Ragazzi avete perfettamente ragione sul fatto che nessuno collabora!

Putroppo non posso contribuire piu di tanto dato che ho sempre preferito dedicarmi ad altri argomenti, piuttosto che seguire quelli cosi noiosi proposti dal nostro amato DTI ☺

Purtroppo il pezzo di carta serve… ☹

P.S: Giusto come idea… non sarebbe bello dare un accesso vip per quelli che fanno una piccola donazione via paypal? ☺

P.P.S: La soluzione che avete trovato non è del tutto ottimale, dato che è possibile sfruttare le cache dei search engines… dumpare il sito e mirrorarlo da qualche parte…

P.P.P.S: Ed inoltre è possibile modificare i file già Verificati, a piacimento….

P.P.P.S: E sul lato security?!

Con questo… Vi ringrazio per il duro lavoro che in questi anni avete portato avanti!

:D

^__^

Il sistema operativo **1**

Un sistema operativo è un insieme di programmi che gestiscono l'hardware in un computer, fornendo un ambiente nel quale i programmi possano funzionare bene. Fondamentalmente, esistono tre categorie di sistemi operativi: quelli orientati agli utenti, quelli orientati all'efficienza e i sistemi che sono un misto fra questi due. Nonostante questa distinzione, lo scopo primario di un sistema operativo è di essere conveniente per l'utente che lo deve usare. In questa ottica, consideriamo il sistema operativo come un distributore di risorse; esso deve occuparsi infatti di gestire i vari dispositivi, come la CPU, la memoria, i dischi, ed esercitare la funzione di programma di controllo sulle richieste contemporanee dei programmi, scegliendo quali vanno eseguite per prime e quali potrebbero creare dei problemi (es. deadlock).

Per fare questo, il sistema operativo deve essere sempre in esecuzione nel computer; da ciò, deriva la definizione di sistema operativo come, appunto, unico programma sempre in esecuzione.

L'evoluzione dei sistemi operativi risale a molto tempo fa. I primi tipi di computer erano detti computer “a lotti”, perchè eseguivano dei lotti, ossia dei gruppi, di lavori (o jobs) in sequenza. I job erano solitamente immessi come schede perforate, raggruppate in lotti da un operatore in base alle loro similitudini. Tuttavia, la lentezza dei dispositivi di I/O (la CPU lavora molto più velocemente di un lettore di schede perforate) portava allo spreco di parecchio tempo di lavoro della CPU.

Vennero progettati quindi i primi sistemi multiprogrammati, nei quali il tempo durante il quale un job effettuava operazioni di I/O era occupato dall'esecuzione di un altro job già pronto. Il passo successivo nell'evoluzione dei sistemi operativi fu la creazione di sistemi che lavoravano in time-sharing, cioè per i quali ogni job aveva a disposizione un quanto di tempo per la sua esecuzione. Se non aveva terminato la sua esecuzione in quel quanto di tempo, veniva rimesso in coda per l'esecuzione e al suo posto subentrava il job successivo.

La nascita dei sistemi multiprocessore, che permettevano una maggiore capacità di elaborazione a un costo minore di quello di due macchine monoprocessore, diede la possibilità di introdurre dei sistemi cosiddetti “fault tolerant”, cioè sistemi che resistevano ai guasti: la rottura di un processore causava soltanto un rallentamento nell'elaborazione, e non il suo completo arresto. I sistemi multiprocessore lavoravano in due modalità: Simmetrica, nella quale ogni processore eseguiva la propria copia del sistema operativo, e master-slave, nella quale un processore (master) delegava l'esecuzione di alcune operazioni ai processori slave.

Dai sistemi multiprocessore si passò ai sistemi distribuiti, cioè gruppi di macchine collegate fra loro attraverso la rete (LAN o WAN); le architetture dei sistemi distribuiti possono essere racchiuse in due grandi schemi: le architetture client-server, nelle quali un gruppo di macchine (client) esegue delle richieste a un fornitore di servizi (server), e le architetture peer-to-peer, architetture decentralizzate nelle quali le informazioni sono contenute in parte in ogni peer. In questi ambiti rientrano anche i sistemi operativi di rete.

Il passo successivo portò alla nascita di sistemi cluster, nei quali l'elaborazione è affidata a un gruppo di macchine collegate tramite veloci collegamenti di rete, ognuna delle quali esegue una parte del lavoro. I cluster lavorano fondamentalmente secondo due modalità: simmetrica, nella quale due macchine controllano a vicenda il proprio lavoro, e asimmetrica, nella quale una macchina aspetta che l'altra abbia compiuto il proprio lavoro, pronta a sostituirla in casi di guasti.

Un ulteriore tipo di sistemi operativi, sono i sistemi in real-time, ossia quei sistemi che presentano una scadenza nell'esecuzione dei lavori assegnati; i sistemi a real-time stretto (hard) presentano scadenze fisse per il lavoro, oltre le quali non si può andare (ad esempio il tempo di esecuzione di un lavoro su una catena di montaggio in movimento costante); i sistemi a real-time lasco, invece, non presentano scadenze fisse, ma danno ai lavori la priorità assoluta sul resto, in modo da garantirne l'esecuzione nel minor tempo possibile.

I sistemi operativi per palmari, infine, sono sistemi operativi progettati per lavorare con CPU poco potenti, in presenza di poca memoria e votati al massimo risparmio possibile di energia, per garantire una maggiore durata della batteria.

Architettura del computer **2**

Un computer è composto fondamentalmente da una CPU, una memoria centrale e un insieme di periferiche di I/O. La CPU è collegata alla memoria e ai controller delle periferiche tramite i bus; per segnalare alla CPU l'avvenimento di un evento hardware, vengono utilizzati degli interrupt, ovvero dei segnali appositi che vengono inviati alla CPU, la quale sospende il lavoro in corso e trasferisce immediatamente il controllo a chi ha generato l'interrupt. L'arrivo di un interrupt causa l'esecuzione di una specifica porzione di codice, che gestisce l'interrupt. Alla fine, il controllo ritorna alla CPU che riprende l'esecuzione dei compiti precedenti. Segnali analoghi, ma in ambiente software, sono le trap.

Ogni periferica di I/O ha a disposizione due metodi di interazione con la CPU: sincrono, nel quale il controllo viene restituito all'applicazione al termine dell'operazione di I/O, e asincrono, nel quale il controllo viene restituito immediatamente dopo l'accettazione dell'operazione, per poi segnalare la conclusione dell'operazione con un interrupt.

Ogni dispositivo è dotato di una tabella di stato, che indica se è occupato, libero o non funzionante, e contiene le informazioni sulle richieste fatte al dispositivo; è disponibile inoltre una coda di attesa per ogni dispositivo, per indicare l'ordine di arrivo delle richieste al dispositivo.

Il controllore del dispositivo è il responsabile del dialogo con il dispositivo fisico, e mantiene i dati da e per il dispositivo in un buffer apposito. Poiché spesso i dati da trasferire sono maggiori della dimensione del buffer, si possono utilizzare due tecniche di trasferimento dati: il PIO (I/O Programmato), con il quale la CPU cicla su un bit di stato, che viene messo a 1 quando è pronto a trasferire e rimesso a 0 quando il trasferimento è completato.; l'altra modalità è il DMA (Direct Memory Access), implementato soprattutto per dispositivi ad alta velocità: i dati da trasferire vengono memorizzati in grandi blocchi; quando il buffer è pieno, viene generato un interrupt per segnalare l'inizio del trasferimento, che avviene direttamente in memoria, senza l'intervento della CPU.

La memoria centrale è collegata alla CPU direttamente, tramite i bus. Ha la caratteristica di essere molto veloce, ma piccola e volatile (se manca l'alimentazione, il suo contenuto viene perso). Le operazioni su di essa (load e store) vengono effettuate su parole di memoria identificate da indirizzi univoci. Esistono anche degli indirizzi speciali che, sebbene sembrino degli indirizzi di memoria, sono in realtà dei buffer specifici di altri dispositivi esterni alla memoria. Nonostante la sua velocità, l'accesso alla memoria centrale è comunque lento. Per questo sono messe a disposizione delle cache, ossia delle memorie molto più piccole della memoria centrale, ma molto più veloci.

Il caching funziona memorizzando i dati in una memoria più veloce di quella centrale, nella quale i dati vengono cercati prima di richiederli alla memoria centrale stessa. È necessario però, soprattutto in sistemi multiprocessore con cache condivisa, garantire che i dati letti dalla cache siano i più aggiornati: bisogna garantire la coerenza della cache.

Per la memorizzazione permanente dei dati, tuttavia, è necessaria una memoria non volatile; questa memoria è definita memoria secondaria: è molto più lenta della memoria centrale, ma quando manca l'alimentazione la memoria secondaria mantiene il suo contenuto. Il supporto più diffuso sono i dischi magnetici, seguiti dai nastri magnetici. La memoria secondaria è collegata al computer con un bus di I/O; il PC invia una richiesta all'Host Bus Controller, che si occupa di conttatare il controller del disco e di trasferire i dati.

Possiamo definire una gerarchia fra tutti i tipi di memoria, basata sulla loro velocità (dalla maggiore alla minore) e dimensione (dalla minore alla maggiore):

I registri sono la memoria più veloce in assoluto, seguiti dalla cache (a vari livelli), dalla memoria centrale, dai dischi elettronici, magnetici, ottici e, infine, dai nastri magnetici.

Tutti questi collegamenti portano però un problema di protezione: un programma potrebbe accedere ad aree di memoria riservate o impartire comandi dannosi al sistema, anche a causa di bug. Per evitare ciò, è utile definire (almeno) due modalità di funzionamento di un programma, identificate da un bit di modalità: la modalità normale (utente) e quella privilegiata (sistema). Vengono definite istruzioni privilegiate quelle istruzioni che possono in qualche modo danneggiare l'hardware, e viene imposto che l'esecuzione di una di quelle istruzioni sia subordinata all'autorizzazione del

sistema operativo. Se un programma cercasse di eseguire un'operazione privilegiata dalla modalità utente, il sistema genererebbe una trap. Per passare alla modalità privilegiata, il programma deve effettuare una chiamata di sistema apposita: la chiamata di sistema attiva un controllo di correttezza e legalità delle istruzioni e, se lo sono, esegue le istruzioni in modalità privilegiata.
Tutte le operazioni di I/O sono definite come privilegiate, perchè è necessario controllare che un programma non cerchi di accedere a locazioni di memoria riservate. Per garantire questa protezione, è necessario assicurarsi che il programma utente non possa mai essere eseguito in modalità supervisore (ossia che non ci siano delle istruzioni, ne programma, capaci di invocare un metodo in modalità supervisore e poi continuare l'esecuzione del programma con la modalità privilegiata).
Un metodo utile per implementare la protezione della memoria è quello di fissare un registro base e uno limite contenenti rispettivamente la locazione minima di memoria e il numero di locazioni alle quali il programma in esecuzione può accedere. Se il programma cerca di accedere a una locazione il cui indirizzo è minore di quello nel registro base o maggiore della somma fra i registri base e limite, esso viene interrotto; inoltre, il programma non deve essere in grado di modificare la routine di gestione degli interrupt (perchè la modifica di quest'ultima gli permetterebbe di modificare, ad esempio, il comportamento del sistema nel caso cercasse di accedere a una locazione di memoria riservata.
L'unione di più computer in reti, porta infine alla definizione delle strutture di rete: sono sostanzialmente di due tipi: LAN (Local Area Network), piccole reti (uffici, abitazioni) collegate tramite collegamenti ad alta velocità (10/100/1000Mbps) e WAN (Wide Area Network), reti di città, paesi, nazioni, caratterizzate da basse velocità (da 56Kbps a 10/20Mbps) e basate sulle reti telefoniche esistenti.

Il sistema operativo è un programma complesso formato da più parti (8).
I processi sono i programmi in esecuzione su un computer: essi costituiscono l'unità di lavoro di un computer e necessitano di risorse (CPU, Memoria, Periferiche). Il sistema operativo si deve occupare della loro generazione e cancellazione, del controllo, della sospensione e riattivazione, della sincronizzazione, della comunicazione e dell'individuazione dei deadlock.
La memoria centrale è il “posto” in cui passano tutti i dati da e per la CPU. Essa contiene i programmi che devono essere eseguiti, e sono necessari diversi algoritmi per la sua corretta gestione. Il sistema operativo deve sapere chi la sta usando, deve allocare e disallocare i blocchi a seconda delle necessità.
Per quanto riguarda la gestione dei file, ossia gruppi di informazioni collegate fra loro (anche i programmi sono dei file), ogni supporto di memorizzazione ha un suo metodo di stoccaggio dei file; in generale, vengono organizzati in directory, si controllano i permessi di accesso, ma in modo diverso a seconda del supporto. Un sistema operativo deve essere in grado di gestirli: crearli, cancellarli, modificarli, spostarli, salvarli in memoria stabile.
Oltre a questo, c'è da considerare anche la gestione dei dispositivi di I/O: il sistema operativo deve mettere a disposizione un'interfaccia generica per dialogare con i driver della periferica, in modo da poterla utilizzare correttamente.
È necessaria anche una parte che controlli i dispositivi di memorizzazione secondari, allocando le unità, gestendo lo spazio libero e occupandosi della schedulazione dei dischi, e una che consenta la realizzazione di reti informatiche, sistemi distribuiti nei quali diverse CPU lavorano su dati contenuti in una memoria comune: ciò permette di aumentare la capacità di elaborazione e l'affidabilità.
Infine, il sistema operativo deve fornire un sistema di protezione, che impedisca a processi diversi di sovrapporsi nell'uso della memoria e che possa essere utilizzato insieme ai sistemi di controllo dell'accesso, e un interprete dei comandi, ossia una shell, che permetta all'utente di fornire dei comandi all'elaboratore.
Queste otto parti, insieme, permettono al sistema operativo di fornire dei servizi: l'esecuzione di programmi, le operazioni di I/O, la manipolazione del file system, la comunicazione fra diversi processi, il rilevamento degli errori, l'allocazione delle risorse in contesti multiutente, la contabilità (cioè la conoscenza dei consumi di risorse del sistema) e la protezione e la sicurezza.
L'interfaccia fra il sistema operativo e i processi è data dalle chiamate di sistema: esse sono generalmente delle specifiche istruzioni assembler, ma anche per effettuare una semplice operazione, spesso, ne servono molte. Per questo motivo, sono state implementate nei linguaggi di alto livello. Ce ne sono diversi tipi: quelle per il controllo dei processi (ad esempio per il debug di programmi che non terminano normalmente, o per l'esecuzione di altri programmi), per la gestione dei file e dei dispositivi (questi ultimi, spesso, sono trattati come file stoccati ad indirizzi di memoria speciali), per la gestione delle informazioni e per le comunicazioni, attraverso lo scambio di messaggi fra processi (che richiede però un protocollo di comunicazione), o attraverso la condivisione di un'area di memoria (previo consenso del processo a cui appartiene l'area di memoria in questione).
Sono disponibili anche programmi di sistema adibiti a compiti particolari, quali la gestione dei file, la fornitura di informazioni di stato, il supporto ai linguaggi di programmazione (tramite assembler, linker e compiler), il caricamento e l'esecuzione di programmi e le comunicazioni.
Per la creazione del sistema operativo, è necessario però definirne una struttura; generalmente, i sistemi operativi sono divisi in blocchi, che possono comunicare in diversi modi. La struttura più semplice è quella stratificata, una struttura a livelli con la quale un livello può chiamare soltanto metodi del livello immediatamente inferiore (e quindi essere chiamato da metodi del livello immediatamente superiore). Questo, però, può creare problemi ad alcune operazioni, che in alcuni casi necessitano di chiamate di “livello superiore” o di essere chiamate da livelli “inferiori”. Per questo motivo, si è sviluppata un'altra struttura: quella a microkernel: il kernel è ridotto all'essenziale e tutte le parti non necessarie vengono implementate come programmi a parte. Questo

però può portare a una riduzione delle prestazioni. Un'altra struttura, quella più usata, è quella modulare: il kernel è estendibile grazie a moduli specifici per un sistema operativo, che garantiscono l'estensibilità e, allo stesso tempo, un minore carico sul sistema. La soluzione ottimale per la sicurezza, tuttavia, sarebbe quella delle Virtual Machines: in un sistema così progettato, a ogni programma in esecuzione viene “fatto credere” di essere l'unico programma in esecuzione sulla macchina, garantendone l'isolamento da tutti gli altri processi. Vengono messe a disposizione del processo due modalità utente e supervisore “virtuali”, che vengono gestite dal vero sistema operativo, il quale simula la risposta alla chiamata di sistema.
Una tecnologià che sfrutta questa tecnica è JAVA, nella quale una Virtual Machine compilata per una specifica macchina interpreta un bytecode specifico. Questo garantisce un'assoluta portabilità dei programmi.
In definitiva, la progettazione e l'implementazione di un sistema operativo deve essere preceduta da un'attenta definizione di specifiche e obiettivi: vanno separate le politiche (“cosa fare”) dai meccanismi (“come farlo”) e va garantita l'implementazione per tutte le specifiche configurazioni hardware. Il sistema operativo va generato per ogni possibile variante, e va deciso se la sua implementazione debba essere fatta a basso livello (garantendo maggiori prestazioni ma legando necessariamente il sistema operativo all'architettura del computer) o ad alto livello (garantendo la portabilità a scapito delle prestazioni).
L'avvio del sistema, o bootstrap, può avvenire in diversi modi: in una fase soltanto (il boot loader carica in memoria il sistema operativo), in due fasi (il bootloader carica un loader più complesso che a sua volta carica il sistema operativo), o in più fasi (ci sono diversi loader uno più complesso dell'altro).

Processi **4**

I processi sono i programmi in esecuzione su un computer. Possono trovarsi in diversi stati (in attesa, in esecuzione, pronto all'esecuzione, terminato) e sono composti non solo dal codice del programma, ma anche dal contenuto dei registri, dagli stack di memoria che stanno usando, dai dati che producono, dal Program Counter e da un eventuale heap. Le informazioni su un processo sono conservate nel Process Control Bloc (PCB), insieme ad altre informazioni quali la schedulazione del processo, l'uso della memoria e lo stato dell'I/O. Un processo può essere composto da più thread, cioè da più flussi di operazioni. Quando la CPU passa da un processo all'altro (tipicamente nei sistemi in time-sharing), si sta procedendo ad un cambio di contesto. In questo caso, è necessario che il PCB venga salvato in memoria e sia recuperabile (tipicamente si memorizzano gli indirizzi dei PCB in una lista nota).

Per massimizzare l'utilizzo della CPU, i processi possono essere schedulati: vengono sfruttate particolari code per dare la possibilità ad uno scheduler di decidere quale sia il miglior ordine che i processi debbano seguire per ottimizzare l'uso della CPU. Gli scheduler sono raggruppabili in due tipi principali: a breve e lungo termine. I primi sono gli scheduler della CPU, che si occupano di definire quali processi devono andare in esecuzione; i secondi si occupano del trasferimento di dati fra memoria centrale e secondaria.

Sui processi si possono eseguire due classi di operazioni: la creazione e la terminazione. Ogni processo che viene creato ha un padre, al quale viene comunicato il PID del figlio; le risorse da destinare a un figlio possono venire allocate dal sistema operativo o dal padre. Quando invece un processo viene terminato (generalmente, solo un processo o suo padre possono decidere di terminarlo), è necessario scegliere la politica da adottare con i figli di quel processo: una politica a cascata imporrebbe la terminazione di tutti i processi figli; una politica più comune, assegna un altro padre ai figli (in UNIX viene posto come padre il processo init).

I processi possono anche cooperare: il vantaggio è una maggiore velocità di elaborazione, ma sono necessari meccanismi appositi per implementarla, e la disponibilità di un'area di memoria condivisa per farli comunicare. Ogni processo, inoltre, può influenzare o essere influenzato dagli altri prcessi del sistema.

Oltre alla condivisione della memoria, vi sono altri metodi di comunicazione fra processi. Le distinzioni si possono fare sulla base del tipo di comunicazione: una comunicazione diretta presume la conoscenza fra mittente e destinatario (il destinatario sa chi dovrà comunicare con lui); una comunicazione indiretta, invece, mette a disposizione una mailbox per la ricezione dei messaggi. La comunicazione sincrona, prevede che un processo si blocchi fino al ricevimento di una comunicazione e che il mittente sia bloccato fino alla risposta; una comunicazione asincrona, invece, non è bloccante. Inoltre, c'è il problema della bufferizzazione, ossia della quantità di messaggi che un processo può ricevere; i messaggi possono essere accodati solo se c'è una coda apposita: la coda a dimensione zero blocca il mittente fino alla ricezione del messaggio; una coda limitata blocca i mittenti che cercano di comunicare con coda piena, una coda illimitata non blocca mai il mittente.

Per la comunicazione in sistemi client-server, sono a disposizione i socket, cioè i due estremi di un canale di comunicazione, attraverso i quali passano i dati, e le RPC, nelle quali un servizio ascolta su una porta predefinita, sulla quale vengono spediti degli identificatori di funzione con i relativi parametri e tramite la quale viene fornita la risposta; i dati vengono trasmessi in un formato comune (XDR) che il server e il client si occupano di tradurre nel loro linguaggio. In alcuni casi, su una porta predefinita ascolta un accoppiatore, cioè un servizio che, su richiesta, comunica al client su quale porta fare la RPC. In JAVA, è disponibile una variante delle RPC, le RMI, che permettono di invocare metodi su oggetti remoti.

Il Thread è l'unità base di utilizzo della CPU: è formato dal Program Counter, dai registri e dallo stack e condivide, se il processo è multi-threaded, il codice con altri thread appartenenti allo stesso processo.

Un processo può essere single-threaded o multi-threaded; quest'ultimo caso è il più rilevante, poiché garantisce una maggior velocità nella risposta e un discreto risparmio di risorse (creare un nuovo thread è meno costoso che creare un processo nuovo identico, per il quale bisognerebbe allocare la memoria necessaria a contenere anche il codice); inoltre, in sistemi multiprocessore, più thread possono essere eseguiti contemporaneamente sui processori a disposizione.

I thread si dividono in due tipi: i thread utente e i kernel thread; questi ultimi sono i più rilevanti, perchè sono quelli che possono effettuare chiamate di sistema. Dalle relazioni fra i tipi di kernel si individuano dei modelli di interazione: nel modello molti-a-uno, un gruppo di user thread è mappato a un solo kernel thread; questo metodo è efficiente, perchè garantisce un risparmio nell'allocazione dei kernel thread, ma nel momento in cui un user thread deve effettuare una chiamata di sistema, i restanti thread si bloccano; il modello uno-a-uno, invece, garantisce il parallelismo e che il sistema non vada mai in blocco a causa di una chiamata di sistema, ma richiede un notevole dispendio di risorse per l'allocazione di numerosi kernel thread; il modello intermedio, cioè il modello molti-a-molti, alloca un numero di kernel thread minore o uguale al numero di user thread allocati; questo sistema è vantaggioso, perchè consente di ridurre il numero di risorse necessarie e garantisce un minor numero di blocchi.

Il mutithreading, tuttavia, presenta dei lati negativi: innanzitutto, è necessario definire il comportamento per le fork dei processi, cioè se sia necessario riprodurre tutti i thread o solo una parte. La gestione dei pool di thread è poi primaria: non è consigliabile permettere la creazione di un numero illimitato di thread, perchè potrebbero esaurire le risorse del sistema. Inoltre, anche il comportamento del sistema alla cancellazione di un thread è da definire: è necessario stabilire come gestire le risorse già allocate per un thread: se un thread stava modificando dei dati, la sua uccisione immediata potrebbe lasciare dei dati inconsistenti; in determinati casi è preferibile lasciare il tempo al thread di terminare la sua esecuzione.

Se poi viene inviato un segnale a un processo, bisogna stabilire se il segnale sia bloccante o non bloccante, se sia destinato a tutti i thread o ad alcuni in particolare; in questo caso, potrebbero essere state definite delle politiche per la gestione di particolari segnali definiti dall'utente. Una politica di gestione dei segnali potrebbe prevedere la consegna del segnale ad un apposito thread “gestore”.

Inoltre, un sistema potrebbe voler schedulare i thread. Un sistema vantaggioso, in questo caso, potrebbe essere la creazione di LightWeight Processes (o LWP), dei processori virtuali sui quali schedulare i vari thread.

Schedulazione della CPU **6**

La schedulazione dell'utilizzo della CPU è necessaria per massimizzare l'uso della CPU; durante il funzionamento di un computer, infatti, si possono notare picchi di utilizzo della CPU, seguiti da picchi di I/O. La schedulazione sta alla base della multiprogrammazione; per renderla il più efficace possibile, si valutano una serie di criteri: l'utilizzo della CPU, il tempo di completamento di un processo, la frequenza di completamento dei processi, il tempo di attesa e di risposta di ciascun processo. Il responsabile della schedulazione dei processi è lo schedulatore a breve termine: esso deve scegliere l'ordine dei processi da mandare in esecuzione. È responsabile anche della politica di risposta agli interrupt: la politica può essere di tipo preemptive, cioè interrompe immediatamente il processo in esecuzione per rispondere all'interrupt, o non-preemptive, il suo contrario. Il problema principale della schedulazione preemptive è che potrebbe lasciare una risorsa in uno stato inconsistente, nel caso in cui la risorsa sia condivisa fra più processi e un processo che la sta modificando venga interrotto.

Per la schedulazione dei processi sono disponibili una serie di algoritmi. Il più semplice è “First Come First Served”, che schedula i processi in base all'ordine di arrivo (è una semplice coda). Nella classe degli algoritmi a priorità, quelli che definiscono una priorità di esecuzione per ciascun processo, troviamo “Shortest Job First”, che esegue per primi i lavori più corti, ma che introduce la necessità (non sempre possibile) della stima della velocità di un processo. L'algoritmo “Round Robin” è un algoritmo di tipo FCFS che utilizza delle time-slice per evitare a un processo di attendere troppo nella coda. L'algoritmo con code a più livelli, infine, è un algoritmo che implementa una serie di code FCFS con priorità diverse, e stabilisce che un processo inserito in una coda non possa essere spostato in un'altra. Una variante di questo algoritmo è il ACCAPL con retroazione, che permette invece lo spostamento in un'altra coda.

La schedulazione, semplice nei sistemi monoprocessore, si complica quando si ha a che fare con sistemi multiprocessore. Il carico infatti deve essere distribuito, e a tal fine sono disponibili due modalità: master-slave, in cui è un processore a eseguire i processi primari, mentre gli altri eseguono codice utente, e simmetrica, in cui la coda di ready è comune ai processori, che la esaminano e eseguono i processi.

Nei sistemi real-time, la schedulazione si complica ancora: in un sistema hard-real-time, la schedulazione è impossibile a causa di variazioni sconosciute che possono avvenire in qualsiasi momento; in generale, comunque, questi sistemi danno una priorità ai processi; nei sistemi soft-real-time, invece, vengono inseriti in punti particolari (e sicuri) del codice, dei preemption points, ossia dei punti in cui avviene un controllo delle priorità; questo, comunque, mette il sistema a rischio starvation per alcuni processi a bassa priorità.

Un altro aspetto da valutare è la schedulazione dei thread: i due scope possibili sono PCS (Process-Contention Scope), che viene usato per schedulare i thread utente e stabilisce la competizione solo fra thread dello stesso processo. La schedulazione SCS (System-Contention Scope) è invece riservata ai kernel thread e serve per decidere quale kernel thread deve essere schedulato in CPU

Per ogni algoritmo di schedulazione è necessario valutarne il costo: per fare ciò, vengono definiti dei criteri e si procede ad effettuare uno (o più) dei seguenti test: la modellazione deterministica consiste nel calcolare le prestazioni dato un carico di lavoro predefinito; il modello a reti di code analizza i picchi di lavoro per determinare matematicamente il carico; la simulazione del funzionamento è ancora più precisa, perchè consente di vedere come funzionerà l'algoritmo, mentre la soluzione più precisa (e più costosa) è l'implementazione.

Il problema più comune in cui possono incorrere più processi che lavorano concorrentemente è quello delle corse critiche, ossia della modifica indipendente di variabili comuni, che può dar luogo ad errori negli altri processi. Questo problema dipende da alcune particolari sezioni del codice, dette “sezioni critiche”, che modificano le variabili comuni. Per evitare che il problema si presenti, si possono definire tre condizioni: mutua esclusione (se più processi hanno sezioni critiche, le sezioni critiche devono essere eseguite in modo sequenziale), progresso (se non ci sono thread in sezione critica e più d'uno vogliono entrarci, solo i thread che non sono in sezione critica possono decidere chi sarà ad entrare) e attesa limitata (a nessun processo può essere rifiutato l'ingresso per più di un certo numero di volte – previene la starvation).
Per garantire queste proprietà, possono essere messe a disposizione delle specifiche istruzioni hardware, che sono garantite essere atomiche; sebbene questo sia comodo in un contesto monoprocessore (posso bloccare gli interrupt intanto che ho un processo in sezione critica), nei sistemi mutiprocessore la sua implementazione diventa particolarmente costosa. Si è ricorso così alla definizione di semafori, ossia di variabili intere (per semafori generalizzati) o booleane (per semafori mutex) che espongono soltanto i metodi acquire e release, che non possono essere eseguite in simultanea. I processi vengono tenuti in attesa fino a quando il semaforo non da la disponibilità all'esecuzione (tipicamente, viene generato un segnale quando un processo fa release). I semafori presentano comunque dei problemi, perchè possono causare deadlock e starvation.
La sincronizzazione di processi causa una serie di problemi: i limiti dei buffer possono portare alla non disponibilità di risorse; inoltre è necessario fare una distinzione fra lettori e scrittori di risorse: sebbene più lettori possano accedere simultaneamente a una risorsa, gli scrittori devono avere un lock esclusivo; infine, il problema che viene definito come “pranzo dei filosofi”, nel quale cinque processi, rappresentati come cinque filosofi, devono accedere a delle risorse, le cinque bacchette per mangiare il riso, disponibili in numero limitato; poiché ogni filosofo ha bisogno di due bacchette e può prenderne soltanto una alla volta, si potrebbe presentare la situazione in cui ogni filosofo ha in mano una bacchetta e aspetta di prendere la seconda. Questo problema rappresenta in modo chiaro i concetti di deadlock (un processo aspetta risorse che non arriveranno mai perchè un altro processo aspetta risorse da lui per sbloccarle) e starvation (un processo non ottiene mai le risorse di cui ha bisogno).
Per sopperire al problema di deadlock e starvation causato dai semafori (che peraltro non funzionano correttamente se un programma contiene errori), è stato introdotto il tipo monitor, ossia un tipo di dato che espone dei metodi che vengono garantiti come mutualmente esclusivi, e che possono operare soltanto su dati interni non accessibili dai vari thread e sui parametri passati.

Si dice che alcuni processi sono in deadlock quando si verifica un ciclo per il quale ogni processo, per liberare le risorse che trattiene, attende che un altro processo ne liberi altre. Si parla di deadlock quando si verificano contemporaneamente quattro condizioni: mutua esclusione (ogni risorsa è detenuta da un solo processo), possesso e attesa (un processo che possiede risorse attende quelle che gli sono necessarie), nessun rilascio anticipato (nessun processo rilascia anticipatamente le risorse che sta trattenendo), attesa circolare (c'è un ciclo di attesa delle risorse). Per individuarli, si utilizza un grafo di allocazione delle risorse, nel quale gli archi si dividono in due categorie: archi di richiesta (vanno dall'istanza alla risorsa) e archi di assegnazione (R->I).
I sistemi operativi possono implementare tre tipi di politiche per la gestione dei deadlock: li possono ignorare, lasciando che siano i programmatori a implementare i metodi per evitarli o gestirli (questa è la politica più diffusa, in quanto meno costosa), possono implementare un protocollo per prevenirli o evitarli o possono implementare un metodo di gestione e recupero dei deadlock. La prevenzione consiste nell'evitare che si verifichi una delle quattro condizioni per la presenza di deadlock: è impossibile evitare la mutua esclusione, ma si potrebbe imporre ai processi di chiedere risorse solo quando non ne detengono altre, oppure di ottenere tutte le risorse necessarie alla loro esecuzione prima dell'esecuzione stessa; queste politiche, che prevengono il “possesso e attesa”, sono però molto costose. Un altro metodo sarebbe l'implementazione di un meccanismo per il rilascio anticipato, facendo in modo che se un processo chiede una risorsa non disponibile, rilasci tutte quelle già acquisite, o che se chiede risorse occupate da un processo in attesa, queste gli vengano date. Infine, per evitare l'attesa circolare, si potrebbe assegnare un identificatore intero univoco ad ogni risorsa e imporre che un processo che possiede una risorsa con identificatore X possa avere accesso a risorse con intero strettamente maggiore di X.
Per evitare il deadlock, invece, il sistema deve ottenere dal processo informazioni sulle risorse che acquisirà, e sta al sistema operativo decidere se far eseguire o meno il processo; il sistema si presenta in uno stato sicuro, cioè in uno stato in cui l'allocazione delle risorse ai processi che le attendono non porti a deadlock; se dall'analisi del processo e delle risorse richieste emerge che quel processo potrebbe mettere il sistema in uno stato non sicuro, cioè uno stato in cui potrebbe verificarsi un deadlock (sebbene non sia condizione sufficiente), il sistema può rifiutare l'esecuzione. Per individuare la possibilità di deadlock sono disponibili diversi algoritmi: l'algoritmo del grafo di allocazione delle risorse (che funziona solo se per ogni risorsa è disponibile una sola istanza) inserisce un arco di prenotazione nel grafo di allocazione delle risorse; l'arco di prenotazione può diventare un arco di richiesta solo se la “trasformazione” non genera cicli nel grafo. L'altro algoritmo, il più utile, è l'algoritmo del banchiere, che, seppur meno efficiente, può funzionare anche in presenza di istanze multiple di una risorsa. Ogni processo, prima dell'esecuzione, dichiara il numero massimo di istanze che potrà chiedere di una data risorsa; l'algoritmo inizializza quattro strutture dati: Available, un array contenente il numero di istanze di una risorsa attualmente disponibili, Max, una matrice mxn contenente il numero massimo di istanze di una risorsa che un processo potrò chiedere, Allocation, una matrice mxn delle istanze di ogni risorsa allocate ad ogni processo, Need, una matrice nxm con il numero di istanze di una risorsa chieste da ogni processo. L'algoritmo del banchiere sfrutta un algoritmo di sicurezza per definire se il sistema di trovi in uno stato sicuro e un algoritmo di richiesta per definire se le richieste dei processi potrebbero portare il sistema in uno stato non sicuro.
L'ultima politica di sistema per la gestione dei deadlock è quella di rilevazione e ripristino.
Per la rilevazione dei deadlock in sistemi con istanze singole per ogni risorsa può essere utilizzata una variante del grafo di allocazione delle risorse, il grafo di attesa: se nel grafo è presente un ciclo, si è verificato un deadlock; per sistemi con istanze multiple per ogni risorsa, si usa un algoritmo simile a quello del banchiere, che controlla tutte le possibili sequenze di allocazione per i processi non completati. Un problema degli algoritmi di rilevazione è la frequenza con cui usarli: essa va adeguata al numero di deadlock che si verificano nel sistema; in nessun caso consideriamo un processo come “responsabile” del deadlock, poiché tutti i processi che l'hanno formato hanno contribuito in egual misura alla sua formazione.

Infine, vi è la necessità di una preocedura che permetta il recupero dal deadlock: per fare ciò, si possono utilizzare due metodi si possono uccidere processi (tutti quelli che hanno causato il deadlock, un metodo costoso perchè richiede la riesecuzione di tutti i processi, o uno alla volta fino alla sparizione del deadlock, metodo altrettanto costoso perchè richiede l'invocazione dell'algoritmo di rilevamento dopo ogni uccisione), oppure si possono rilasciare anticipatamentre risorse; in quest'ultimo caso, è necessario calcolare il costo dell'uccisione di ciascuna potenziale vittima, al fine di scegliere quella con il costo minore; questa dovrà necessariamente fare un rollback (che spesso si traduce in un rollback completo, cioè un abort) e potrebbe causare starvation, perchè un processo a basso costo verrebbe continuamente ucciso per evitare il deadlock.

Capitolo 9 - Gestione della memoria centrale

Concetti Generali

La memoria centrale vede istruzioni, dati, etc. solo come un flusso di indirizzi detti “indirizzi fisici”. La CPU, invece, genera quelli che sono chiamati “indirizzi logici” o “virtuali”, cioè indirizzi che non sono necessariamente corrispondenti a quelli in memoria centrale.
Possiamo ignorare allora come siano stati generati gli indirizzi, perchè sarà un componente hardware apposito a occuparsi della loro traduzione in indirizzi fisici.
I processi su disco che sono in attesa di essere caricati in memoria centrale formano la **coda di entrata**. Quando uno di questi processi viene caricato in una locazione di memoria può accedere ai dati in memoria centrale, quando termina la sua esecuzione, lo spazio che gli era stato assegnato viene dichiarato spazio libero.
Gli indirizzi che un programma genera sono simbolici (ad esempio, *il terzo indirizzo dopo questo*) e sarà un compilatore a modificarlo con l'indirizzo rilocabile, cioè un indirizzo “numerico” che identificherà una locazione di memoria. Gli indirizzi rilocabili possono essere generati in tre fasi: alla **fase di compilazione**, cioè quando il programma viene compilato. Requisito necessario è che si conosca la locazione di memoria in cui il programma risiederà, altrimenti saranno generati indirizzi non validi; alla **fase di caricamento**, cioè quando il programma è caricato in memoria; il vantaggio è che gli indirizzi dipenderanno direttamente da punto in cui il programma è stato inserito, e non sarà necessario conoscere l'area di memoria in cui verrà caricato; se cambia l'indirizzo, sarà necessario cambiare soltanto il codice utente per far corrispondere gli indirizzi; alla **fase o tempo di esecuzione**, consente infine lo spostamento del programma durante la sua esecuzione, perchè non sarà necessario sapere dove è caricato il programma, dal momento che gli indirizzi verranno generati soltanto all'esecuzione del programma; questo tipo di generazione degli indirizzi richiede però dell'hardware speciale.
Questo hardware è l'MMU, ovvero Memory Management Unit, **unità di gestione della memoria centrale**; il suo funzionamento base consiste nel prendere l'indirizzo virtuale, sommarlo a quello contenuto nel **registro di rilocazione** e ottenere l'indirizzo fisico; così facendo il programma non vede mai gli indirizzi reali nei quali è mappato.
Un'ulteriore ottimizzazione dello spazio occupato da un programma è il **caricamento dinamico**: esso consiste nel caricare una procedura soltanto nel momento in cui è chiamata. Le procedure sono mantenute tramite indirizzi rilocabili; il loader, **caricatore**, si occuperà di verificare se una procedura è già caricata ed eventualmente caricarla, per poi fornire al programma l'indirizzo della procedura. Il vantaggio maggiore è che procedure inutilizzate non sono caricate, risparmiando spazio in memoria (soprattutto le routine di gestione di errori insoliti, che non sono quasi m,ai necessarie).
Si possono inoltre usare librerie **collegate in modo dinamico:** uno *stub* di una libreria è caricato nel programma (anziché la libreria intera), contenente soltanto il metodo per richiamare la libreria vera e propria: la libreria sarà caricata in memoria una volta soltanto, a dispetto del numero di programmi che la richiedono. Questo sistema, applicato soprattutto alle librerie di sistema, è detto “delle librerie condivise”.
Un altro metodo di risparmi di spazio è quello degli **overlay**: se un programma è divisibile in più parti **indipendenti**, nel programma verrà inserito un piccolo driver di overlay, che, al termine di una parte del programma, la sovrascriverà con la successiva; questo permette, ad esempio, di caricare in memoria programmi più grandi della memoria disponibile. Il sistema operativo vedrà soltanto un aumento del traffico con la memoria secondaria, dovuto al caricamento e scaricamento delle diverse parti del programma.

Swapping
Lo swapping è la tecnica che consente lo spostamento temporaneo di un programma **in stato di riposo** nella memoria secondaria, per fare spazio ad altri programmi, per poi riportarlo in memoria centrale e continuarne l'esecuzione.
Il tempo di swapping deve essere sufficientemente grande da permettere ragionevoli quantità di calcolo durante il caricamento/scaricamento del programma. Una variante dello swapping, il **roll out/roll in**, scambia in memoria secondaria processi a priorità bassa con programmi a priorità maggiore quando questi vanno in esecuzione.
Il sistema mantiene una lista dei processi pronti, che contiene i processi sottoposti a swapping che sono in memoria secondaria

Allocazione contigua di memoria
La memoria è generalmente divisa in due parti: una riservata al sistema operativo (posizionata normalmente dalle parti del vettore di interrupt) e una per i processi utente.
È necessario proteggere la memoria, in modo che non sia possibile per un processo accedere a una locazione di memoria che non gli appartiene. A tal proposito, vengono usati un registro limite e un registro di rilocazione: il secondo contiene l'indirizzo base a cui il programma è stato caricato; l'altro, contiene il massimo numero di indirizzi che sono disponibili per un processo; se l'indirizzo generato dal programma (0x + base) è maggiore del massimo indirizzo (base + rilocazione), allora l'accesso è illegale.
Esistono diversi metodi di allocazione della memoria: il più semplice è quello di assegnare partizioni della memoria ai processi: quando una partizione si libera (terminazione del programma), viene assegnata a qualcun altro. La tabella in cui sono indicate le partizioni di memoria assegnate è mantenuta dal sistema operativo.; se lo spazio assegnato è troppo grande, viene diviso e la parte non necessaria restituita all'insieme dei blocchi liberi. Le tecniche di allocazione della memoria sono tre: *first-fit* (la migliore, assegna il primo blocco libero sufficientemente grande), *best-fit* (assegna il blocco libero più piccolo in cui il programma può risiedere; ottima, ma richiede la scansione di tutti i blocchi liberi), *worst-fit* (la peggiore, assegna il blocco libero più grande esistente; richiede la scansione di tutti i blocchi).
Un problema legato all'allocazione della memoria è la **frammentazione**: quando un blocco assegnato è maggiore della dimensione necessaria, la parte non necessaria viene restituita all'insieme dei blocchi liberi; generalmente, però, questa parte è molto piccola (in particolare con *best-fit*); l'insieme di parti piccole può anche essere sufficiente per un programma, ma non è detto che ci sia un singolo blocco libero abbastanza grande (**frammentazione esterna**); la soluzione alla frammentazione è la **compattazione**, che sposta i blocchi di memoria occupati e crea un unico blocco libero. Purtroppo, la compattazione è un metodo costoso. Un'altra via è concedere a un processo di risiedere in blocchi di memoria non contigui. Le tecniche usate per fare ciò sono la paginazione e la segmentazione.
Un altro tipo di frammentazione è quella **interna**: dal momento che la tabella necessaria per memorizzare lo spazio libero potrebbe occupare più dello spazio libero stesso, per blocchi particolarmente piccoli, si tende ad allocare blocchi di dimensione fissa; questo, però porta allo spreco di spazio, che non può essere allocato a nessun altro.

Paginazione
La paginazione è uno dei metodi che consentono di allocare a un processo memoria non contigua; il metodo base è l'allocazione di un certo numero blocchi di dimensione fissa (**frame**, o **pagine**) a un processo, a seconda delle necessità di memoria di quest'ultimo. Ogni indirizzo che la CPU genera è composto da un numero di pagina + lo spiazzamento all'interno della pagina. Il numero di pagina viene confrontato con una tabella delle pagine, che restituisce l'indirizzo fisico della locazione di memoria a cui quella pagina comincia. È necessario porre attenzione nella scelta della dimensione delle pagine: una dimensione grande porta a una maggiore frammentazione interna, ma una piccola dimensione porta a una crescita inaccettabile della dimensione della tabella delle pagine.

La tabella delle pagine può essere implementata in diversi modi: per un hardware con poche pagine, è possibile mantenerla in registri dedicati, ma se ci sono parecchie pagine è necessario mantenerla in memoria, a causa del costo troppo elevato; un registro apposito (*Page-Table Base Register, PTBR*) mantiene l'indirizzo base della tabella delle pagine; quando cambia il processo, viene cambiato solo questo registro. L'accesso continuo alla memoria centrale è però costoso (due accessi a memoria per ogni indirizzo generato), un problema che si è arginato usando un TLB (*Transaction Look-aside Buffer*), una memoria associativa ad alta velocità nella quale è caricata solo una parte della tabella delle pagine; se la pagina cercata è presente nella TLB, l'indirizzamento è immediato; altrimenti, è necessario recuperare dalla memoria centrale il numero di pagina corretto. Alcuni TLB mantengono un ASID, un identificatore del processo, che permette di individuare gli accessi illegali alla memoria. Se la TLB non supporta gli ASID, deve essere ripulita dopo ogni cambio di contesto, per evitare che processi successivi accedano a indirizzi che non sono loro riservati.
La protezione della memoria centrale in un ambiente paginato si ottiene tramite bit di protezione associati a ogni frame, identificando determinate pagine come read-only; altrimenti, si usano bit di validità per capire se la pagina appartiene al processo o meno; alcuni sistemi, infine, possiedono un PTLR, *Page-Table Length Register*, che indica la dimensione della tabella delle pagine viene controllato a ogni accesso per evitare che un processo acceda a indirizzi non validi.
La tabella delle pagine può essere costruita in diversi modi: la paginazione gerarchica prevede che l'indirizzo logico sia diviso in tre parti: due numeri di pagina (uno per indicare la pagina in cui cercare la seconda tabella delle pagine, e il secondo per indicare quale pagina sia, in questa seconda tabella, quella fisica corrispondente) e lo spiazzamento.
Si può inoltre usare una tabella con hashing: l'hash trovato corrisponde a una lista di frame, che vengono confrontati uno a uno con il numero di frame contenunto nell'indirizzo, fino a trovare quello corretto.
La tabella delle pagine invertita, invece, contiene un elemento per ogni pagina fisica in memoria centrale; la tabella delle pagine del sistema è unica e il processo può accedere soltanto alle pagine contarssegnate dal proprio identificatore di processo.
La paginazione consente anche l'uso di pagine condivise: se più processi condividono del codice rientrante (cioè codice che non effettua modifiche a se stesso), la pagina può essere condivisa fra quei processi (risparmiando prezioso spazio di memorizzazione).

Segmentazione

La segmentazione è un metodo alternativo alla paginazione per la gestione della memoria centrale. Il metodo base di gestione della segmentazione consiste nell'allocare ai processi lo spazio di cui hanno bisogno, associare a quello spazio un numero di segmento e considerare gli indirizzi logici generati dalla CPU come numero del segmento +spiazzamento.
È necessario predisporre una tabella dei segmenti, contenente una base del segmento e un limite del segmento, che identificano l'indirizzo di inizio del segmento e la sua dimensione massima. Così possiamo proteggere la memoria dagli accessi illegali; inoltre, possiamo condividere i segmenti fra più processi (sempre a patto che il codice che li usa rientrante). La segmentazione viene gestita tramite gli algoritmi *best-fit* o *first*-fit, e da origine a un problema di frammentazione esterna.

Segmentazione con paginazione

Per controbilanciare gli svantaggi dovuti alla segmentazione e alla paginazione, si è provato ad usarle in contemporanea: ogni segmento viene paginato; vengono mantenute una *Local Descriptor Table*, contenente i segmenti riservati a un processo, e una *Global Descriptor Table*, contenente i segmenti condivisi fra i processi. L'indirizzo logico generato è una coppia <selettore, spiazzamento> nella quale il selettore è composto dal numero di segmento, un bit per indicare se si trova in GDT o in LDT e i bit di protezione.
Tramite la segmentazione con paginazione, si vanno a ridurre sia la frammentazione interna che quella esterna.x

Capitolo 9 - Gestione della memoria centrale

Ambiente

I programmi, per funzionare devono per forza essere in memoria centrale, ma non sempre tutto il codice del programma è usato.
La memoria virtuale consente la separazione della memoria logica da quella fisica, lo spazio di indirizzamento virtuale rappresenta il modo logico (virtuale) con cui un programma è inserito in memoria; la MMU si occuperà della mappatura logica → fisica.
Altro vantaggio della memoria virtuale è la possibilità di condividere codice.

Richiesta di paginazione

La memoria virtuale viene generalmente implementata con la richiesta di paginazione (ma poò essere anche segmentata).
In un sistema con richiesta di paginazione si usa un *lazy swapper*, che non scambia una pagina in memoria se non è strettamente necessario (per quanto il nome sia ambiguo: in questo caso si tratta più di un paginatore che di uno scambiatore).
Al paginatore spetta supporre quali pagine dovranno essere trasferite in memoria, per evitare di caricare pagina che non serviranno. Per fare ciò, si usa un bit di validità: se è 1, la pagina è in memoria ed appartiene a quel processo. Se il processo cerca di accedere a un pagina con bit di validità 0,. provoca una *page fault*: a questo punto, il sistema operativo procederà a verificare se la pagina richiesta appartiene al processo, oppure se si tratta di un accesso illegale a memoria; nel secondo caso, l'esecuzione si interromperà; nel primo, la pagina verrà caricata in memoria e l'esecuzione potrà continuare.
Una variante della richiesta di paginazione, è la richiesta pura di paginazione, nella quale nessuna pagina è in memoria all'avvio del processo, pertanto solo le pagina necessarie all'esecuzione verranno caricate, perchè ogni richiesta di una nuova pagina provocherà un page fault.
L'hardware necessario è lo stesso della paginazione normale (page table, swap).
È necessario che la richiesta di paginazione non infici l'esecuzione, ma che sia possibile riprendere l'esecuzione quando la pagina è stata caricata.
La richiesta di paginazione influisce parecchio sulle prestazioni del computer: il tempo di accesso effettivo dipende dalla probabilità che si verifichi un page fault; è una buona idea far eseguire alla CPU altro calcolo mentre le pagine vengono scambiate: un page fault ogni 1000 accessi, rallenterebbe il computer del 40%.

Copia durante la scrittura

La tecnica di *copy-on-write* consente a due processi (padre e figlio) generati da una fork di condividere il codice del padre fino al momento in cui non sarà necessario che uno dei due modifichi i dati condivisi. A quel punto, verrà creata una copia dei dati e ambedue lavoreranno sulla propria.
La pagina libera verrà presa da un pool del sistema operativo e riempita con zeri (*zero-fill-on-demand*) per evitare che i dati di altri processi vengano erroneamente usati.
La funzione di UNIX vfork() non usa la copy on write, ma condivide lo spazio di indirizzamento. Qualsiasi modifica verrà vista da ambedue i processi (padre e figlio).

Sostituzione della pagina

Un problema dell'allocazione delle pagine è la sovra-allocazione: se un processo durante la sua esecuzione userà meno pagine di quelle allocate, ci sarà spazio sprecato. Per ovviare a questo problema, si può pensare di sostituire una pagina.
Di base, la sostituzione di una pagina avviene scegliendo un frame vittima, mettendolo su disco e usando lo spazio liberato per il frame di cui si ha bisogno. In questo caso, però, è necessario trasferire due pagine, aumentando il tempo di accesso effettivo. Per ridurlo, si può utilizzare un it di modifica: si tenderà a scegliere pagine che non sono state modificate (e che non necessitano quindi del trasferimento in memoria), mentre se la pagina è stata modificata, sarà necessario trasferirla in memoria prima di sostituirla.
Per allocare la pagina e per sostituirla sono necessari due algoritmi appositi.

Per la sostituzione, il sistema base è FIFO, che però per alcune particolari richieste, nonostante il crescere dei frame disponibili fa aumentare il numero di page fault (anomalia di Belady).
La sostituzione ottimale della pagina (OPT o MIN) è una tecnica teorica per la quale è da sostituire la pagina che non sarà usata per il più lungo periodo di tempo. Questo algoritmo è praticamente impossibile da implementare, perchè richiederebbe troppe informazioni sui processi; un'ottima approssimazione di questa tecnica è l'LRU (Least Recently Used), che sostituisce la pagina che è non stata usata per il più lungo periodo di tempo, implementandolo con uno stack o con un contatore. Anche qui, però, potremmo andare a rimuovere pagine che dovranno essere usate nell'immediato.
Un aiuto può venire aggiungendo bit di riferimento, che vengono assegnati dall'hardware quando la pagina è referenziata.
L'algoritmo dei bit di riferimento addizionali viene implementato come una sequenza di bit (tipicamente un byte): il SO porrà a 1 il bit di ordine più elevato ogni volta che la pagina è referenziata, e ogni quanto di tempo i bit verranno spostati a destra (scartando quello di ordine meno elevato e ponendo a 0 quello di ordine più elevato), in modo da ottenere un "numero" che sarà tanto più alto quanto la pagina viene usata.
Un altro algoritmo è quello della seconda possibilità, che utilizza un solo bit di riferimento posto a 1 . Quando la pagina verrà designata per la sostituzione, se il bit è 1, verrà posto a 0 (dando una "seconda possibilità" alla pagina), mentre se è 0, la pagina verrà sostituita. Questa tecnica aiuta quando la scelta della pagina è sbagliata.
Un miglioramento di quest'ultimo algoritmo tiene in considerazione la coppia "bit di modifica/bit di riferimento" usandola per determinare quale pagina sostituire (nell'ordine: insost. → sost. (R,M): (1,1), (1,0), (0,1), (0,0)).
Altri algoritmi sono l'algoritmo *Least Frequently Used* che sostituisce la pagina con il conteggio minore (usata meno delle altre, quindi sostituibile) e il *Most Frequently Used* che sostituisce quella usata di più (basandosi sul fatto che pagine con conteggio basso sono state appena trasferite e quindi devono ancora essere usate).
Alcuni sistemi implementano il loro algoritmo, perchè ottengono prestazioni migliori rispetto all'uso di un algoritmo generico (es. basi di dati); per queste applicazioni viene messo a disposizione un disco grezzo (raw), che non beneficia degli algoritmi del sistema operativo, ma che deve implementare i propri.

Allocazione dei frame

Per evitare problemi con i page fault, si deve allocare un numero minimo di frame. Per allocarli, sono disponibili diversi algoritmi:, il più semplice è allocare a ogni processo lo stesso numero di frame (allocazione omogenea); questo è sconveniente, perchè non sappiamo mai se un processo sprecherà frame o non ne avrà mai abbastanza. Per risolvere, si può usare l'allocazione proporzionale, che alloca i frame a seconda della necessità del programma.
La sostituzione può essere locale o globale: quella locale sceglie il frame vittima solo fra i frame del processo, quella globale fra tutti i frame di tutti i processi. La seconda è comoda nei sistemi con priorità, perchè possono togliere frame ai processi a priorità minore per completare la propria esecuzione.

Thrashing

Il thrashing è un problema che si verifica quando una richiesta di paginazione innesca page fault in altri processi, provocando un aumento delle richieste di pagina ciclico, che termina solo con il blocco del computer. Quando ci sono processi in attesa del dispositivo di paginazione, lo scheduler vede un calo dell'attività della CPU e manda in esecuzione altri processi, aumentando le richieste di pagina ecc.
Si può cercare di limitare il thrashing usando una procedura di rimpiazzamento locale (o procedura di rimpiazzamento a priorità), che toglie frame solo al processo stesso, evitando che le tolga agli altri e vada a bloccare il sistema. Per sapere di quanti frame avrà bisogno un sistema si può usare la strategia working-set, che definisce il modello di località di esecuzione di un processo: esso è l'insieme delle zone di memoria a cui un processo accederà durante la sua esecuzione (anche se

chiama un sottoprocesso). Il modello working-set si basa su un parametro Δ che definisce la finestra del working-set, cioè l'insieme dei riferimenti esaminati. Se una pagina è in uso, sarà del WS, altrimenti sarà tolta dopo un certo tempo. La dimensione di Δ è importante: se è troppo piccola non comprende tutta la località, se è troppo grande ne comprenderà più d'una. Il SO esamina il WS per sapere quanti frame allocare al processo e quali processi sospendere per fare spazio a un WS grande.
Questo modello è efficace ma non gestisce bene il thrashing; un'altra strategia si basa sulla frequenza dei page fault: maggiori PF, portano a un aumento dei frame allocati.

File mappati in memoria

I file possono anche essere mappati in memoria: invece di usare le chiamate specifiche per il disco, si scrive il file in memoria, per poi trasferirlo su disco alla necessità.
Alcuni sistemi implementano chiamate apposite per mappare i file in memoria.

Altre considerazioni

Per evitare un alto numero di page fault si può usare la **prepaginazione**, che consiste nel caricare in memoria tutte le pagine che saranno necessarie (basandosi in particolare sul working-set di un processo); resta da chiedersi se il costo necessario alla prepaginazione sia inferiore a quello dei corrispondenti page fault.
Inoltre, c'è da considerare la **dimensione della pagina**, che deve essere scelta per avere il minor spreco di spazio e tempo: pagine minori richiedono meno spazio, ma sono compensate da page fault pià frequenti. Pagine più grandi sono soggette a meno PF, ma il tempo di caricamento è maggiore.
Altro problema è la TLB: bisogna stabilirne l'**estensione**, cioè la quantità di memoria alla quale può accedere (numero di elementi moltiplicato per la dimensione della pagina).
La **tabella delle pagine invertita**, inoltre, è un ottimo metodo per garantire protezione e risparmio di spazio.
È comodo poi poter **bloccare alcune pagine in memoria**, per favorire, ad esempio, i dispositivi di I/O ed evitare che dopo un I/O la pagina in cui il dispositivi deve scrivere non sia più disponibile. È necessario però prestare attenzione all'uso dei bit di blocco della pagina: dal momento che l'unico che può spegnerlo è chi l'ha acceso, se il processo che l'ha acceso restasse bloccato, un frame andrebbe perso.

L'interfaccia del file system

Il concetto di file

Il file è un insieme di informazioni con una specifica struttura identificato in modo univoco da un nome, conservato sulla memoria secondaria.
Ogni file ha degli attributi che possono variare da un sistema all'altro: nome, identificatore, tipo, locazione, dimensione, protezione, tempo, data e identificativo dell'utente.
Le informazioni sui file sono memorizzate nelle directory.
Un file è un tipo di dati astratto su cui possono essere effettuate delle operazioni: creazione, lettura, scrittura, riposizionamento al suo interno, cancellazione e troncamento. Il sistema operativo mantiene una tabella dei file aperti che contiene l'identificatore del file e del processo che l'ha aperto, oltre al tipo di accesso richiesto; un puntatore al file, il contatore degli accessi e la posizione del file su disco. Se un file non è in uso si dice *chiuso* e il suo descrittore è rimosso dalla tabella.
Un file può essere aperto da diversi processi, compatibilmente con i blocchi imposti dal sistema operativo: uno *shared lock* è un blocco condiviso, con cui si permette a più processi di leggere, ma non modificare il file. Un *exclusive lock* impedisce ad altri processi di aprire, anche in lettura, un file, perchè solitamente c'è un processo con il file aperto in scrittura. Alcuni SO possono imporre blocchi obbligatori o consigliati.
Ogni file ha un tipo, che è solitamente, ma non necessariamente, identificato dall'estensione. C'è la possibilità di dare un estensione a un file che ha un tipo diverso da quello specificato da essa. Nei sistemi UNIX, ad esempio, il tipo di file è identificato da un “numero magico” posto all'inizio del file.
Il tipo di file può essere usato per definirne la struttura, che può talvolta essere richiesta dal sistema operativo o dai programmi che dovranno leggerlo.

Metodi di accesso

Vi sono diversi modi per accedere a un file: l'accesso sequenziale è la lettura del file dal principio alla fine; l'accesso diretto presuppone che il file sia composto da record logici, ai quali si può far riferimento per un accesso diretto; il numero di blocco fornito all'utente è solitamente relativo (n blocchi dall'inizio). Sulla base dell'accesso diretto sono stati creati altri metodi di accesso basati su indici.

Struttura delle directory

Una directory è una struttura nella quale vengono memorizzate le informazioni sui file in essa contenuti; su una directory si possono effettuare diverse operazioni: ricerca di file, creazione/cancellazione di file, elencare il contenuto, rinominare un file, attraversare il file system.
La struttura più semplice di directory è quella a singolo livello: tutti i file stanno nella stessa directory; ciò richiede però che i nomi dei file siano univoci e crea problemi soprattutto nella condivisione delle directory. La directory a due livelli, invece, prevede che all'interno di una directory root possano essere presenti altre sottodirectory; ogni utente ha la sua *User File Directory* all'interno della *Master File Directory*; quando un utente cerca un file, questo viene cercato solo nella sua UFD. Ciò permette di isolare gli utenti, ma se questi volessero cooperare sarebbe un problema. Si può permettere a più utenti di condividere directory facendo in modo che si riferiscano a quella di altri utenti con la sintassi (ad esempio) */utente2/filecond*.
Un altro tipo di directory è quella strutturata ad albero, che si basa sulla directory a due livelli, ma consente la creazione di infinite sottodirectory. Ogni utente ha una directory corrente, nella quale sta lavorando, e può usare la chiamata di sistema *change directory* per spostarsi all'interno di un'altra. I percorsi possono essere definiti in maniera relativa (riferita alla dir corrente) o assoluta (riferiti alla root). La cancellazione dovrà seguire determinate politiche: per cancellare una directory bisognerà controllare che non sia piena o contenga altre sottodirectory, ed eventualmente chiedere all'utente come comportarsi.
Una altro tipo di directory è quella a grafo aciclico, che permette di referenziare lo stesso file/directory da più directory (permettendo così una facile condivisione) attraverso dei *link* (collegamenti) ai file. La cancellazione del file equivale alla cancellazione del link. Quando non ci

saranno più link a un file (un contatore apposito può servire), il file sarà allocato come spazio libero. Bisognerà però definire metodi specifici per la ricerca di un file nel filesystem, per evitare che lo stesso file sia trovato più volte, dal momento che percorsi diversi potranno portare allo stesso file.
L'ultimo tipo di directory è quella a grafo generale, che è identica a quella a grafo aciclico, se non per il fatto che consente la creazione di cicli fra le directory. Ciò crea una serie di problemi, tra cui la possibilità che il contatore di link al file segni valori errati a causa dei cicli; gli algoritmi di gestione saranno quindi più complessi.

Montaggio del file system

Il filesystem deve, come un file, essere aperto prima di poter essere letto; questa operazione è chiamata montaggio del filesystem. Sarà necessario associare al filesystem un punto di mount, ossia una directory nella quale visualizzare il contenuto del filesystem.

Condivisione dei file

In un sistema multiutente può presentarsi la necessità di condividere file fra più di un utente: in questi casi è necessaria una politica che specifichi quali accessi un determinato utente può acere a un determinato file. La maggior parte dei sistemi realizza ciò attraverso identificatori utente inseriti in un elenco. Poiché l'elenco può diventare parecchio lungo, in alcuni sistemi è presete anche un indicatore di gruppo, che permette di raggruppare un certo numero di utenti e concedere determinati diritti su specifici file.
La rete permette poi di creare filesystem remoti, che devono essere accessibili a determinate condizioni. Il modello più semplice è quello client-server (pensiamo al comune protocollo HTTP), che può consentire di montare un filesystem esposto da un server remoto. Per identificare l'origine della richiesta client si può usare l'indirizzo IP, che però possono essere contraffatti.
È necessario allora implementare altri metodi per identificare l'utente che cerca di avere accesso ai file. Un altro tipo di accesso remoto a un filesystem sono i file system distribuiti.
Come i filesystem locali, anceh quelli remoti si possono guastare, e con più facilità: oltre ai guasti locali ai dischi (dovuti anche al maggiore uso), possono verificarsi guasti alla rete.
Un client che abbia montato un filesystem remoto, il quale va down, può comportarsi in due modi: può chi,udere immediatamenteil collegamento, troncando il lavoro, oppure salvare una copia locale del file, da salvare sul server quando questo tornerà disponibile.

Protezione

Per proteggere i file da acessi non autorizzati ci sono diverse tecniche: innanzitutto bisogna definire i tipi di accesso consentiti. Tipicamente sono: lettura, scrittura, esecuzione, accodamento, cancellazione e elenco. Per controllare l'accesso si può usare una *Access Control List*, che però può essere noiosa e lunga da definire. Per abbreviare il compito si possono usare delle “riduzioni”: proprietario, gruppo e universo. La lista però comporta problemi sia di dimensione che di conflitto fra autorizzazioni. Per fare questo sono stati implementati altri metodi di protezione, tra cui associare password a ogni file; ciò può diventare gravoso nel momento in cui un itente deve ricordarsi password diverse per ogni file; la password può, in alcuni sistemi, essere associata a sottodirectory.

Implementazione del File System

Struttura del file system

I file devono essere organizzati in modo che l'accesso sia il più semplice possibile. Il file system deve essere strutturato in modo da garantire ciò; generalmente, un file system è diviso in livelli: il driver di dispositivo e i gestori delle interruzioni sono il livello più basso, seguito dal file system di base che deve eseguire solo comandi generici su tracce e cilindri. Il modulo di organizzazione dei file gestisce l'allocazione dei blocchi fisici in modo da ricostruire ogni file. Infine, il file system logico, la parte di livello più alto, gestisce i metadati dei file, ignorando completamente i dati. Nel FS logico è conservato il blocco di controllo del file (FCB, descrittore del file) che contiene le informazioni sul file.

Realizzazione del file system

Per realizzare un file system vengono usate differenti strutture su disco e sulla memoria: su disco, il blocco di controllo del boot contiene informazioni sull'avvio del SO, il blocco di controllo della partizione contiene i dettagli sulla partizione, le directory e gli FCB.
In memoria ci possono essere la tabella delle partizioni montate, i descrittori delle directory, la tabella dei file aperti globalmente e quella dei file aperti localmente. Per creare un nuovo file, una chiamata apposita al file system alloca un nuovo FCB.
Le partizioni del disco possono essere grezze (raw), cioè senza nessun file system, che non possono essere utilizzate se non da chi ne conosce la struttura (es. linux swap), oppure avere un file system. La partizione root contiene il kernel e i file necessari al funzionamento del SO.
I file system possono anche essere virtuali: per far coesistere file system diversi si implementa un'interfaccia virtuale (interfaccia VFS) che sta fra i diversi filesystem e l'interfaccia esposta al SO, e attraverso vnode (id univoci per ogni file fra tutti i FS in rete) permette di identificare ogni file univocamente.

Realizzazione delle directory

Il modo più semplice di rappresentare le directory è atraverso una lista di nomi file con puntatori ai blocchi di dati. La ricerca (pura, ai fini dell'inserimento per evitare duplici nomi, ai fini della cancellazione) in questa implementazione è però molto scomoda. L'unico vantaggio è la possibilità di stampare velocemente la lista dei file.
Le liste ordinate migliorano questo aspetto, ma hanno problemi di spostamento quando si fanno inserimenti. Strutture sofisticate, quali i B-alberi, sono invece molto comode.
La rappresentazione tramite una tabella di hash, invece, è più comoda, ma necessita delle routine per la gestione delle collisioni.

Metodi di allocazione

Per allocare lo spazio su disco vengono usate principalmente tre tecniche: l'allocazione contigua richiede che ogni file occupi un certo numero di blocchi su file. Le ricerche su file contigui sono semplicissime. Questo tipo di allocazione da problemi nella ricerca dello spazio libero, nonché causa frammentazione esterna (perchè usa *first-fit* e *best-fit* come l'allocazione dinamica della memoria). Per evitarlo, ancora una volta si può usare la compattazione, ma il tempo morto blocca tutto il sistema. L'altro problema è la necessità di determinare lo spazio necessario a un file. Se ne allochiamo troppo poco, avrò problemi.
L'allocazione collegata risolve i problemi di quella contigua: la directory contiene un puntatore al primo e all'ultimo blocco del file, e ogni blocco contiene un puntatore al blocco successivo. Così facendo, la compattazione non è necessaria. Un problema è lo spazio occupato dai puntatori: viene risolta raggruppando i blocchi in cluster. Oppure, si pensi a cosa potrebbe succedere se un puntatore viene perso o danneggiato. Una variazione a questo metodo è quello di usare una *File Allocation Table* che viene scritta in una partizione separata del disco.
La tabella ha un elemento per ogni blocco del disco ed è indicizzata dal numero del blocco.
L'allocazione indicizzata risolve risolve i problemi di accesso diretto spostando tutti i puntatori in un blocco indice. Se i blocchi sono troppi, si possono usare più blocchi indice (schema collegato), oppure un indice multi livello, che contiene più livelli di blocchi indice. Un altro tipo di accesso è lo

schema combinato, che mantiene in un blocco indice alcuni puntatori a blocchi diretti e alcuni a blocchi indice.
Per scegliere un metodo di allocazione bisogna tenere conto della velocità del supporto di memorizzazione e il tempo di accesso dei blocchi di dati.

Gestione dello spazio libero

Si può tenere traccia dello spazio libero in un'apposita lista dello spazio libero. Generalmente, si implementa come un vettore di bit o come una mappa di bit. Questo metodo non è efficiente se il vettore non è mantenuto in memoria centrale Un altro metodo è quello di mantenere una lista collegata; è però molto costoso se è necessario attraversare l'intera lista, dal momento che è sparsa per il disco. I blocchi liberi possono anche essere memorizzati in gruppi, con l'ultimo blocco che contiene gli indirizzi di altri blocchi liberi. Infine, si può trarre vantaggio dal fatto che i blocchi vengono tipicamente liberati a gruppi, e si tengono memorizzati l'indirizzo del primo blocco e il numero di blocchi liberi contigui.

Efficienza e prestazioni

L'uso efficiente dello spazio su disco dipende dall'algoritmo utilizzato per l'allocazione. Inoltre, se vi sono attributi come “ultimo accesso” o “ultima modifica”, ad ogni accesso o modifica al file sarà necessario scrivere su disco.
Le prestazioni del sistema possono migliorare qualora si sfrutti una cache del disco o una cache delle pagine. Alcuni sistemi usano anche una buffer cache unificata o una doppia cache.
Le scritture in cache possono essere sincrone o asincrone. Alcuni sistemi ottimizzano la cache usando algoritmi di sostituzione della pagina. Alcuni usano la tecnica *free-behind*, che rimuove una pagina dal buffer quando si richiede la successiva, altri la tecnica *read-ahead* che mette in cache la pagina richiesta e le pagine successive. Si possono anche utilizzare dischi RAM (dischi virtuali) per la cache.

Recupero del file system

Dal momento che le informazioni sulle directory sono conservate in memoria centrale, se si verificasse un crash del computer il file system potrebbe rimanere in una situazione di incoerenza. Per questo motivo, il controllore della concorrenza (o consistenza) confronta i dati nella struttura delle directory con quelli su disco per determinare eventuali corruzioni. Se ne trova, è necessario recuperare i file corrotti da un supporto di backup.
Il backup può essere effettuato in modo completo o incrementale. Il caso migliore è quello di intervallare adeguatamente il completo all'incrementale.

File system basato sulla registrazione delle attività

Questo tipo di filesystem può essere implementato come file system orientato alle transazioni basate su log o come file system con registrazione delle attività. Ciò consente, se necessario, di recuperare le modifiche effettuate senza la necessità di ricorrere ai backup.

Sottosistemi di ingresso/uscita

L'hardware di I/O

Una periferica comunica con il computer attraverso bus e porte. Se ci sono più periferiche, si può implementare una daisy chain (collegamento a cascata) che opera come i bus. I componenti che operano sulle porte sono i controller, la cui complessità varia a seconda della periferica che devono controllare.
Il controller comunica con il processore attraverso registri dedicati, oppure attraverso l'I/O mappato in memoria, in cui i registri di controllo della periferica sono mappati in memoria centrale.
Una porta di I/ è costituita solitamente da 4 registri: stato, controllo, data-in e data-out.
Il computer comunica con la periferica facendo l'handshaking: il computer controlla il bit di stato *busy* fino a quando non indica che la periferica è libera; a questo punto ordina un'operazione tramite il corrispondente bit del registro di controllo e pone il bit *command-ready* a 1; quando il controller si accorge che c'è un comando pronto, passa in stato *busy*, legge il comando e a seconda di quello che gli è stato detto di fare, legge o scrive dati; alla fine, azzera i bit *busy, command-ready* e *error*.
Questo metodo di comunicazione è detto **attesa attiva**.
Un altro metodo di comunicazione è quello che sfrutta gli interrupt: la CPU ha una linea di richiesta degli interrupt che controlla dopo ogni istruzione; quando vede che è attiva, salva lo stato del processo in esecuzione e passa il controllo alla procedura di gestione degli interrupt. Spesso, le linee di interrupt sono due: quella non mascherabile, per interrupt che non devono mai essere ignorati (accessi illegali a memoria, etc.) e una mascherabile, che può essere disabilitata se sono in esecuzione istruzioni critiche (ad esempio derivate da un interrupt non mascherabile).
La procedura di interrupt accetta un indirizzo, che corrisponderà a una entry del vettore di gestione degli interrupt in memoria centrale, il quale contiene l'indirizzo delle routine specifiche per la gestione degli interrupt.
L'interrupt chaining permette di rifurre la dimensione del vettore di interrupt: a ogni elemento del vettore corrisponde una lista di interrupt handler, che vengono chiamati uno a uno fino a trovare quello specifico per la periferica.
Agli interrupt può anche essere associata una priorità, per impedire che interrupt non critici vadano a bloccare la gestione di interrupt più importanti.
Dal momento che il tempo che la CPU impiega per gestire l'I/O da una periferica può essere lungo, si è creato il sistema del *Direct Memory Access*. Un controller DMA riceve dalla CPU le istruzioni da eseguire sulla periferica e si occupa lui dell'interfacciamento con il controller di periferica; il DMA scriverà direttamente in memoria i dati, senza costringere la CPU a farlo (rallentandola); quando il DMA trasferisce dati, blocca i bus (e quindi il collegamento CPU-Memoria centrale), ma la CPU può ancora accedere alla cache, quindi non è del tutto bloccata.

Le interfacce I/O per le applicazioni

A ciascun tipo si accede attraverso un'interfaccia specifica; le differenze fra le varie interfacce sono incapsulate nei moduli del kernel chiamati driver di dispositivo; il loro scopo è quello di nascondere le differenze dei controller di periferica all'interfaccia I/O del kernel, creando la compatibilità fra le periferiche e il sistema operativo senza aspettare che il produttore del SO adegui le chiamate alle periferiche per ogni singolo componente hardware. Le interfacce sono diverse per ogni sistema operativo, rendendo comunque necessaria la programmazione di diverse interfacce per ogni sistema operativo. Le differenze fondamentali fra le periferiche possono essere: il trasferimento dei dati (carattere o blocchi), l'accesso (sequenziale o diretto), il trasferimento (sincrono o asincrono), il tipo di periferica (condivisibile o dedicata), la velocità di elaborazione e le operazioni che la periferica può effettuare (RO, RW, WO).
Nonostante le differenze nelle chiamate dovute ai diversi SO, le periferiche sono fra loro piuttosto simili; le principali convenzioni sono sui lock, il flusso di caratteri, l'accesso a file a memoria mappata e i socket di rete.
I dispositivi con trasferimento di dati a blocchi necessitano di read(), write() e, se ad accesso diretto, seek(); le interfacce a caratteri, invece, prevedono get() e put().
Le periferiche di rete espongono dei socket che possono essere basati sull'attesa attiva (accept()) o

su chiamate asincrone (che sfruttano la select(), che restituisce informazioni sui socket entro un certo tempo specificato nella chiamata: chi è libero, chi ha spazio, chi è pieno e chi ha pacchetti in attesa); la maggior parte dei computer possiede inoltre un orologio o un temporizzatore che non è standard per tutti i SO. Le chiamate di sistema possono essere poi sincrone (bloccanti) o asincrone (non bloccanti)

Il sottosistema I/O del kernel

Il kernel fornisce diversi servizi: la schedulazione dell'I/O si occupa di garantire un accesso equo alle periferiche da parte di tutti i processi. Quando una periferica fa una richiesta di I/O, questa viene inserita nella coda di I/O e poi viene chiamato lo schedulatore per riordinarle.
Il SO mette a disposizione anche un sistema di buffer, che permette, ad esempio, di evitare lo spreco di tempo a causa di una richiesta di I/O: i dati vengono messi nel buffer, mentre il sistema continua il suo lavoro; quando il buffer è pieno, i dati vengono trasferiti. Spesso, viene usato un doppio buffer, in modo che mentre il primo trasferisce i dati, si comincia a riempire il secondo. Il sistema mette anche a disposizione delle cache, che possono essere usate per un accesso più veloce ai dati (piuttosto che prelevarli dalla memoria centrale, specialmente se quei dati sono usati di frequente).
Un altra tecnica messa a disposizione è quella dello spooling: dal momento che alcune periferiche non possono accettare più flussi di dati contemporaneamente (stampanti in primis), il SO mantiene un buffer apposito (spool) con l'elenco delle richieste, e le esegue in sequenza.
Inoltre, quasi sempre il SO mette a disposizione un sistema di protezione della memoria.
Per fare tutto ciò, il kernel deve mantenere un elenco delle informazioni di stato delle periferiche, e lo fa utilizzando strutture dati apposite.

Trasformazione dell'I/O in operazioni hardware

I sistemi operativi mappano generalmente una periferica a un determinato file su disco, e poi identificano la periferica quando si cerca di accedervi. In UNIX, ad esempio, nella tabella di montaggio sono indicati anche i percorsi associati alle periferiche.
Una generica chiamata di procedura bloccante viene verificata dal kernel (correttezza): se i dati sono disponibili, vengono forniti, altrimenti il processo viene messo nella coda di attesa I/O e si procede a contattare il driver perchè effettui l'operazione; un interrupt segnala la terminazione dell'operazione di I/O (a meno di attesa attiva) e l'operazione è completata.; il processo viene riportato in coda di pronto e potrà riprendere l'esecuzione.

Prestazioni

la gestione degli interrupt e i cambi di contesto sono piuttosto onerosi per un sistema, e anche il solo traffico di rete può generarne parecchi. In generale, ad inficiare le prestazioni sono i cambi di contesto, le copie di dati in memoria durante il passaggio dispositivo ↔ applicazione, le primitive di gestione dell'HW (che devono essere spostate in modo da poter essere eseguite in contemporanea con le operazioni di CPU e bus) e il non equilibrio fra le prestazioni della CPU, della memoria del sottosistema, dei bus e dell'I/O: se uno di essi si sbilancia, bloccherà gli altri.

Struttura delle memorie di massa

Struttura dei dischi

I dischi moderni sono grandi array monodimensionali di blocchi logici che vengono mappati nei settori del disco in modo sequenziale; così facendo possiamo teoricamente convertire un numero di blocco logico in un indirizzo fisico su disco (nella pratica no, causa settori difettati).

Schedulazione degli accessi a disco

Il tempo di accesso a disco è influenzato dal tempo di ricerca e dalla latenza di rotazione. La larghezza di banda è il numero totale di byte trasferiti diviso il tempo impiegato.
In un sistema multiprocesso, è indispensabile schedulare gli accessi a disco; la schedulazione più semplice è la FCFS, che però per determinate richieste farà schizzare la testina da una parte all'altra del disco, invece che ottimizzare le letture, come fa ad esempio la schedulazione *Shortest Seek Time First*, una schedulazione che in determinati causa la starvation di alcune richieste, perchè da la precedenza alle richieste più vicine.
Un'alternativa alla SSTF è la SCAN, che parte da un estremo del disco e va verso l'altro, servendo le richieste che incontra durante il tragitto e poi facendo la stessa cosa per il tragitto inverso. SCAN a volte è chiamato “algoritmo dell'ascensore”. Può risultare svantaggioso perchè le richieste vicino alla fines del disco sono già state idealmente servite, mentre quelle all'inizio devono aspettare che la testina torni indietro. C-SCAN è una variante di SCAN che quando arriva alla fine del disco torna immediatamente all'inizio, senza servire nulla nel tragitto. [C-]SCAN arrivano sempre all'inizio e alla fine del disco, il che può non essere necessario: LOOK e C-LOOK arrivano solo alla richiesta più vicina all'inizio o alla fine del disco, e poi invertono la marcia (LOOK) o ripartono da capo (C-LOOK)
La scelta dell'algoritmo di schedulazione dipende molto dal sistema sul quale si sta lavorando, dal carico di lavoro e dalla distribuzione dei blocchi. In generale sarebbe ottimo avere gli algoritmi implementati come moduli del kernel, in modo da poter scegliere ogni volta quale sia il migliore e sostituirlo.

Amministrazione del disco

Il primo passo da fare per configurare un disco è la sua formattazione a basso livello; così facendo si riempie il disco con una speciale struttura dati che contiene un'intestazione, una zona dati e un codice di correzione dell'errore per ogni settore. Poi, è necessario partizionare il disco in gruppi di cilindri che saranno visti come dischi separati. Infine, si effettuerà la formattazione logica, che creerà un file system sul disco.
Alcuni dischi possono contenere una speciale partizione di boot, che contiene le informazioni per il bootstrap del sistema operativo. Il BIOS caricherà il bootstrapper del sistema operativo in memoria, cominciandone poi l'esecuzione,.
Non è detto che i dischi siano privi di difetti: nel caso un disco contenga blocchi difettosi, il sistema può reagire in diversi modi: alcuni sistemi informano la FAT di blocchi difettosi durante la formattazione, in modo che il settore non venga usato ma sostituito con uno “di scorta”; anche se il check del disco è eseguito durante l'esecuzione, la scoperta di blocchi difettosi viene trattata allo stesso modo. Dischi più sofisticati, come i dischi SCSI, contengono un gruppo di settori di scorta che vengono sostituiti a quelli difettosi quando sono scoperti.
Alcuni dischi effettuano lo slittamento dei settori, in modo che i dati restino contigui: se un blocco è difettoso, i successivi vengono slittati di 1 blocco, di modo che l'ultimo vada a finire in un settore di ricambio e il settore successivo a quello guasto contenga le informazioni di quello guasto.

Gestione dello spazio di swap

L'uso dello spazio di swap è lasciato alla discrezione del sistema operativo, il quale può usarlo per tenere una copia della memoria, piuttosto che soltanto le pagine interessate.
Lo spazio di swap può essere messo in una partizione separata oppure può essere ricavato dallo spazio del normale filesystem; in quest'ultimo caso, possono sorgere problemi legati allo scorrimento della struttura delle directory e ai tempi di accesso a un file (che è a tutti gli effetti un file normale); per contro, l'allocazione di una partizione è più veloce ma richiede un

ripartizionamento per modificarne la dimensione.

La struttura RAID

Le strutture di *Redundant Array Independent/Inexpensive Disk* sono usate per l'affidabilità e la velocità di trasferimento. L'affidabilità di un disco infatti migliora (stranamente) se vi è ridondanza dei dati. Il metodo più semplice per aumentare l'affidabilità è il mirroring (o shadowing) dei dati: un disco contiene la copia esatta dell'altro, consentendo di recuperare i dati da un disco quando vengono persi sull'altro (per guasti o corruzione).
Per aumentare le prestazioni, invece, si può usare il parallelismo: un disco (ma funziona anche con – ad esempio – otto dischi a un bit per disco) contiene la metà dei dati (mezzo byte, per intenderci) e l'altro contiene l'altra metà: dovremo leggere contemporaneamente da due dischi la metà dei dati, aumentando la velocità dei trasferimenti.
Il RAID è organizzato in diversi livelli: il livello 0 è lo striping non ridondante, l'1 è il mirroring. Il RAID 2 utilizza lo striping con gli ECC per ridurre il numero di dischi necessari allo striping: se viene persa una porzione dei dati, si usano gli ECC per recuperarla. Questo schema non è usato, perchè è più costoso dell'altrettanto efficiente RAID 3, dell'organizzazione di parità a bit alternati, che usa un disco per memorizzare i bit di parità degli altri: dal momento che possiamo sapere esattamente qual'è il settore danneggiato, riusciamo a calcolare la sequenza corretta basandoci sulla parità dei bit negli altri dischi.
Il RAID 4 (organizzazione di parità a blocchi alternati) mantiene i dati organizzati in blocchi sui singoli dischi, più un disco di parità che può essere utilizzato per il recupero se un disco viene a mancare.
Il RAID 5 (parità distribuita a blocchi alternati) suddivide i blocchi di parità fra tutti i dischi, anziché su un unico disco, mentre i dati sono distribuiti fra i restanti dischi, per evitare un sovraccarico del singolo disco di parità,
Il RAID 6 (schema di ridondanza P+Q), infine, utilizza, al posto della parità, dei codici di correzione dell'errore, garantendo la possibilità del fallimento di due dischi anziché uno solo.
Sono inoltre disponiili il RAID 0+1 e 1+0, che sono mix dei RAID 0 e 1 per garantire sia l'affidabilità che le prestazioni. Nel primo, dischi striped sono messi in mirror, nel secondo dischi in mirror sono divisi in stripe.
La scelta del livello RAID dipende, ancora una volta, dall'ambiente di lavoro necessario: se è necessaria velocità, si sceglie l'1, per l'affidabilità lo 0. L'1+0 e 0+1 non sono molto usati, a casuua del costo più elevato rispetto al 5.
Spesso le strutture RAID contengono un disco di ricambio, che viene usato come spazio di archiviazione dei dati recuperati dopo un fallimento.

Collegamento dei dischi

I computer accedono ai dischi tramite le porte fisiche o tramite la rete. La memorizzazione con collegamento all'host presuppone l'accesso fisico all'host. Le implementazioni variano dalle strutture desktop, che tipicamente supportano non più di due dischi per ogni bus I/O alle complesse architetture server basate su SCSI e Fiber Channel, che ne supportano molte di più (anche per via delle necessità di RAID). Ad un host possono essere collegati una enorme varietà di supporti di memorizzazione, dai dischi, ai RAID, ai nastri, ai CD/DVD. I comandi per le R/W vengono dati direttamente sui bus.
La memorizzazione con collegamento di rete, invece, è strutturata per l'accesso remoto ai dati. I client accedono tipicamente ai dischi con RPC trasportate dalla rete. Questo può portare, in grandi sistemi di memorizzazione, a un sovraccarico della rete; per risolvere questo problema si usano delle *Storage Area Network*, reti provate in cui server e dispositivi di memorizzazione comunicano con protocolli di memorizzazione invece che con protocolli di rete.

Implementazione dell'archiviazione stabile

Le informazioni che risiedono in un'unità di memorizzazione stabile non vanno mai perse. Per fare ciò, dobbiamo avere le informazioni replicate su più dispositivi con modalità di guasto indipendenti. Per fare ciò, è necessario che ogni volta che è rilevato un guasto sia invocata una procedura per la sua correzione, e bisogna coordinare la scrittura degli aggiornamenti in modo che un guasto durante

gli aggiornamenti non lasci tutte le copie in uno stato danneggiato.
Se durante il recupero da un guasto le copie hanno lo stesso valore, e non sono rilevabili errori, il valore è corretto; se viene individuato un errore, si sostituisce il valore con quello delle copie corrette; se tutte le copie presentano errori, si sceglie una delle copie e si usa quella come valore corretto.

Struttura di memorizzazione terziaria

Le periferiche di archiviazione terziaria possono essere dischi magneto-ottici, un disco magnetico protetto da uno strato di plastica che viene scritto con tecniche simili a quelle dei dischi ottici, per proteggerlo dalle cadute della testina; per leggere i dati non si può rilevare il campo magnetico a causa della distanza dal disco: si usa allora un effetto della luce (effetto Kerr) che fa cambiare la luce a seconda del magnetismo.
Ci sono poi i comuni dischi ottici, tra cui i dischi a cambiamento di fase (RW) e i dischi a sola lettura.
I nastri sono un altro tipo di supporto rimovibile, che sfruttano un accesso sequenziale ai dati e che, a causa del costo bassissimo, vengono usati come supporti di backup.
I compiti del sistema operativo per quanto riguarda queste strutture di memorizzazione sono quelli di fornire un'interfaccia per il loro utilizzo: i dischi e i nastri hanno metodi completamente diversi di interazione con il sistema operativo.
Per i supporti di memorizzazione terziaria è necessario anche sapere come nominare i file, per garantire la compatibilità fra diversi sistemi, ma è necessario ancha garantire la facilità di ricerca di un file.
Anche la memorizzazione terziaria presenta problemi di velocità, affidabilità e costi: un disco è molto più veloce di un nastro, ma se dobbiamo cambiare il disco per trovare i dati, anche un disco veloce impiegherà più tempo del nastro: dovrà fermarsi, essere scollegato, cambiato, quello giusto dovrà essere preso, collegato, acceso e il blocco cercato.
I dischi sono anche generalmente più affidabili dei supporti ottici e dei nastri, che vengono esposti alle condizioni ambientali, ma la caduta della testina distrugge il disco, la rottura del nastro, male che vada, fa perdere qualche KB.
Il costo dei nastri è poi decisamente inferiore rispetto a quello dei dischi o degli altri supporti.

Strutture dei sistemi distribuiti

Concetti di base

un sistema distribuito è un insieme di sistemi di elaborazione collegati da una rete di comunicazione. Ogni workstation può differire dalle altre. I vantaggi principali sono dovuti alla condivisione delle risorse, all'aumento della velocità di calcolo e dell'affidabilità, nonché della possibilità di comunicare, chiamando procedure non disponibili localmente ma solo in altri sistemi.
Esistono diversi tipi di sistemi operativi distribuiti: i SO di rete, che condividono un ambiente multiutente fra locazioni diverse, i sistemi di login remoto, che consentono di utilizzare sistemi posti a grandi distanze , sistemi per il trasferimento di file a distanza (ftp) e sistemi operativi distribuiti, nei quali gli utenti accedono alle risorse remote come se fossero locali, sia in termini di dati che in termini di distribuzione della computazione, che in termini di migrazione dei processi.
Quest'ultimo, in particolare, permette il bilanciamento del carico, l'aumento della velocità di calcolo, la preferibilità dell'hardware (ovvero la possibilità di usare hardware dedicato non disponibile sulla macchina locale) e del software (idem, ma per il software) che l'accesso ai dati. La migrazione della computazione e dei processi, può avvenire senza l'intervento dell'utente (in modo trasparente), sia con l'intervento dell'utente (tramite chiamate specifiche).

Topologia

I siti possono essere connessi in diversi modi, tutti con vantaggi e svantaggi da valutare a seconda del caso: sono da tenere in considerazione il costo dell'installazione, della comunicazione e la disponibilità.
In una rete parzialmente connessa, ci sono risorse disponibili direttamente, mentre altre necessitano di instradamento. Se un collegamento viene a mancare, i messaggi vengono persi e devono essere reinstradati.

Comunicazione

Il progettista di una rete di comunicazione deve tener conto di cinque punti fondamentali: l'attribuzione e risoluzione dei nomi, le strategie di instradamento, le strategie di pacchetto, le strategie di connessione e la gestione dei conflitti. La prima è risolta, generalmente, utilizzando una coppia <nome host, identificatore>, in cui il nome host è univoco e deve essere risolto (tradotto) in un identificatore numerico da un name server.
Le strategie di instradamento sono tre l'instradamento fisso fornisce un percorso unico per raggiungere un host: è comodo perchè veloce, ma se il collegamento viene a mancare l'host non sarà più raggiungibile; l'instradamento virtuale fissa un percorso per tutta la durata di una sessione; il cambio della sessione potrebbe portare un percorso diverso (ad esempio se fra una sessione e l'altra un nodo è caduto); l'instradamento dinamico instrada indipendentemente ogni pacchetto, e quindi se un nodo cade richiede soltanto di rispedire quel pacchetto, senza perdita di dati. Generalmente, i nodi hanno un instradamento statico verso un gateway che poi instrada dinamicamente i pacchetti verso un gost; uno o più router (instradatori) sono le entità all'interno della rete che si occupano di instradare i pacchetti usando un protocollo ben definito.
Le strategie di pacchetto si implementa comunemente con lo scambio di pacchetti di dimensione fissa che possono essere spediti in diversi modi: con o senza una connessione e in modo affidabile o inaffidabile (pacchetto spedito e ignoro se è stato ricevuto).
Le strategie di connessione sono quelle che vengono messe in atto per stabilire come le coppie di processo che vogliono comunicare devono parlarsi: commutazione di circuito (tutta la sessione), di messaggi (messaggi singoli) o di pacchetto (parti di messaggio, richiede un identificatore del numero di pacchetto per ricreare l'ordine per ricostruire il messaggio).
A seconda del tipo di rete, è necessario anche stabilire come gestire i conflitti: la tecnica CSMA/CD, quando rileva un conflitto, aspetta un tempo casuale prima di rispedire il pacchetto, mentre in una rete a token solo chi ha il gettone può spedire.

Protocolli di comunicazione

i computer di una rete devono concordare un protocollo o un gruppo di protocolli con cui comunicare: l'ISO/OSI è composto da sette strati: fisico (connessione elettrica), collegamento dei

dati (gestione dei frame), rete (instradamento attraverso i protocolli), trasporto (accessi a basso livello della rete e trasferimento messaggi), sessione (implementazione delle sessioni), presentazione (far coesistere reti diverse), applicazione (interazione con l'utente). Di questi sette, il protocollo TCP/IP ne usa solo 4.
Il protocollo TCP è un protocollo orientato alle connessioni ed è affidabile, mentre UDP è inaffidabile e orientato alle comunicazioni senza connessione.

Robustezza

Un sistema distribuito può soffrire di guasti hardware maggiori di un sistema non distribuito: pertanto è necessario individuare i guasti, riconfigurare il sistema e recuperare dai guasti. Per individuarli, la tecnica migliore è quella dell'handshaking con tempo massimo di attesa; se scopriamo un guasto alla rete,dovremo riconfigurare il sistema affinchè, ad esempio, instradi diversamente il traffico e avvisi tutti gli altri siti che un sistema è guasto. A quel punto, un sistema che dopo un guasto ritorna online, dovrà recuperare informando gli altri siti di essere tornato disponibile.

Problemi progettuali

In un sistema distribuito, la sfida maggiore è rendere l'utente virtualmente inconsapevole della distribuzione del sistema, in modo che possa collegarsi al sistema da qualsiasi località.
Un altro problema è quello di garantire la tolleranza ai guasti, facendo in modo che il sistema possa continuare a funzionare anche se su scala ridotta.
Inoltre, ogni sistema deve essere scalabile, cioè deve potersi adattare alle condizioni di carico dell'intero sistema distribuito, per non sovraccaricare il tutto. Questi ultimi due problemi sono collegati: un sistema sovraccarico si comporta come un sistema guasto, sebbene nella realtà non lo sia; sarebbe opportuno riuscire a distribuire il carico su quel sistema, per farlo tornare a regime.
Sistemi di questo tipo non devono in alcun modo essere centralizzati, perchè un guasto al sistema centrale guasterebbe l'intero sistema. L'approssimazione migliore di un sistema in cui ogni sito ha uguale peso nella rete è il clustering.

File system distribuiti

L'ambiente distribuito

un sistema distribuito è un insieme di computer lascamente collegati tramite una rete di comunicazione; un servizio è un'applicazione eseguita su un server; il client accede al servizio.
Un DFS è un file system in cui client, server e dispositivi di memorizzazione sono sparsi in una rete; idealmente, esso dovrebbe apparire agli utenti come un unico file system ed avere prestazioni accettabili nel recupero dei file.

Attribuzione dei nomi e trasparenza

in un DFS trasparente, è opportuno nascondere la reale locazione dei file sulla rete, alla quale non si deve poter risalire tramite il nome del file (trasparenza della locazione). È opportuno inoltre garantire l'indipendenza della locazione: non importa dove un file sia, il suo nome non deve cambiare; nei sistemi attuali è praticamente impossibile implementare quest'ultima.
Se nome e locazione sono indipendenti, potremmo addirittura utilizzare client senza dischi che, con procedure speciali, all'avvio recuperano dalla rete il kernel del sistema operativo.
Er attribuire i nomi di file in un DFS si usano principalmente tre schemi: il più semplice e una combinazione nome macchina+nome file; il secondo consiste nel collegare directory remote a directory locali con un particolare tipo di montaggio. Il terzo metodo consiste nel far generare l'albero dei file a una singola struttura globale dei nomi, rendendo la struttura del tutto identica a quella di un file system locale.
Per implementare ciò si possono usare degli identificatori di file indipendenti dalla locazione, che verranno mappati all'unità che li contiene.

Accesso remoto ai file

Quando è stato localizzato il server contenente un file, è necessario richiedere il file; per farlo, sipuò usare un meccanismo di servizio remoto, che fa una richiesta al server, il quale identifica l'utente e restituisce (se autorizzato) il file.
In questi casi, torna comodo l'uso di una cache per memorizzare un file che è stato richiesto, in modo da non doverlo ritrasferire se servisse nuovamente; il problema principale di questo metodo consiste nella coerenza della cache.
È da stabilire inoltre dove memorizzare la cache: su disco sarebbe ottima per l'affidabilità, ma se il file fosse mantenuto in memoria centrale le workstation potrebbero funzionare senza dischi e sarebbe molto più rapida.
In caso di modifica di un file remoto, inoltre, è da stabilire come questo dovrà essere aggiornato sul server: le politiche sono principalmente due: la politica di scrittura immediata, che però causa un sovraccarico di rete, e la politica di scrittura ritardata, che però porta problemi di affidabilità quando il computer di guasta. Una politica intermedia (la scansione e aggiornamento della cache a intervalli regolari) è buona.
Come già detto il problema principale della cache in un DFS è la coerenza: un client deve sempre sapere se la propria copia dei dati è aggiornata rispetto a quella principale. I metodi per verificarlo sono due: la verifica iniziata dal client, che contatta il server per verificare la coerenza; l'intervallo di tempo fra una verifica e l'altra è cruciale, perchè se troppo frequente rallenterà il sistema, ma se lo fosse troppo poco la copia principale potrebbe venire modificata; la verifica iniziata dal server, invece, è una politica per la quale il server che rileva una potenziale incoerenza nei dati inizia la verifica basandosi su una tabella che contiene i client che hanno aperto il file; File server con e senza stato
Un file server con stato richiede che il client, prima di qualsiasi operazione, faccia una richiesta open() al server e che faccia una close() alla fine delle sue operazioni; questo implica però la necessità, in caso di guasto, di ristabilire tutte le connessioni per evitare problemi di incoerenza dei dati, e risulta gravoso per il server a causa del necessario dialogo ocn i client. Un file server senza stato, invece, permette il passaggio diretto dei file, senza operazioni di apertura o chiusura e, in caso di guasto, la sua resurrezione lo mette immediatamente a disposizione dei client, semnza problemi di ristabilimento dello stato.

Gli svantaggi di un file system senza stato sono la maggiore lunghezza dei messaggi di richiesta e la necessità che questi siano standardizzati, in modo da poterli sempre identificare. Inoltre, quando i client ritrasmettono le operazioni sui file, queste devono essere sempre rifatte e garantite idempotenti.

Replica dei file

La disponibilità di più copie di un file su diversi server implica che un client possa richiedere un file e questo gli venga fornito dal server più vicino, con conseguente aumento delle prestazioni. Il requisito base per la replica è l'indipendenza dei server rispetto ai guasti. È opportuno inoltre nascondere all'utente la replicazione e lasciare che sia lo schema di attribuzione dei nomi a scegliere. Il problema principale delle repliche è la loro coerenza, cioè la necessità di aggiornare tutte le repliche quando una viene modificata: in determnati casi è necessario sacrificare la coerenza a favore dell'affidabilità e delle prestazioni (pensiamo a un partizionamento della rete) ma sarebbe meglio evitarlo sempre.

Coordinamento distribuito

Ordinamento degli eventi

In un sistema centralizzato si può sempre determinare l'ordine in cui due operazioni sono state eseguite.
La relazione *accaduto-prima* implica che un evento A sia avvenuto prima di un evento B. Se la relazione non si applica, allora due eventi sono avvenuti in modo concorrente.
Per implementare questa relazione è necessario un orologio o un numero di orologi perfettamente sincronizzati che permettano di assegnare un tempo a ogni evento e ricostruire la relazione →.
In un sistema distribuito questo non è possibile, perciò si tende ad associare un timestamp ad ogni operazione, in modo da stabilire un ordinamento globale. Quando un sistema riceverà un messaggio con TS > del suo orologio, aggiornerà il suo orologio in modo da riflettere il TS.

Mutua esclusione

Per implementare la mutex in un sistema distribuito, si possono usare diversi metodi: quello centralizzato consiste nell'avere un coordinatore che controlli la mutua esclusione. Il metodo completamente centralizzato, invece, si basa sull'invio di un messaggio a tutti i nodi quando un nodo vuole entrare in sezione critica. Un nodo può dare l'OK immediatamente oppure ritardare (perchè c'è lui in sezione critica oppure vuole entrarci; in questo secondo caso confronta il proprio TS con quello dell'altro nodo e decide chi l'ha chiesto prima); un ultimo metodo è quello del passaggio di token: solo chi ha il token può entrare in sezione critica.

Atomicità

per garantire l'atomicità in un sistema distribuito è necessario un coordinatore della transazione che effettui un protocollo di commit a due fasi: nella prima il controllore invia un messaggio “prepare” a tutti i nodi; quando tutti i nodi hanno risposto “ready” entro un tempo prefissato, il controllore passa alla seconda fase, altrimenti se qualcuno ha risposto “not ready” o non ha risposto, si farà un “global abort”. Nella seconda fase il controllore scrive sul proprio log la decisione che ha preso (“global commit” o “global abort”) e poi la invia a tutti. I gestori della transazione scriveranno la decisione sul log, effettueranno la decisione presa e poi spediranno un ACK al controllore; se trascorso un certo tempo al controllore mancheranno delle risposte, rispedirà ai ritardatari la decisione, in attesa che concludano.
Se un nodo è guasto, nel momento in cui tornerà a funzionare sarà esaminato il suo log e sarà deciso : se l'ultima entry è commit(T) → redo(T); se l'ultima entry è abort(T) → undo(T); se l'ultima operazione è la ready(T) si chiede al controllore.
Il guasto del coordinatore, invece, viene gestito solo se il controllore è già nella seconda fase; se era ancora nella prima, farà un “global abort”; se è nella seconda, se l'ultima entry è il “global commit” o il “global abort”, si trasmetterà la decisione ai nodi che potrebbero non averla ricevuta.
Il guasto della rete, invece, dipenderà dal tipo di partizionamento: se tutti i nodi rimangono in una partizione, la transazione continua, altrimenti “global abort” (scade timeout) e quando la rete sarà ripristinata lo si dirà anche al ritardatario.

Controllo della concorrenza

Se i dati non sono replicati, si useranno i normali lock per un sistema non distribuito; in caso contrario, si potranno usare un coordinatore singolo che stabilirà se c'è la possibilità di ottenere un lock su una risorsa; questo metodo però è rischioso, perchè oltre a essere un collo di bottiglia per le richieste, se il coordinatore cadesse sarebbero guai.
Il protocollo di lock a maggioranza prevede invece che ogni sito mantenga un gestore locale dei lock, il quale per chiedere un lock dovrà ottenere l'autorizzazione di più di metà dei siti che contengono una replica del file. Questo metodo è più difficile da implementare e richiede la modifica degli algoritmi locali di gestione degli stalli.
Una variante del protocollo di lock a maggioranza è il protocollo polarizzato, che chiede il lock condiviso solo a uno dei siti che detiene una replica, ma chiede il lock esclusivo a tutti.
Un ultimo metodo è la copia primaria, che mantiene una replica “primaria” in un “sito primario”; il lock andrà richiesto al sito primario.

Per il controllo della concorrenza possono essere usate anche delle marche di tempo; la generazine delle marche può essere affidata a un controllore centralizzato, oppure essere calcolata come TS locale concatenato all'ID sito; ci sono comunque problemi se un sito ha un timer più veloce o più lento degli altri; pertanto, quando un sito riceve una richiesta con un TS maggiore di quello locale, aggiornerà il contatore locale a quel TS.

Gestione delle situazioni di stallo

Gli algoritmi di prevenzione degli stalli usati nei sistemi non distribuiti possono essere applicati, con le opportune modifiche, anche ai sistemi distribuiti. Gli stalli possono essere prevenuti, ad esempio, definendo un ordinamento globale delle risorse in un sito e stabilendo che un sito possa richiedere solo risorse con identificativi minori di quello minimo che detiene. Dal momento che l'algoritmo del banchiere genera molti messaggi, non è utile in un sistema distribuito.
Il sistema che è comodo usare è un sistema di prevenzione degli stalli basato sull'ordinamento delle marche di tempo con rilascio anticipato delle risorse.
Lo schema *wait-die* non prevede il rilascio anticipato, ma consente a un processo di attendere una risorsa solo se chi la detiene ha un TS maggiore del suo; in caso contrario fa rollback;
lo schema *wound-wait*, invece, prevede il rilascio anticipato delle risorse: se il processo richiedente è più vecchio di chi detiene le risorse, il detentore farà rollback.
Questi schemi evitano la starvation a patto che non venga associata una nuova marca di tempo a un processo che fa rollback, o continuerà a venire accoppato.
C'è anche la possibilità di rilevare gli stalli prima che si verifichino:i metodi che lo permettono si basano su un grafo locale di attesa: se il grafo ha cicli, c'è uno stallo, ma l'assenza di cicli, in questo caso, non garantisce l'assenza di deadlock: pià siti potrebbero richiedere risorse portando a uno stallo. Per verificarlo, è necessario organizzare un grafo di attesa di tutto il sistema distribuito: la sua costruzione può essere affidata a un unico sito (metodo centralizzato). La costruzione del grafo globale tramite l'invio di messaggi al coordinatore può provocare però la rilevazione di falsi stalli; per evitare ciò, deve essere il controllore a invocare il controllo degli stalli e deve costruire un grafo con un nodo per ogni processo del sistema, inserendo un arco solo se esiste in uno dei grafi locali o se compare in più grafi con l'etichetta TS.
Nell'algoritmo completamente distribuito per il rilevamento degli stalli, ogni sito è responsabile della rilevazione degli stalli: ognuno creerà un grafo parziale basato sul comportamento del sistema; i grafi locali conterranno un nodo P(ex) se una risorsa è richiesta da un processo esterno; un ciclo in uno di questi grafi indica uno stallo.

Algoritmi di elezione del coordinatore

Gli algoritmi di elezione del coordinatore prevedono che ogni sito abbia un identificatore univoco. Se un nodo suppone un malfunzionamento del controllore, e il sistema utilizza l'algoritmo del bullo, il sito manderà un messaggio a tutti i nodi eleggendosi a coordinatore e attenderà una risposta per un certo tempo T. Solo nodi con numero di priorità più elevato si potranno sostituire a lui; se riceve risposta da un nodo con un numero più elevato, lascerà il posto; altrimenti, scaduto il tempo, lui diventa il coordinatore.
L'algoritmo dell'anello, invece, presuppone che i processi siano disposti ad anello. Quando un processo rileva un malfunzionamento del controllore, genera una nuova lista attiva vuota, invia un messaggio elect(i) [i=mio ID] al nodo alla sua destra e aggiunge i alla lista. Se il processo riceve un messaggio elect(j) da sinistra può:

1) se è il primo elect che riceve o manda, crea una nuova lista attiva contenente i e j e poi spedisce a destra elect(i) e elect(j)
2) Se i != j, aggiunge j alla lista e inoltra
3) Se i=j, allora la lista è completa e so chi ha il numero più alto. Lui diventa il coordinatore.

Raggiungimento dell'accordo

Perchè un sistema funzioni correttamente è necessario un meccanismo di accordo su un valore comune fra diversi processi.
Questo problema è noto come “problema dei generali bizantini” e riguarda il problema delle comunicazioni inaffidabili (cioè dei guasti alla rete o della perdita di messaggi), che può essere

risolta stabilendo un tempo massimo di attesa, superato il quale sarà necessario ripetere il tentativo di accordo. La risposta del secondo sistema potrà essere “pronto” o “non funzionante”; solo se tutti e due i sistemi riceveranno “pronto” come risposta al loro messaggio di richiesta, sarà garantita l'affidabilità del mezzo di comunicazione.
Per evitare invece di incappare in processi difettosi, si può stabilire che ogni processo debba avere un proprio valore privato che sarà restituito in caso esso non sia difettoso; ogni processo mantiene una lista dei valori privati ricevuti e se, confrontandola con quella di un altro processo, scoprirà differenze, saprà quale processo è difettoso.