - **Operazioni di computazione**
- Spesso, anche i processi applicativi sono classificati in base alla prevalenza di un tipo di operazione rispetto all'altra:
  - **Processi I/O-bound** (se le operazioni di I/O sono prevalenti)
  - **Processi CPU-bound** (se le operazioni di computazione sono prevalenti)

- La figura seguente mostra l'architettura semplificata di un calcolatore, in cui sono evidenziati alcuni moduli principali.



Architettura di un moderno computer



Scheda madre di un computer

Vedi dispositivi

- Ogni dispositivo è collegato ai bus di sistema tramite un interfaccia hardware (***controller***). Il controller ha il compito di gestire e controllare il dispositivo e comunica con esso tramite un proprio protocollo. Il controller è dotato di registri, che la CPU vede come locazioni di memoria (memory-mapped) o indirizza con particolari istruzioni di I/O, tramite i quali avvengono le operazioni di lettura e scrittura dei dati.
- Per comunicare con i registri del controller a basso livello, usando l'assembly sarebbe necessario conoscere l'insieme dei registri (e in dettaglio le loro funzioni) contenuti nel controller per poter gestire il dispositivo.
- Un compito fondamentale del sottosistema di I/O è fornire **un'astrazione dei dispositivi** per evitare che i programmatori debbano conoscere in dettaglio il funzionamento hardware dei dispositivi.
- Ciascuna periferica viene vista dai processi applicativi come una **risorsa astratta** alla quale è possibile accedere tramite un insieme di funzioni le **I/O API – Input Output Application Programming Interface**.

Classificazione dei dispositivi

- Le velocità con cui i dispositivi effettuano il trasferimento di dati sono molto diverse, anche di molti ordini di grandezza.

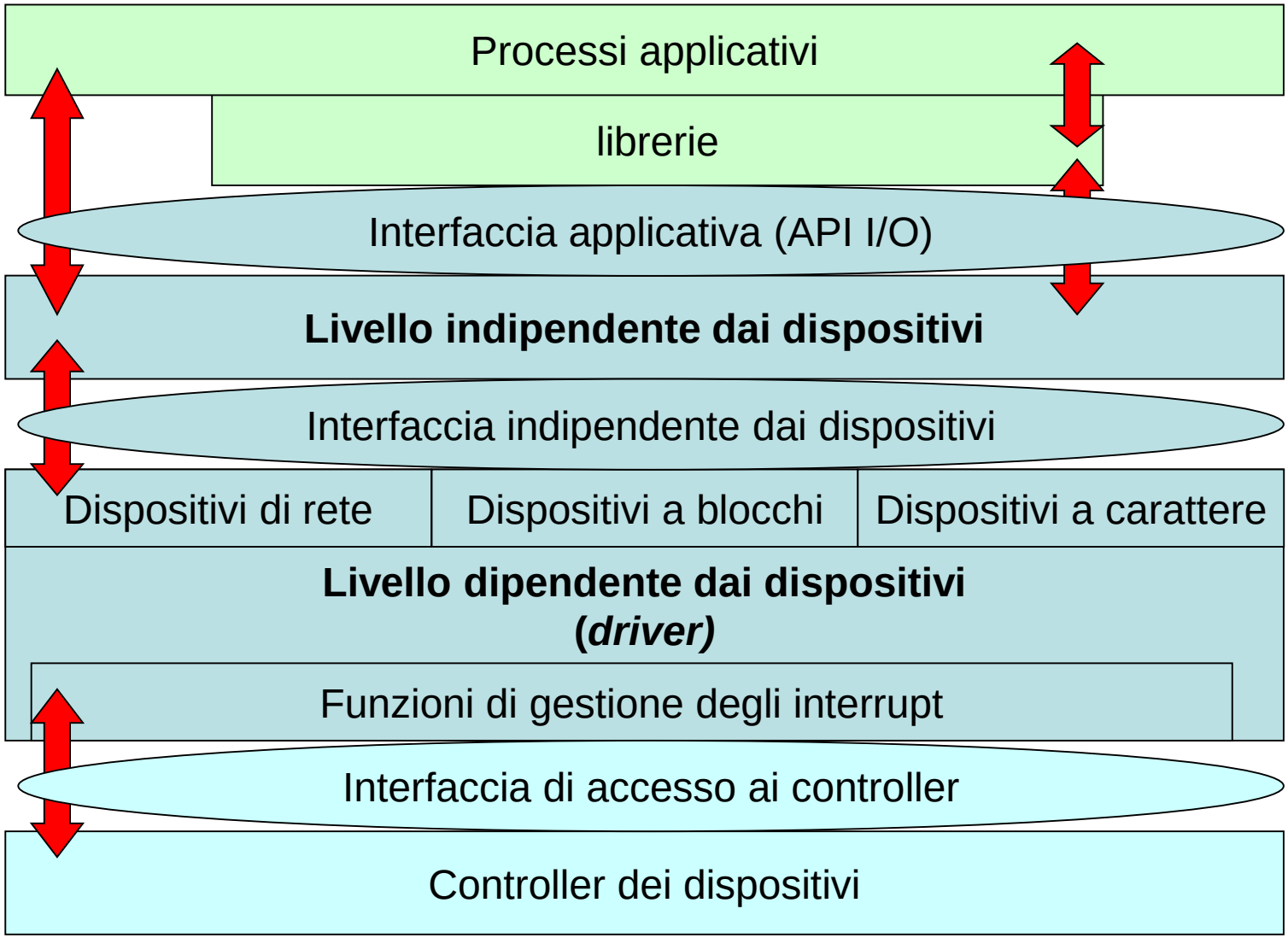
Dispositivo	Velocità di trasferimento
Tastiera	10 B/s
Mouse	100 B/s
Modem 56 Kbit/s	7 KB/s
Scanner	400 KB/s
802.11g wireless	6,75 MB/s
Cd-rom 52x	7,8 MB/s
Fast ethernet	12,5 MB/s
USB 2.0	60 MB/s
Disco SCSI ultra 2	80 MB/s
Gigabit ethernet	125 MB/s
Disco SATA 2.0	300 MB/s
Bus PCI	528 MB/s
Disco SATA 3.0	600 MB/s
USB 3.0	600 MB/s

- I dispositivi possono essere classificati in base a vari criteri, uno dei quali, particolarmente utile ai fini della loro gestione, è relativo al modo in cui sono organizzati i dati da trasferire:

  - **Dispositivi a blocchi**
    dispositivi di memoria secondaria (dischi, NVM, cd, pennette usb..) che memorizzano i dati in blocchi di dimensione fissa. Ogni blocco ha un proprio indirizzo e può quindi essere letto o scritto indipendentemente dagli altri.
  - **Dispositivi a carattere**
    i dati consistono in sequenze di byte (mouse, tastiere, stampanti, schede di rete..)
  - **Dispositivi speciali**
    dispositivi che non rientrano nelle precedenti classi, come ad esempio il timer che genera segnali di interruzione.

- A prescindere dalle tipologie di dispositivi, il sottosistema di I/O deve fornire ai processi applicativi un insieme di funzioni ad alto livello come **read** e **write**, **open** e **close** ecc.
- Ad esempio, quando un processo esegue un operazione di lettura/scrittura su un file è opportuno che il codice che il processo esegue non cambi al variare del supporto su cui il file è memorizzato.
- Il sottosistema di I/O deve anche gestire tutti i tipi di **eccezione** che si possono verificare nel trasferimento di dati.
- Un altro importante compito che deve svolgere è il **naming**, che consiste nell'identificare i dispositivi mediante nomi logici in modo tale che i processi possano fare riferimento ad essi semplicemente tramite il nome.

Struttura logica del sottosistema di I/O



Livello indipendente dai dispositivi

- Alcune funzioni del sottosistema di I/O sono indipendenti dai dispositivi.
- I servizi offerti da questo livello rendono più semplice, efficiente ed affidabile l'uso dei dispositivi.
- I progettisti tendono a rendere uniforme l'interfaccia applicativa del file system a quella dei dispositivi. Ad esempio nel sistema UNIX, un dispositivo è visto dai processi applicativi, come un file (speciale). Pertanto i comandi:
  - **cp miofile.txt lettere/miofile2.txt**
    crea una copia di miofile.txt nella directory lettere assegnandogli il nome miofile2.txt
  - **cp miofile.txt /dev/lp**
    stampa il file miofile.txt in quanto **lp** è un **file speciale** associato ad una stampante.

- Appartengono a questo livello anche le funzioni di **naming** dei dispositivi e dei file. La funzione di naming dei dispositivi ha il compito di identificare i dispositivi mediante un nome simbolico. Ad esempio in Windows, **c:** indica la directory radice di un disco indicato con la lettera c:. In unix il simbolo ***/*** indica la *root* dell'intero file system.

Bufferizzazione

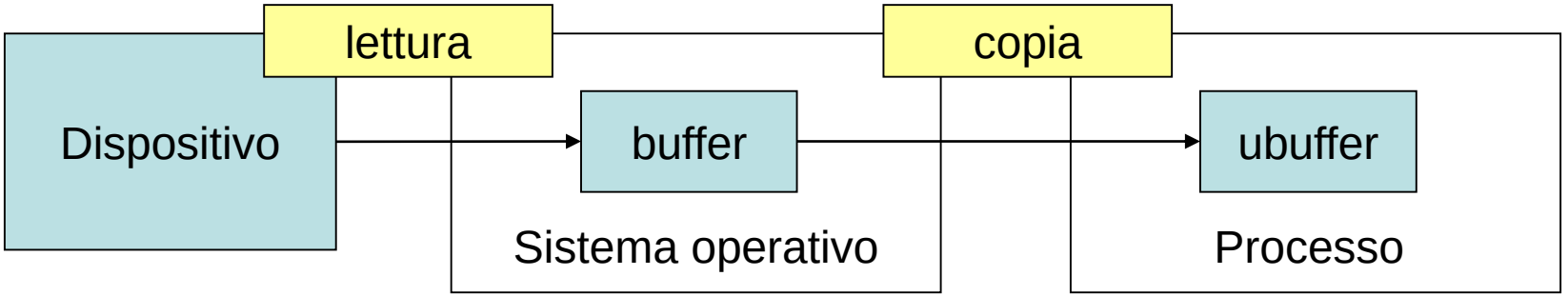
- Un'importante funzione del livello indipendente dai dispositivi è la funzione di bufferizzazione (buffering), che consiste nell'utilizzare un area di memoria gestita dal kernel detta ***buffer di sistema*** per effettuare il trasferimento dei dati, tra il dispositivo e l'area di memoria virtuale del processo applicativo.
- La bufferizzazione serve principalmente per far fronte alle notevoli differenze di velocità di funzionamento con cui operano il processore e i dispositivi.

- Consideriamo il caso di dispositivi a blocchi, come ad esempio i dischi. Supponiamo che un processo **P** esegua una chiamata di sistema di I/O del tipo:

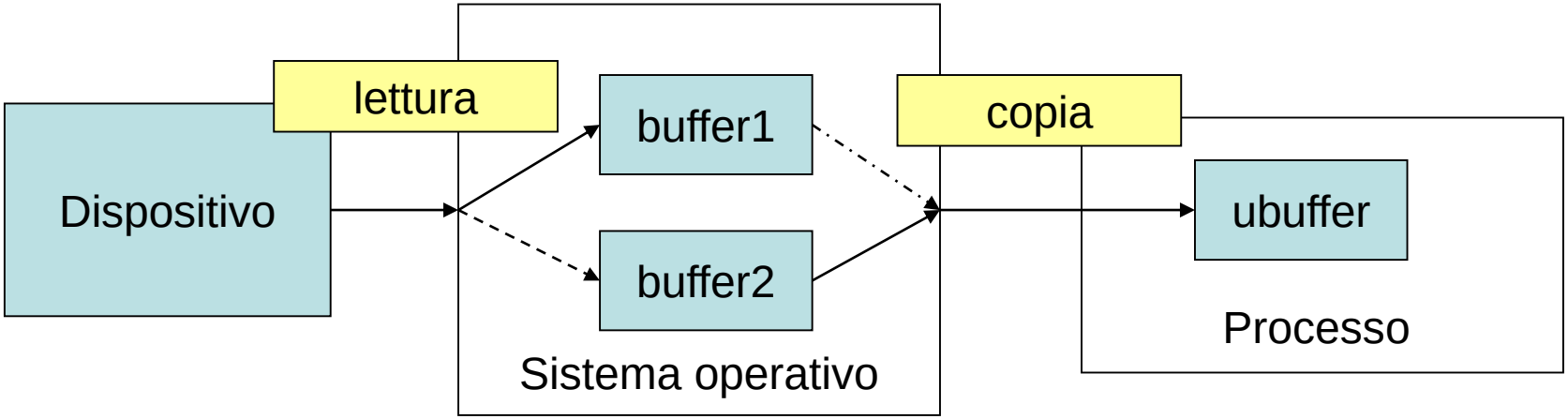
  n = read(disp, userbuffer, numbyte);

  dove, **disp** specifica il dispositivo, **userbuffer** è l'indirizzo del buffer del processo e **numbyte** indica il numero di byte che devono essere trasferiti.

  Il processo **P** eseguendo una chiamata di sistema di I/O passa nello stato di bloccato, fino al termine della chiamata che trasferirà in **userbuffer** i ***numbyte*** richiesti.
- Se la funzione **read** viene realizzata in modo che utilizzi un buffer di sistema come indicato nella figura seguente, l'operazione di lettura si realizza trasferendo i dati da disco al **buffer di sistema** e quindi da questo al buffer ***userbuffer*** dello spazio virtuale del processo.



Operazione di lettura con un solo buffer



Operazione di lettura con doppio buffer

- Al termine del trasferimento il processo può riprendere la sua esecuzione. Il vantaggio di questa soluzione si ha quando, come spesso avviene, il processo deve leggere ed elaborare una sequenza di blocchi.
- In questo caso, il tempo di attesa del processo, e il numero di commutazioni di contesto, si riduce se l'elaborazione di un blocco di dati da parte del processo viene eseguita in parallelo alla lettura nel buffer del successivo blocco di dati (read ahead).
- Un miglioramento del parallelismo si ha con il doppio buffering.

Gestione degli errori e delle eccezioni

- Durante un’operazione di ingresso/uscita si possono verificare molte eccezioni.
- Alcune di queste possono essere dovute a guasti hardware che si possono verificare nei dispositivi come, ad esempio, la rottura di una testina di lettura/scrittura di un disco.
- In altri casi, meno gravi, l‘eccezione, può essere causata da uno stato particolare in cui si può trovare un dispositivo. Ad esempio, in fase di stampa la mancanza della carta nel vassoio della stampante o, in fase di lettura di un file da un DVD, lo sportello non completamente chiuso.

- Altre eccezioni possono essere originate da errori di programmazione come, ad esempio, il tentativo di comunicare con un dispositivo non connesso al computer, o semplicemente spento, oppure il tentativo di aprire un file inesistente.
- E' necessario che tutte le suddette eccezioni siano adeguatamente gestite affinché i processi possano completare la loro esecuzione in modo corretto. Se non è possibile risolvere l'eccezione è necessario far eseguire ai processi funzioni di programma alternativi.
- Per via dell'organizzazione stratificata del sottosistema di I/O la gestione delle eccezioni è **svolto all'interno dei diversi livelli**.
- In genere è **conveniente gestire le eccezioni localmente**, a partire dal livello hardware, propagandola al livello superiore soltanto nel caso in cui non sia stato possibile gestirla completamente.

- Per esempio, molti problemi di funzionamento che sono rilevati dal controllore di un dispositivo sono dovuti a inconvenienti temporanei come nel caso di un errore in lettura da DVD causato da dalla superficie del disco polverosa e rilevato dal controllo di parità o di *checksum.*
- In questi casi è il driver del dispositivo che tenta di ripristinare il corretto funzionamento, ad esempio ripetendo più volte l'operazione. Se il driver non riesce a risolvere il problema, propaga l'eccezione al livello superiore.
- Anche nel livello indipendente dai dispositivi sono implementate routine di gestione delle eccezioni, in particolare di tutte quelle che si verificano a questo livello oltre a quelli propagati dal livello inferiore.
- In casi sfortunati l'eccezione arriva a livello di applicazione, dove è necessario gestirla per non mandare in crash il processo.
- Per rendere più robusto il software applicativo i linguaggi di programmazione, come ad esempio Java, obbligano il programmatore ad occuparsi delle possibili eccezioni.