**SOTTOSISTEMA PER L’I/O DEL KERNEL**

Il **kernel** fornisce molti servizi riguardanti l’I/O -scheduling dell’I/O , gestione del buffer, delle cache , delle code , uso esclusivo dei dispositivi , gestione degli errori- sono offerti dal sottosistema per l'I/O del kernel e sono realizzati a partire dai dispositivi e dai relativi driver.

## Scheduling di IO

Fare lo **scheduling** di un insieme di richieste di I/O significa stabilirne un ordine d’esecuzione efficace.

Lo scheduling può migliorare le prestazioni complessive del sistema , distribuire equamente gli accessi dei processi ai dispositivi e ridurre il tempo d’attesa media per il completamento di un’operazione di I/O.

I progettisti di SO realizzano lo scheduling mantenendo una **coda di richieste** per ogni dispositivo.

Quando un’applicazione richiede l’esecuzione di una chiamata di sistema di I/O **bloccante**, si aggiunge la richiesta alla coda relativa al dispositivo appropriato.

Lo scheduler dell’I/O riorganizza l’ordine della coda per migliorare l’efficienza globale del sistema e il tempo medio d’attesa cui sono sottoposte le applicazioni. Il sistema operativo può anche tentare di essere equo in modo che nessuna applicazione ricerca un sevizio carente o può dare priorità a quelle richieste la cui corretta esecuzione potrebbe essere inficiata da un ritardo nel servizio.

I kernel che mettono a disposizione l' I/O asincrono devono essere in grado di tener traccia di più richieste di I/O contemporaneamente. A questo fine alcuni sistemi annettono una tabella dello stato dei dispostivi alla coda dei processi in attesa. Gli elementi della tabella indicano il dispositivo e lo stato.

Lo scheduling dell'I/O è uno dei modi in cui il sottosistema per l'I/O migliora l'efficienza di un calcolatore.

## Gestione del buffer

Un buffer è un’area di memoria che contiene dati durante il trasferimento fra due dispositivi o tra un’applicazione e un dispositivo.

Si ricorre al buffer per 3 ragioni: la **prima** è la necessità di gestire la differenza di velocità tra produttore e consumatore. Un modem ad esempio che è nettamente più lento di qualsiasi altro dispositivo ha bisogno di una doppia bufferizzazione altrimenti si renderebbe troppo critico il problema della sincronizzazione con il disco.

Un **secondo** uso del buffer riguarda la gestione dei dispositivi che trasferiscono dati in blocchi di dimensione diversa. Questa disparità si verifica soprattutto nelle reti di calcolatori , dove spesso è necessario frammentare e ricomporre messaggi. Quando un mittente spedisce un messaggio molto lungo , esso è spezzato in piccoli pacchetti che si spediscono attraverso la rete ; il sistema destinatario provvede a ricostruire in un buffer l'intero messaggio originario.

Il **terzo** modo in cui si può impiegare un buffer è per la realizzazione della **semantica delle copie** nell’ambito dell’I/O delle applicazioni. La semantica delle copie garantisce che la versione dei dati scritta nel disco sia conforme a quella contenuta nel buffer al momento della chiamata di sistema indipendentemente da ogni successiva modifica.

## Cache

Una **cache** è una regione di memoria veloce che serve per mantener copie di certi dati: l’accesso a queste copie è più rapido dell’accesso agli originali. Ad esempio le istruzioni di un processo correntemente in esecuzione sono memorizzate in un disco , copiate nella memoria fisica e copiate ulteriormente nella cache primaria e secondaria della CPU.

La differenza tra **buffer** e **cache** consiste nel fatto che il primo può contenere dati di cui non esiste altra copia, la seconda mantiene su un mezzo più efficiente una copia d’informazioni già memorizzate.

## Code ad uso esclusivo dei dispositivi

Una coda di file da stampare(spool) è un buffer contenente dati per un dispositivo che non può accettare flussi di dati intercalati. Sebbene una stampante possa servire una sola richiesta alla volta, diverse applicazioni devono potere richieder simultaneamente la stampa di dati, senza che questi si mischino.

Il SO risolve questo problema filtrando tutti i dati per la stampante: i dati da stampare provenienti da ogni singola applicazione si registrano in uno specifico file in un disco. Quando un’applicazione termina di emettere dati da stampare, si aggiunge tale file alla coda di stampa, quest’ultima viene copiata sulla stampante, un file per volta.

In questo caso il SO fornisce **un’interfaccia di controllo** che permette agli utenti e agli amministratori del sistema di esaminare la coda ed eliminare elementi della coda prima che siano stampati, sospendere una stampa e cosi via.

## Gestione degli errori

Un SO che usi la protezione della memoria può proteggersi da molti tipo di errori dovuti ai dispositivi o alle applicazioni cosicché il blocco completo del sistema non è l’ordinaria conseguenza di piccoli difetti tecnici.

I SO sono spesso capaci di compensare efficacemente le conseguenza negative dovute a errori generati da cause contingenti. Il sistema Unix usa una variabile intera detta ***errno*** per codificare tutti gli errori.

## 

## Protezione dell’I/O

Gli errori sono strettamente connessi alla tematica della protezione.

Un processo utente che cerchi di impartire istruzioni I/O illegali può disturbare il funzionamento normale di un sistema , sia che lo faccia intenzionalmente sia accidentalmente.

Onde evitare che gli utenti impartiscano istruzioni di I/O illegali, si definiscono come privilegiate tutte le istruzioni relative all’I/O. Ne consegue che gli utenti non potranno impartire in via diretta alcuna istruzione, ma dovranno farlo attraverso il SO.

Un programma utente per eseguire I/O invoca una chiamata di sistema per chiedere al SO di svolgere una data operazione nel suo interesse.

Inoltre il sistema di protezione della memoria deve tutelare dall’accesso degli utenti tutti gli indirizzi mappati in memoria e gli indirizzi delle porte di I/O. Il kernel tuttavia non può semplicemente negare qualunque tentativo di accesso da parte degli utenti: quasi tutti i videogiochi e i programmi di grafica usano l’accesso diretto al controllore della grafica mappato in memoria. In questi casi il kernel potrebbe applicare dei **lock** per assegnare a un solo processo per volta la porzione della memoria grafica che rappresenta l'appropriata finestra sullo schermo.

## 

## Strutture dati del kernel

Il **kernel** ha bisogno di mantenere informazioni sullo stato dei componenti coinvolti nelle operazioni di I/O e usa a questo fine diverse strutture dati interne(per es. la tabella dei file aperti).

Il SO Unix, per mezzo del file system, permette l’accesso a diversi oggetti: file degli utenti, dispostivi, spazio degli indirizzi dei processi.

Quando il kernel ad esempio deve leggere un file utente ha bisogno di controllare la **buffer cache** prima di decidere l’effettiva esecuzione. Per leggere un disco tramite un’interfaccia a basso livello, il kernel deve accertarsi del fatto che la dimensione dell’insieme dei dati di cui è stato richiesto il trasferimento sia un multiplo della dimensione dei settori del disco e sia allineato con il settore interessato.

Alcuni SO quali windows NT usano metodi orientati agli oggetti in misura più rilevante sullo scambio di messaggi. Quando l’operazione richiesta è di emissione di dati(output) il messaggio contiene i dati da scrivere , quando è di immissione(INPUT) il messaggio contiene l’indirizzo del buffer che si usa per ricevere i dati.

Questo metodo può comportare una minore efficienza rispetto alle tecniche procedurali basate sulla condivisione delle strutture dati , ma semplifica la progettazione e la struttura del sistema di I/O e permette una maggiore flessibilità.

## TRASFORMAZIONE DELLE RICHIESTE DI I/O IN OPERAZIONI DEI DISPOSITIVI

Si consideri ad esempio la lettura di un file da un’unita a disco. L’applicazione fa riferimento ai dati per mezzo del nome del file: è compito del file system fornire il modi di giungere attraverso la struttura delle directory alla regione del disco appropriata, cioè quella dove i dati del file sono fisicamente residenti.

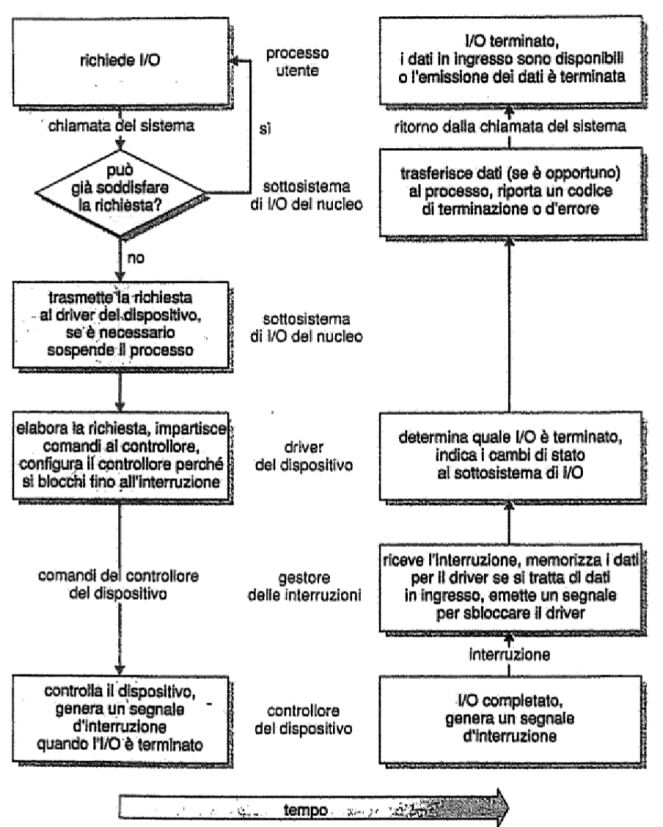
In UNIX il nome è associato a un elemento di una lista di oggetti detti **numeri di *inode***.

La prima parte del nome in DOS identifica il dispositivo e successivamente la collocazione.

Unix utilizza una tabella di montaggio per associare i prefissi dei nomi di percorso ai corrispondenti nomi di dispostivi.

La seguente descrizione del tipico svolgimento di una **richiesta di lettura bloccante** indica che l’esecuzione di un’operazione di I/O richiede una gran quantità di passi:

1. un processo impartisce una chiamata di sistema **read bloccante** relativa a un descrittore di file di un file precedentemente **aperto**.
2. il codice della chiamata di sistema all’interno del kernel controlla la correttezza dei parametri. Se sono presenti nella buffer cache, si passano i dati richiesti al processo chiamante e l’operazione **è conclusa**.
3. altrimenti è necessario eseguire un’operazione fisica di I/O così si rimuove il processo dalla coda **dei processi pronti** per l’esecuzione per inserirlo nella **coda d’attesa** relativa al dispositivo interessato. E si effettua lo scheduling della richiesta di I/O. Infine il sottosistema per l’I/O invia la richiesta al driver del dispositivo ; secondo il sistema operativo ciò avviene tramite la chiamata di una procedura o per mezzo dell’emissione di un messaggio interno al kernel.
4. il **driver del dispositivo** assegna un buffer nello spazio d’indirizzi del kernel che serve per ricevere i dati immessi ed esegue lo scheduling dell’I/O. Infine il driver impartisce il comandi al **controllore del dispositivo** scrivendo nei suoi registri.
5. il **controllore** aziona il dispositivo per compiere il trasferimento dei dati.
6. il driver può eseguire **un’interrogazione ciclica** o può avere predisposto un trasferimento DMA nella memoria del kernel(si assume che il trasferimento sia gestito dal controllore DMA , il quale genera un'interruzione al termine dell'operazione).
7. tramite il vettore le interruzioni , si attiva l’appropriato gestore delle interruzioni , che dopo aver memorizzato i dati necessari, avverte con un segnale il driver del dispositivo e restituisce il controllo.
8. il driver riceve il segnale , individua la richiesta di I/O , si accerta della riuscita o del fallimento dell’operazione e segnala al sottosistema per l’I/O del kernel il completamento dell’operazione
9. il kernel trasferisce dati(nel caso di successo) e/o codici di stato(per esempio , per comunicare la mancata riuscita dell'operazione) nello spazio degli indirizzi del processo chiamante , e sposta tale processo dalla coda d'attesa alla coda dei processi pronti per l'esecuzione.
10. nel momento in cui è posto nella coda dei processi pronti per l’esecuzione il processo non è più bloccato: quando lo scheduler gli assegnerà la CPU , esso riprenderà l’elaborazione. L’esecuzione della chiamata di sistema è completata.



## PRESTAZIONI

L’I/O è uno tra i principali fattori che influiscono sulle prestazioni di un sistema: richiede un notevole impiego della CPU per l’esecuzione del codice dei driver e per uno scheduling equo ed efficiente dei processi quando essi sono bloccati o riavviati.

Sebbene i calcolatori moderni siano capaci di gestire migliaia di interruzioni al secondo, la gestione delle interruzioni è un processo relativamente oneroso: ciascuna di loro fa si che il sistema cambi stato, esegua il gestore delle interruzioni e infine ripristini lo stato originario. Se il numero di cicli impiegato nell’attesa attiva non è eccessivo, l’I/O programmato può essere più efficiente di quello basato sulle interruzioni. Il completamento di un'operazione di I/O , in genere implica la riattivazione di un processo , comportando così lo svantaggio in termini di efficienza dovuto a un cambio di contesto.

Per eliminare cambi di contesto implicati dal trasferimento di ciascun carattere dal demone del kernel, gli sviluppatori di Solaris hanno implementato nuovamente il demone **telnet** tramite thread interni al kernel. Secondo stime della Sun, queste migliorie hanno portato il massimo numero di connessioni contemporanee sostenibili da un grande server da qualche centinaio a qualche migliaio.

Ad esempio i concentratori di terminali convogliano il traffico di informazioni proveniente da centinaia di terminali a un’unica porta di un grande calcolatore. I canali di I/O sono unità di elaborazione specializzate presenti nei mainframe e altri sistemi di alto profilo; il loro compito è sollevare la CPU di una parte del peso della gestione dell’I/O.

Per migliorare l’efficienza dell’I/O si possono applicare diversi principi:

* Ridurre il numero dei cambi di contesto
* Ridurre il numero di copiature dei dati in memoria durante i trasferimenti fra dispositivi e applicazioni
* Ridurre la frequenza delle interruzioni
* Aumentare il tasso di concorrenza usando controllori DMA intelligenti o canali di I/O per sollevare la CPU dalle semplici copiature di dati
* Realizzare le primitive direttamente tramite dispositivi fisici , così da permettere che la loro esecuzione sia simultanea alle operazioni di bus e di CPU
* Equilibrare le prestazioni della CPU , del sottosistema per la gestione della memoria , del bus e dell'I/O.

Il driver inoltre esegue raffinati algoritmi di gestione degli errori e di recupero dati e poichè hanno un’incidenza notevole sulle prestazioni complessive del sistema, svolge diverse funzioni di ottimizzazione delle prestazioni dell’unità a disco.