



Architetture degli Elaboratori e Sistemi Operativi

Master di II livello

Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

Docente

Prof. **Daniele Frigioni**

Dip. di Ingegneria Elettrica e dell'Informazione

Università degli studi dell'Aquila

Tel: 0862 - 434448

E-mail: frigioni@ing.univaq.it

URL: <http://www.diel.univaq.it/frigioni>



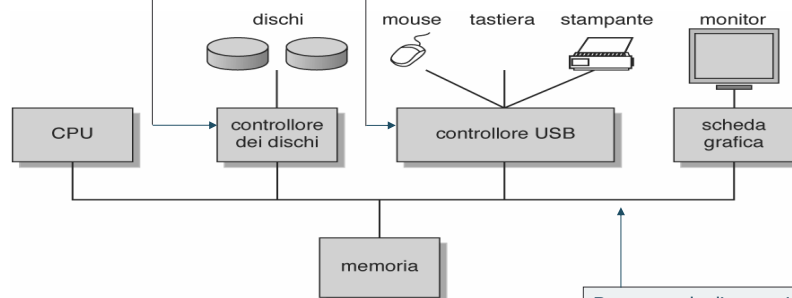
Sistemi Operativi

- Generalità
- Definizione di SO
- Struttura a strati di un SO

Moderno calcolatore elettronico



Controllori – si occupano del controllo del funzionamento dei dispositivi fisici



Bus – canale di comunicazione comune per l'accesso alla memoria condivisa



Funzionamento di un computer



- **Avvio tramite programma di avviamento (*bootstrap*)**
 - Risiede in una memoria ROM
 - Inizializza i vari componenti del sistema (registri della CPU, controllori dei dispositivi, contenuto della RAM) all'accensione del computer
 - Carica in RAM il kernel del SO e ne avvia l'esecuzione
- Il SO avvia l'esecuzione del primo **processo** (*init*) e si pone in attesa del verificarsi di qualche evento
- Un evento viene in generale segnalato attraverso una **interruzione** hardware (segnale inviato da un dispositivo fisico tramite il bus) o software (segnale di **eccezione** o **system call** causato da un programma), che interrompe l'attuale sequenza di esecuzione della CPU



Gestione delle interruzioni (1)



- I moderni sistemi operativi sono **interrupt driven** (guidati dalle interruzioni)
- Esistono molti eventi causa di interruzioni:
 - Completamento di una operazione di I/O
 - Accesso non valido alla memoria
 - Divisione per zero
 - Richiesta di un servizio del SO
 -
- A ciascun evento è associata una particolare **routine** (procedura) di gestione di quella interruzione



Gestione delle interruzioni (2)



- Quando riceve una interruzione la CPU:
 - Interrompe l'elaborazione corrente
 - Salva il contenuto dei registri e del program counter (stato corrente del sistema)
 - Trasferisce il controllo alla procedura di servizio di quella interruzione
- Il SO determina il tipo di interruzione attraverso l'uso del **vettore delle interruzioni**, che contiene gli indirizzi di memoria di tutte le procedure di servizio (il numero dei possibili segnali di interruzione è finito e predefinito!)



Interruzioni di I/O

(1)



- Ad esempio nell'esecuzione di un I/O, i dispositivi di I/O e la CPU operano come segue:
 - Ogni dispositivo ha un *controllore*
 - Ogni controllore ha un *buffer locale*
 - La CPU muove dati da memoria centrale ai buffer locali e viceversa
 - L'operazione di I/O avviene dal dispositivo al buffer locale del controllore
 - Il controllore del dispositivo informa la CPU che ha finito la sua operazione causando una **interruzione**



Interruzioni di I/O

(2)



- Se il SO riceve una richiesta di I/O da parte di un programma utente i dispositivi di I/O e la CPU operano come segue:
 - Ogni controllore di un dispositivo ha un *buffer locale* di memoria
 - Per ogni dispositivo di I/O il SO ha un *driver del dispositivo* che si coordina con il controllore e funge da interfaccia con il resto del sistema
 - La CPU muove i dati interessati alla operazione di I/O dalla memoria centrale al buffer locale del controllore del dispositivo e viceversa
 - Per iniziare l'operazione di I/O il driver del dispositivo carica i registri opportuni all'interno del controllore del dispositivo per segnalare l'operazione da compiere
 - Il controllore esamina il contenuto di questi registri per determinare l'azione da compiere
 - L'operazione di I/O avviene dal dispositivo al buffer locale del controllore e viceversa
 - Il controllore del dispositivo informa la CPU che ha finito la sua operazione causando una **interruzione**
 - Il driver passa quindi di nuovo il controllo al SO



Definizione di SO



- Non esiste una definizione universale di SO
- In generale un SO è un insieme di programmi che agisce come intermediario tra utenti e hardware di un computer. Il suo scopo è di mettere a disposizione un ambiente in cui gli utenti possano eseguire i propri programmi in maniera comoda ed efficiente
- Un SO può essere considerato da due punti di vista diversi:
 - **Punto di vista utente:** il SO è l'interfaccia attraverso la quale l'utente percepisce il sistema di calcolo
 - **Punto di vista del sistema:** il SO è il gestore delle risorse fisiche del sistema di calcolo (CPU, memoria, dispositivi di I/O, ecc.)



Punto di vista utente



- La percezione di un calcolatore da parte di un utente dipende principalmente dall'**interfaccia** utilizzata
- La maggior parte degli utenti usa PC
 - I PC sono progettati per essere usati da un singolo utente che usa le risorse del PC stesso in maniera esclusiva
 - In questi casi il SO viene progettato considerando principalmente la **facilità d'uso**
- Altri utenti usano terminali connessi a calcolatori di cui condividono le risorse con altri utenti
 - In questi casi il SO viene progettato cercando di **massimizzare l'uso delle risorse**



Punto di vista del sistema



- Due diverse definizioni possibili
 - **Assegnatore di risorse** – gestisce e assegna risorse (hardware e software) condivise rispetto al tempo (CPU) e rispetto allo spazio (memoria)
 - **Programma di controllo** – controlla l'esecuzione dei programmi utente e le operazioni dei dispositivi di I/O garantendo che non vengano commessi errori



Cosa fa parte di un SO ?



- In generale non esiste una definizione accettata universalmente di cosa faccia parte di un SO
- Possibili definizioni
 - Tutto quello che il rivenditore fornisce quando gli si richiede un SO
 - Il solo programma che è sempre in esecuzione nel computer, generalmente chiamato **kernel**, tutti gli altri sono programmi applicativi (questa è la definizione seguita nel corso)



Compiti di un SO



- In generale, è più facile definire un SO rispetto a ciò che *fa* piuttosto che a ciò che esso *è*
 - Interfacciare utente e sistema di calcolo
 - Rendere facile la soluzione dei problemi utente
 - Rendere conveniente l'uso del computer
 - Gestire le risorse hardware in maniera efficiente
 - Eseguire programmi
- Queste esigenze sono spesso contrastanti tra loro, ed è per questo che il progetto di un SO è un compito molto complesso e chi se ne occupa deve affrontare molti compromessi in fase di progetto e di realizzazione



Obiettivi di un SO



- **Semplicità:** consente uno sviluppo del software semplice mascherando le peculiarità della piattaforma hardware
- **Efficienza:** ottimizza l'uso delle risorse da parte dei programmi applicativi
- **Flessibilità:** garantisce la trasparenza verso le applicazioni di modifiche hardware e quindi la portabilità del software



Sistemi operativi time sharing



- SO che gestiscono le attività di un sistema di calcolo dotato di un unico elaboratore utilizzato da più utenti che eseguono contemporaneamente i propri programmi o *jobs* (**multi programmazione**)
- La CPU esegue molti jobs presenti in memoria centrale e su disco (la CPU è allocata ad un job solo se il job è in memoria)
- Il SO assegna la CPU a un job per un *quanto* di tempo prestabilito (**time sharing**) al termine del quale la CPU passa al job successivo salvando lo stato del job interrotto. Il SO passa rapidamente da un job all'altro e ogni utente ha l'impressione di un uso esclusivo del computer
- Ogni utente può interagire con il proprio programma durante la sua esecuzione (*sistemi interattivi*)



Caratteristiche del time sharing



- **Multiprogrammazione**
- **Scheduling** della CPU
- **Swapping** dei job da memoria centrale a disco
- **Memoria virtuale**
 - Dimensione dei programmi può essere maggiore della memoria fisica
 - Swapping tra memoria centrale e disco trasparente agli utenti
 - Separazione tra memoria fisica e memoria logica
 - Gestione del **file system**



Componenti di un SO



- Un SO è un sistema grande e complesso e si può realizzare solo se lo si scompone in piccole parti correlate tra loro, e in particolare:
 - Gestione dei **processi**
 - Gestione della **memoria centrale**
 - Gestione dei **file**
 - Gestione del **sistema di I/O**
 - Gestione della **memoria secondaria**
 - Gestione delle **reti**
 - Sistema di **protezione**
 - **Interprete** dei comandi



Gestione dei processi

(1)



- Un **processo** è un programma in esecuzione
- Un processo è una entità *attiva*, mentre un programma è una entità *passiva*
- Un processo ha bisogno di risorse per eseguire il proprio compito: tempo di CPU, memoria, files, dispositivi di I/O
- Le risorse sono fornite ad un processo quando viene creato o assegnate ad esso in fase di esecuzione
- Il processo è l'unità di lavoro elementare di un sistema di calcolo
- Un sistema di calcolo esegue processi che si dividono in processi **di sistema** e processi **utente**
- Tutti questi processi possono essere eseguiti in maniera **concorrente**



Gestione dei processi

(2)



- Il SO è responsabile delle seguenti attività in relazione alla gestione dei processi:
 - Creazione e cancellazione
 - Sospensione e ripristino
 - Fornire meccanismi per la sincronizzazione dei processi,
 - Fornire meccanismi per la comunicazione tra processi
 - Fornire meccanismi per la gestione di problematiche di *deadlock e starvation*



Gestione della memoria centrale



- La memoria centrale (RAM) è un insieme di *parole*, ognuna delle quali è caratterizzata dal proprio *indirizzo*
- È un deposito di dati velocemente accessibile ed è condivisa dalla CPU e dai dispositivi di I/O
- La memoria centrale è *veloce, piccola e volatile* ed è il passaggio obbligato di qualsiasi elaborazione
- I SO più sofisticati consentono di tenere più programmi in memoria contemporaneamente
- Il SO è responsabile delle seguenti attività relative alla gestione della memoria centrale:
 - Tenere traccia di quali parti della memoria sono usate e da chi
 - Decidere quali processi caricare in memoria quando c'è spazio disponibile
 - Assegnare e revocare lo spazio di memoria in base alle necessità



Gestione dei file

(1)



- La gestione dei file è uno degli aspetti più visibili di un SO
- I calcolatori memorizzano informazioni su supporti fisici diversi (dischi magnetici, dischi ottici, nastri magnetici)
- Ciascuno supporto ha caratteristiche peculiari ed una organizzazione fisica propria
- Ciascuno supporto è controllato da un dispositivo (controllore del disco, controllore del nastro) con caratteristiche proprie
 - Velocità di accesso
 - Capacità
 - Tasso di trasferimento di dati
 - Metodo di accesso



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

21

Gestione dei file

(2)



- Per rendere conveniente l'utilizzo del calcolatore il SO fornisce una visione *logica* uniforme del processo di memorizzazione delle informazioni: il **file**, ossia una collezione di informazioni correlate tra loro e definite dal suo creatore
- Il file è una unità *logica* di memorizzazione che astrae le caratteristiche fisiche di dispositivi come dischi e nastri
- Il SO è responsabile delle seguenti attività per la gestione dei file:
 - Creare e cancellare file e directory (cartelle)
 - Fornire primitive per manipolare file e directory
 - Collocare i file nella memoria secondaria
 - Eseguire il backup dei file sulla memoria stabile (non volatile)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

22

Gestione della memoria secondaria



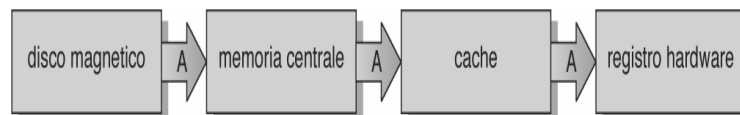
- La memoria centrale è il passaggio obbligato di qualsiasi elaborazione ma è piccola, veloce e volatile
- Il sistema di calcolo mette a disposizione la memoria *secondaria* a sostegno della memoria centrale
- La memoria secondaria è *grande, lenta e permanente*
- La maggior parte dei programmi risiede nella memoria secondaria. Un programma viene caricato in memoria centrale solo quando deve essere eseguito
- Il SO è responsabile delle seguenti attività relative alla gestione della memoria secondaria (disco):
 - Gestione dello spazio libero
 - Assegnazione dello spazio
 - Scheduling del disco
- L'efficienza complessiva di un sistema di calcolo può dipendere dalla velocità del sistema di gestione dei dischi



Gestione della memoria cache



- Il concetto di *caching* è fondamentale in ogni sistema di calcolo e consiste nel copiare le informazioni di uso più frequente in una memoria più veloce della RAM chiamata *memoria cache*
- Richiede una opportuna politica di gestione della *memoria cache*, che rappresenta un problema di progettazione molto importante
- Il caching introduce un altro livello nella gerarchia di memorie. Questo richiede che dati presenti simultaneamente su più livelli siano *consistenti*



Migrazione di A dal disco magnetico a un registro della CPU



Gestione del sistema di I/O



- In un sistema di calcolo esiste una incredibile varietà di dispositivi di I/O largamente diversi tra loro per funzioni e velocità
- I metodi di controllo di tali dispositivi sono quindi altrettanto diversi
- Uno dei compiti del SO è quello di nascondere agli utenti le caratteristiche fisiche dei vari dispositivi fornendo una interfaccia uniforme per l'accesso ai dispositivi
- Questo compito è svolto dal sottosistema di I/O, che è composto da:
 - Un sistema di gestione della memoria comprendente *buffering*, *caching* e *spooling*
 - Un'interfaccia generale per i *driver* dei dispositivi
 - I driver per gli specifici dispositivi hardware (soltanto il driver di un dispositivo conosce le caratteristiche del dispositivo cui è associato)



Protezione e sicurezza



- Se un sistema consente l'esecuzione simultanea di processi, allora i vari processi devono essere protetti l'uno dall'altro
- Il meccanismo di **protezione** serve a controllare l'accesso di programmi, processi o utenti alle risorse condivise di un sistema di calcolo
- Particolarmente utile nei sistemi multi programmati e multi utente
- Il meccanismo di protezione deve:
 - Distinguere tra uso autorizzato e non autorizzato
 - Specificare i controlli da eseguire
 - Fornire i mezzi per effettuare tali controlli
- Il meccanismo di **sicurezza** serve a proteggere il sistema da attacchi provenienti dall'interno (furto di identità ecc.) e dall'esterno del sistema stesso (virus, worms, ecc.)



Sistemi distribuiti



- Un sistema *distribuito* è un insieme di unità di elaborazione fisicamente separate che non condividono memoria o clock
- Ogni unità di elaborazione ha la propria memoria locale
- Le unità sono connesse tramite una rete di comunicazione
- La comunicazione avviene usando un *protocollo* (ftp, http)
- Un sistema distribuito gestisce l'accesso alle proprie risorse da parte degli utenti del sistema stesso
- L'accesso alle risorse condivise consente di:
 - Velocizzare il calcolo
 - Aumentare la disponibilità di dati
 - Aumentare l'affidabilità
- Richiedono una infrastruttura di rete (LAN o WAN)
- Normalmente i SO generalizzano l'accesso ad una rete come una forma di accesso a file



Altri tipi di sistemi



- **Sistemi da scrivania:** sistemi operativi per PC dedicati a un singolo utente. L'obiettivo principale è massimizzare la facilità d'uso e la velocità di risposta. Adottano tecnologie sviluppate per SO più complessi e non necessitano di un gestione avanzata della CPU
- **Sistemi palmari:** sistemi di elaborazione di dimensioni ridotte con caratteristiche di elaborazione e di memoria limitate
- **Sistemi paralleli:** sistemi con molte CPU che comunicano e che condividono memoria, clock e bus. La comunicazione avviene tramite memoria condivisa. Hanno il vantaggio di avere elevato *throughput* (numero di lavori elaborati nell'unità di tempo), di essere economici (condivisione di periferiche, chassis, ecc.) e molto affidabili (un guasto ad un processore non blocca il sistema – sistemi *fault tolerant*)
- **Sistemi real time:** presentano rigidi e ben definiti vincoli di tempo sulle operazioni della CPU e sono usati come dispositivi di controllo in applicazioni dedicate come il controllo di esperimenti scientifici, sistemi di controllo industriale, ecc.



Servizi di un SO



- Per agevolare la programmazione, un SO offre i seguenti **servizi**:
 - **Interfaccia con l'utente** – può assumere varie forme, ma la più diffusa è sicuramente quella grafica (GUI – Graphical User Interface)
 - **Esecuzione di programmi** – capacità del sistema di caricare un programma in memoria centrale e di eseguirlo
 - **Operazioni di I/O** – siccome per motivi di protezione i programmi utente non possono eseguire operazioni di I/O direttamente, il SO deve fornire i mezzi per farlo
 - **Gestione del file-system** – capacità di leggere, scrivere, creare e cancellare file
 - **Comunicazioni** – scambio di informazioni tra processi in esecuzione sullo stesso sistema di calcolo o su sistemi di calcolo differenti collegati da una rete (implementato attraverso *memoria condivisa* o *scambio di messaggi*)
 - **Rilevamento di errori** – assicurare una elaborazione corretta rilevando errori che possono verificarsi nella CPU, nella memoria, nei dispositivi di I/O o nei programmi utente



Funzioni aggiuntive di un SO



- Esistono funzioni del SO che non sono di ausilio per l'utente, ma che garantiscono il funzionamento efficiente del sistema stesso:
 - **Assegnazione di risorse** – nel caso di utenti multipli o di processi multipli in esecuzione in un medesimo istante
 - **Contabilizzazione** – registra gli utenti che usano il computer tenendo traccia di quali e quante risorse impiegano
 - **Protezione** – garantisce che tutti gli accessi alle risorse del sistema siano controllate



Interfaccia con l'utente



- Gli utenti comunicano con il SO attraverso due modi fondamentali
 - **Interprete comandi** - è il programma (*shell* in UNIX/Linux) che interpreta ed esegue le *istruzioni di controllo*, ossia i comandi dati al SO dalla riga di comando e che riguardano:
 - Creazione e gestione di processi
 - Gestione dell'I/O
 - Gestione della memoria secondaria
 - Gestione della memoria centrale
 - Accesso al file-system
 - Protezione
 - Comunicazione su rete
 - **Interfaccia grafica** – strumento più intuitivo della shell dove i comandi vengono dati al SO attraverso il mouse e vengono scelti all'interno di finestre con i relativi menù (tipiche dei sistemi Windows)



Chiamate di sistema



- Le **chiamate di sistema** (*system call*) costituiscono l'interfaccia tra i *processi* e il SO
- Generalmente disponibili sottoforma di istruzioni assembler
- Esistono linguaggi (C, C++) che consentono di eseguire direttamente le chiamate di sistema di UNIX/Linux
- Le chiamate di sistema dei SO tipo Windows sono disponibili tramite l'API Win32 per tutti i compilatori scritti per tali SO
- Le chiamate di sistema sono raggruppate in cinque categorie
 - **Controllo dei processi** (create, end, abort, load, execute, ecc.)
 - **Gestione dei file** (create, delete, open, close, read, ecc.)
 - **Gestione dei dispositivi** (request/release device, read, write, ecc.)
 - **Gestione delle informazioni** (time, date, ecc.)
 - **Comunicazione** (open/close connection, send/receive messages, ecc.)



Programmi di sistema



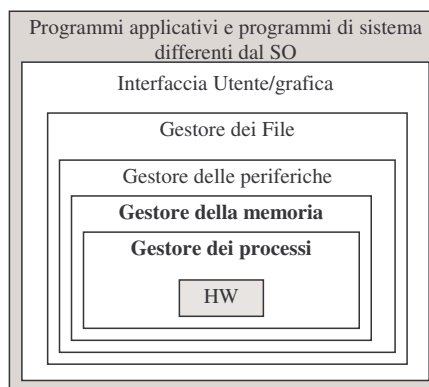
- Offrono un ambiente conveniente per lo sviluppo e l'esecuzione di programmi e si possono classificare in base alle seguenti categorie:
 - Gestione dei file (creazione, cancellazione, copia, ecc.)
 - Informazioni di stato (data, ora, spazio disponibile su disco)
 - Modifica dei file (elaborazione testi)
 - Supporto alla programmazione (compilatori, interpreti)
 - Caricamento ed esecuzione dei programmi
 - Comunicazioni (connessioni remote, invio e-mail, ecc.)
 - Programmi applicativi (browser web, fogli elettronici, ecc.)
- Gli utenti percepiscono il SO attraverso i programmi di sistema, piuttosto che dalle effettive system calls



Struttura del sistema operativo



- Il SO è suddiviso in strati o livelli:
 - Strato inferiore (strato 0) - hardware
 - Strato superiore (strato N) - interfaccia utente
- Uno strato è la realizzazione di un oggetto astratto che incapsula dati e operazioni per manipolare tali dati
- Gli strati sono costruiti in modo tale da usare solo le funzionalità e i servizi degli strati sottostanti
- Elevata modularità
- Progettazione e realizzazione più semplici



Macchine virtuali

(1)



- L'approccio stratificato conduce in modo naturale al concetto di **macchina virtuale**
- Il SO crea l'illusione che ogni processo disponga di una copia (virtuale) del computer sottostante
- Una macchina virtuale fornisce ad ogni processo una interfaccia che è *identica* all'architettura sottostante
- A utenti e applicazioni vengono mostrate risorse virtuali più semplici da usare rispetto a quelle reali (ad esempio un disco è un insieme di files)
- La corrispondenza tra risorse virtuali e risorse reali è mantenuta dal SO mascherandone la struttura
- La disponibilità di molte risorse virtuali favorisce l'esecuzione **concorrente** di più applicazioni (molteplici utenti in simultanea, utenti che usano molteplici applicazioni in simultanea)



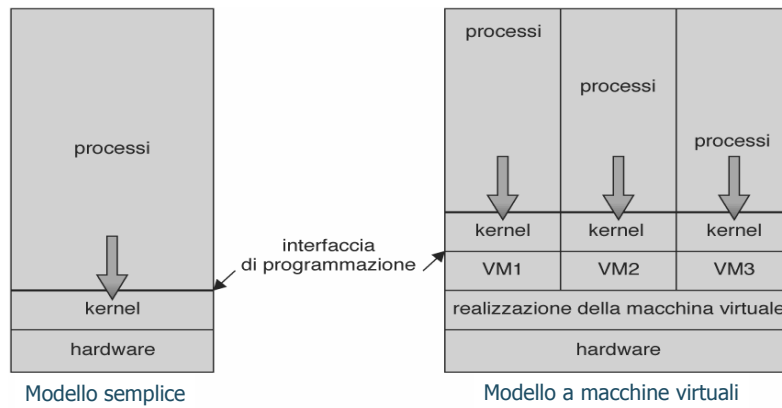
Macchine virtuali

(2)



- Le risorse del computer fisico sono condivise in modo da creare macchine virtuali
 - Lo **scheduling della CPU** crea l'illusione che ogni utente dispone del proprio processore
 - Il **gestore della memoria** crea l'illusione che ogni utente dispone della propria memoria centrale
 - Lo **spooling** e il **file system** consentono di creare stampanti e dischi virtuali
 - Un normale terminale utente di un sistema con time sharing funge da console della macchina virtuale





Gestione dei Processi

- Il concetto di **processo**
- **Scheduling** di processi
- **Context switch**
- Processi **cooperanti**

Concetto di processo

(1)



- I moderni sistemi di calcolo consentono di caricare in memoria centrale più programmi contemporaneamente e di eseguirli almeno apparentemente in parallelo
- Questa evoluzione richiede un più severo controllo dei programmi e fa nascere la nozione di **processo** come **programma in esecuzione**
- I processi si dividono in
 - Processi di sistema
 - Processi utente
- Tutti i processi di un sistema possono essere eseguiti in modalità **concorrente** con la CPU suddivisa tra loro
- Trasferendo velocemente l'uso della CPU tra i processi, il SO può diventare complessivamente più produttivo

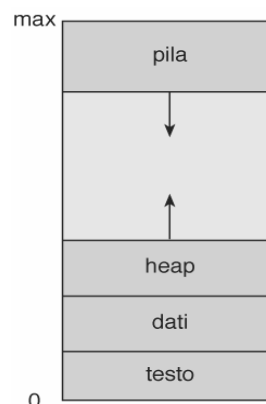


Concetto di processo

(2)



- Un **processo** è un programma in esecuzione
- L'esecuzione di un processo è *sequenziale* cioè viene eseguita al massimo un'istruzione del processo in ogni istante
- Un processo include:
 - **sezione testo** (codice del programma)
 - **attività corrente** (contenuto del program counter e dei registri della CPU)
 - **pila** (dati temporanei, cioè variabili locali e parametri di metodi)
 - **sezione dati** (variabili globali)
 - **heap** (memoria dinamicamente allocata durante l'esecuzione del processo)



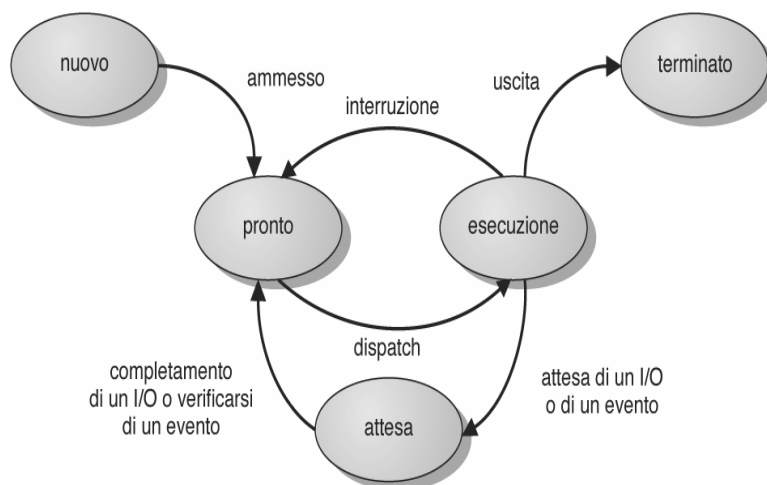
Stato di un processo



- Un programma è una entità **passiva**, un file su disco che contiene una sequenza di istruzioni, mentre un processo è un'entità **attiva**
- Durante la sua esecuzione un processo cambia **stato**, e può trovarsi in uno dei seguenti stati possibili:
 - **Nuovo**: il processo viene creato
 - **Esecuzione**: le istruzioni vengono eseguite dalla CPU
 - **Attesa**: il processo è in attesa di qualche evento
 - **Pronto**: il processo attende di essere assegnato alla CPU
 - **Terminato**: il processo ha terminato la sua esecuzione



Diagramma degli stati



Process Control Block

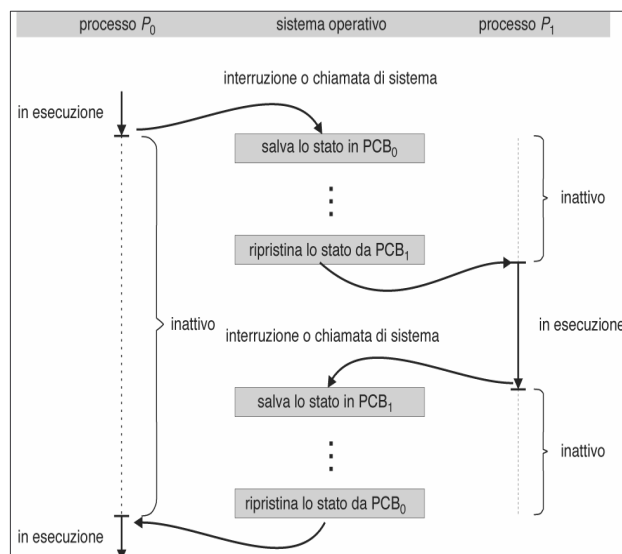


- Ciascun processo è rappresentato nel SO da un **Process Control Block (PCB)** o **descrittore del processo**
- Un PCB contiene molte informazioni associate ad un processo, tra cui le seguenti:
 - Stato del processo
 - Program counter
 - Registri della CPU
 - Informazioni sullo scheduling della CPU
 - Informazioni sulla gestione della memoria
 - Informazioni di contabilizzazione delle risorse
 - Informazioni sullo stato dell'I/O

stato del processo
numero del processo
contatore di programma
registri
limiti di memoria
elenco dei file aperti
...



Commutazione della CPU



Scheduling di processi

(1)



- Un SO di tipo time sharing gestisce tanti processi concorrenti ma solo uno di essi è eseguito in ogni istante per cui gli altri devono aspettare
- L'obiettivo è commutare la CPU tra i processi in modo da massimizzare l'uso della CPU stessa e di consentire agli utenti di interagire con i loro programmi durante la loro esecuzione, dando la sensazione che i programmi evolvano in parallelo
- Questo compito viene svolto da un programma del SO chiamato **scheduler dei processi** che seleziona, in base a qualche criterio, uno dei processi pronti all'esecuzione per eseguirlo effettivamente sulla CPU



Scheduling di processi

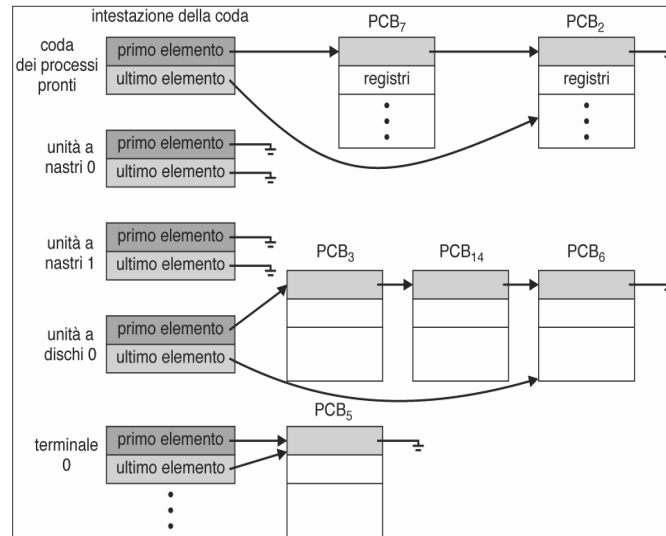
(2)



- Un solo processo viene eseguito in ogni istante per cui gli altri devono aspettare che la CPU sia libera e possa quindi essere rischedulata
- Esistono varie code in un sistema dove i processi si pongono in attesa durante il loro ciclo di vita:
 - **Coda dei processi** – tutti i processi del sistema
 - **Coda dei processi pronti** – tutti i processi residenti in memoria centrale, pronti e in attesa di esecuzione (lista concatenata dei PCB associati ai processi)
 - **Coda del dispositivo** – tutti i processi in attesa di un dispositivo di I/O (ogni dispositivo di I/O ha la propria coda di attesa)
- I processi migrano tra le varie code durante il loro ciclo di vita in base al verificarsi di opportuni eventi



Code di processi



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

47

Scheduler

(1)



- Nel corso della sua esistenza un processo si trova in diverse code di scheduling
- Il SO seleziona i processi dalle varie code attraverso un opportuno programma chiamato **scheduler**
- Esistono due diversi tipi di scheduler
 - **Scheduler a lungo termine** (*job scheduler*) seleziona i processi memorizzati su disco che vanno trasferiti nella coda dei processi pronti
 - **Scheduler a breve termine** (*CPU scheduler*) seleziona quale tra i processi pronti deve essere eseguito e assegna ad esso la risorsa CPU
- Si differenziano per la frequenza di esecuzione



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

48

Scheduler

(2)



- Lo scheduler a *breve termine* è invocato frequentemente (millisecondi) ⇒ **deve essere veloce**
- Lo scheduler a *lungo termine* è invocato con una frequenza inferiore (minuti) ⇒ **può essere lento**
- Lo scheduler a lungo termine controlla il *grado di multi programmazione* (numero di processi in memoria)
- I processi possono essere descritti come:
 - *Processi I/O-bound* – spendono più tempo per l'I/O che per l'elaborazione
 - *Processi CPU-bound* – spendono più tempo per l'elaborazione
- Lo scheduler a lungo termine deve scegliere una buona combinazione dei due tipi di processi



Cambio di contesto



- Quando la CPU viene assegnata ad un nuovo processo il SO deve salvare lo stato del vecchio processo e caricare lo stato (salvato precedentemente) del nuovo processo
- Questa operazione prende il nome di **cambio di contesto** o **context switch**
- Il contesto di un processo è salvato nel proprio PCB
- Il tempo di context switch è puro *sovraccarico* (*overhead*), infatti il sistema non esegue alcun lavoro utile durante il cambio di contesto
- Il tempo di context switch dipende dal supporto hardware ed è compreso tra 1 e 1000 microsecondi



Operazioni sui processi



- I processi in un sistema di calcolo vengono eseguiti in maniera concorrente e si devono poter creare e cancellare dinamicamente
- Il SO deve quindi fornire meccanismi per:
 - *Creare* un processo
 - *Terminare* un processo
- Esistono diversi eventi che causano la creazione di un processo e precisamente:
 - L'inizializzazione del sistema
 - L'esecuzione di una system call da parte di un processo attivo
 - Una richiesta da parte di un utente
- Esistono diversi eventi che causano la terminazione di un processo e precisamente:
 - Un processo termina dopo l'esecuzione della sua ultima istruzione
 - Un processo può terminare a causa di un errore



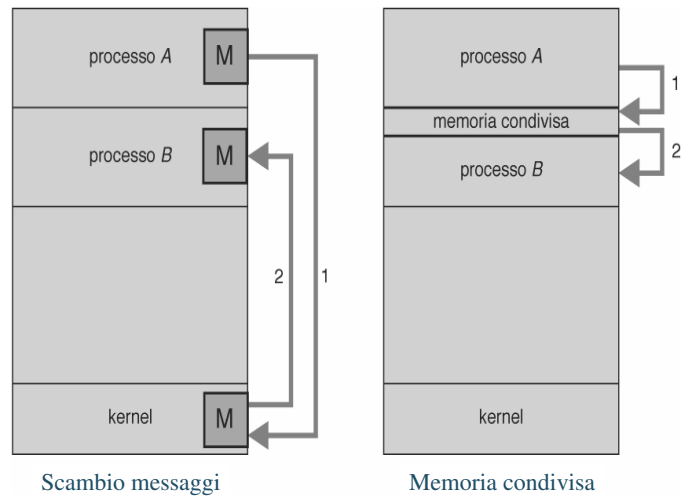
Processi cooperanti



- Ciascuno dei processi concorrenti in esecuzione in un SO può essere:
 - **Indipendente**: non può influenzare altri processi in esecuzione nel sistema né esserne influenzato (non condivide dati)
 - **Cooperante**: può influenzare altri processi in esecuzione nel sistema o esserne influenzato (condivide dati)
- Vantaggi della cooperazione tra processi:
 - Condivisione delle informazioni
 - Accelerazione del calcolo
 - Modularità
 - Convenienza (esecuzione parallela di varie attività utente, come scrittura, stampa e compilazione)
- L'esecuzione concorrente che richiede la cooperazione tra processi ha bisogno di meccanismi che consentano ai processi di **comunicare** tra loro e di **sincronizzare** le loro azioni



Modelli di comunicazione



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

53

Sincronizzazione dei processi

- Produttore/Consumatore
- Sezione critica
- Semafori
- Lettori/Scrittori
- 5 filosofi
- Threads

Produttore e consumatore

(1)



- Il problema del **produttore** e del **consumatore** è un classico paradigma per processi cooperanti definito come segue
 1. Il processo produttore produce informazioni che vengono consumate dal processo consumatore (ad esempio un compilatore produce codice assembly che viene consumato da un assembler)
 2. Il processo consumatore non può consumare un elemento non ancora prodotto
- Il produttore e il consumatore devono essere sincronizzati
- La sincronizzazione avviene tramite un **buffer di memoria condivisa** di dimensione limitata. Questo introduce ulteriori vincoli di sincronizzazione
 3. Il processo consumatore deve aspettare se il buffer è vuoto
 4. Il processo produttore deve aspettare se il buffer è pieno



Produttore e consumatore

(2)



- Il buffer di memoria condivisa viene implementato come un vettore **circolare** di dimensione finita con due puntatori logici
 - *in* che rappresenta la successiva posizione libera nel buffer
 - *out* che rappresenta la prima posizione piena nel buffer

- Dati condivisi:

```
#define Size 10      /* dimensione del buffer */  
typedef struct {  
    ...  
} elemento;  
elemento buffer[Size];  
int in = 0;  
int out = 0;  
int counter = 0;
```



- Buffer vuoto quando counter = 0
- Buffer pieno quando counter = Size



Produttore e consumatore

(3)



- Dati condivisi:

```
#define Size 10
typedef struct{
    ...
} item;
item buffer[Size];
int in = 0;
int out = 0;
int counter = 0;
```
- Buffer vuoto quando counter = 0
- Buffer pieno quando counter = Size

```
void produttore() {
    item NuovoProdotto; /* nuovo elemento da produrre */
    while (1) {
        produci un elemento in NuovoProdotto;
        while (counter == Size)
            ; /* finché il buffer è pieno non fa niente */
        buffer[in] = NuovoProdotto;
        in = (in + 1) % Size;
        counter++;
    }
}
```

Codice processo Produttore

```
void consumatore() {
    item DaConsumare; /* elemento da consumare */
    while (1) {
        while (counter == 0)
            ; /* finché il buffer è vuoto non fa niente */
        DaConsumare = buffer[out];
        out = (out + 1) % Size;
        counter--;
        consuma l'elemento in DaConsumare;
    }
}
```

Codice processo Consumatore



Produttore e consumatore

(4)



- L'accesso concorrente a dati condivisi da parte di più processi può determinare **inconsistenza** dei dati
- Mantenere la consistenza dei dati richiede meccanismi per l'esecuzione ordinata di processi cooperanti
- Il paradigma Produttore/Consumatore ben rappresenta molte questioni di sincronizzazione nei SO



Produttore e consumatore

(5)



- Osservazioni

- I due processi condividono la variabile **counter**
- Lo schema funziona nel caso concorrente solo se le istruzioni:

counter++;
counter--;

sono eseguite **atomicamente**

- Una operazione è atomica se la sua esecuzione termina senza interruzioni



Produttore e consumatore

(6)



- L'istruzione **counter++** può essere implementata in linguaggio macchina come segue:

registro₁ = counter
registro₁ = registro₁ + 1
counter = registro₁

- L'istruzione **counter--** può essere implementata in linguaggio macchina come segue:

registro₂ = counter
registro₂ = registro₂ - 1
counter = registro₂



Produttore e consumatore

(7)



- Se il produttore e il consumatore tentano di aggiornare il buffer concorrentemente, le istruzioni del linguaggio assembler possono essere alternate (**interleaving**) conservando l'ordine interno alle istruzioni di alto livello
- Il fenomeno del interleaving dipende da come i processi produttore e consumatore vengono schedulati



Produttore e consumatore

(8)



- Assumiamo che **counter** è inizialmente 5. Un possibile interleaving di istruzioni macchina è:

produttore:	registro₁ = counter	(registro ₁ = 5)
produttore:	registro₁ = registro₁ + 1	(registro ₁ = 6)
consumatore:	registro₂ = counter	(registro ₂ = 5)
consumatore:	registro₂ = registro₂ - 1	(registro ₂ = 4)
produttore:	counter = registro₁	(counter = 6)
consumatore:	counter = registro₂	(counter = 4)

- Il valore di **counter** può essere 4 o 6 quando il risultato corretto dovrebbe essere 5



Race condition



- La situazione in cui molti processi accedono e manipolano dati condivisi in maniera concorrente si chiama **race condition**
- Il valore finale del dato condiviso dipende da quale processo finisce per ultimo
- Per prevenire il verificarsi di race conditions, i processi concorrenti devono essere **sincronizzati** (un solo processo alla volta può modificare il dato condiviso)



Problema della sezione critica

(1)



- Consideriamo n processi che competono per l'uso di dati condivisi (variabili, tabelle, files)
- Ogni processo ha un segmento di codice chiamato **sezione critica** che modifica i dati condivisi
- Quando un processo sta eseguendo la propria sezione critica nessun altro processo è autorizzato ad eseguire la propria sezione critica
- L'esecuzione delle sezioni critiche dei vari processi deve essere **mutuamente esclusiva**
- Bisogna progettare un protocollo che consenta ai processi di cooperare rispettando la mutua esclusione



Problema della sezione critica

(2)



- Una soluzione al problema della sezione critica deve soddisfare i seguenti requisiti
 - **Mutua esclusione:** Se un processo sta eseguendo la sua sezione critica nessun altro processo può eseguire la propria sezione critica
 - **Progresso:** Se nessun processo sta eseguendo la propria sezione critica ed esiste qualche processo che desidera entrare nella propria sezione critica, allora la selezione del processo che entrerà per primo nella sezione critica dipende dai processi che si trovano fuori dalla sezione non critica e non può essere rimandata indefinitamente
 - **Attesa limitata:** Se un processo P chiede di entrare nella propria sezione critica, esiste un limite al numero di volte che altri processi possono entrare nella propria sezione critica prima che la richiesta di P sia stata accordata

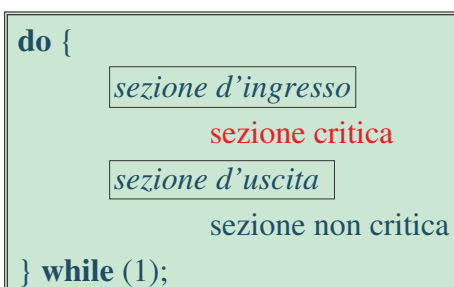


Problema della sezione critica

(3)



Struttura generale del processo P_i



La soluzione migliore è una soluzione software che fa uso di uno strumento noto sotto il nome di **semaforo**



Semafori



- Strumenti software per la sincronizzazione
- **Semaforo** = variabile intera S
- Si può accedere ad un semaforo **solo** attraverso due operazioni **atomiche** standard denominate **wait** e **signal**

```
wait(S) {  
    while ( $S \leq 0$ ) ;  
     $S--$ ;  
}
```

```
signal(S) {  
     $S++$   
}
```

- Tutte le operazioni di modifica del valore del semaforo S devono essere eseguite in maniera **indivisibile**



Sezione critica di n processi



- Dati condivisi:
semaforo mutex; (inizialmente mutex = 1)
- Processo P_i :

```
do {  
    wait(mutex);  
    sezione critica  
    signal(mutex);  
    sezione non critica  
} while (1);
```



Deadlock e starvation



- **Deadlock:** Due o più processi stanno aspettando il verificarsi di un evento che può essere causato solo da uno dei processi in attesa
- Siano S e Q due semafori inizializzati a 1

P_0	P_1
<code>wait(S);</code>	<code>wait(Q);</code>
<code>wait(Q);</code>	<code>wait(S);</code>
...	...
<code>signal(S);</code>	<code>signal(Q);</code>
<code>signal(Q);</code>	<code>signal(S);</code>

- **Starvation:** Un processo può attendere indefinitamente al semaforo dove è sospeso



Problemi di sincronizzazione



- **Produttore/Consumatore**
- **Lettori/Scrittori**
- **Cinque filosofi**



Produttore e consumatore

(1)



- Il problema del **produttore** e del **consumatore** è un classico paradigma per processi cooperanti definito come segue
 1. Il processo produttore produce informazioni che vengono consumate dal processo consumatore (ad esempio un compilatore produce codice assembleativo che viene consumato da un assemblatore)
 2. Il processo consumatore non può consumare un elemento non ancora prodotto
- Il produttore e il consumatore devono essere sincronizzati
- La sincronizzazione avviene tramite un **buffer di memoria condivisa** di dimensione limitata
- Questo introduce ulteriori vincoli di sincronizzazione
 3. Il processo consumatore deve aspettare se il buffer è vuoto
 4. Il processo produttore deve aspettare se il buffer è pieno



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

71

Produttore e consumatore

(2)



- Dati condivisi:

```
#define Size 10
```

```
typedef struct {
```

```
...
```

```
} item;
```

```
item buffer[Size];
```

```
semaforo vuote, piene, mutex;
```

Semaforo che conta le posizioni piene del buffer

Semaforo per la mutua esclusione sul buffer

- Inizialmente:

Semaforo che conta le posizioni vuote del buffer

piene = 0, vuote = Size, mutex = 1



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

72

Produttore e consumatore

(3)



```
void Produttore() {  
    item NuovoProdotto; /* nuovo elemento da produrre */  
    do {  
        produci un elemento in NuovoProdotto;  
        wait(vuote); ← Attendi se il buffer è pieno  
        wait(mutex); ← Attendi se il consumatore è in sezione critica  
        aggiungi al buffer l'elemento in NuovoProdotto;  
        signal(mutex); ← Sblocca il primo processo in attesa di entrare in sezione critica  
        signal(piene); ← Sblocca il consumatore  
    } while (1);  
}
```

Codice processo Produttore



Produttore e consumatore

(4)



```
void Consumatore() {  
    item DaConsumare; /* elemento da consumare */  
    do {  
        wait(piene); ← Attendi se il buffer è vuoto  
        wait(mutex); ← Attendi se il produttore è in sezione critica  
        rimuovi un elemento da buffer e mettilo in DaConsumare;  
        signal(mutex); ← Sblocca il primo processo in attesa di entrare in sezione critica  
        signal(vuote); ← Sblocca il produttore  
        consuma l'elemento in DaConsumare;  
    } while (1);  
}
```

Codice processo Consumatore



Lettori e Scrittori

(1)



- Un insieme di dati (file) è condiviso da processi concorrenti appartenenti a due diverse categorie:
 - **lettori** che accedono ai dati in sola lettura
 - **scrittori** che accedono ai dati in lettura/scrittura
- Se due processi, di cui almeno uno è scrittore, accedono ai dati in contemporanea si possono creare incoerenze
- I processi scrittori devono avere accesso esclusivo ai dati condivisi
- Nessun lettore attende, a meno che uno scrittore non abbia già ottenuto l'accesso ai dati condivisi



Lettori e Scrittori

(2)



- Dati condivisi:
 - int** numerolettori;
 - semaforo** mutex, scrittura;
- La variabile *numerolettori* indica il numero di lettori in esecuzione
- Il semaforo *mutex* è usato per la mutua esclusione sull'aggiornamento della variabile *numerolettori*
- Il semaforo *scrittura* è comune a lettori e scrittori ed è usato per la mutua esclusione per gli scrittori
- Inizialmente:
 - $\text{mutex} = 1, \text{scrittura} = 1, \text{numerolettori} = 0$



Lettori e Scrittori

(3)



```
void scrittore() {
    wait(scrittura);
    esegui l'operazione di scrittura;
    signal(scrittura);
}
```

Codice processo Scrittore

Attendi se c'è un lettore o uno scrittore sul buffer

Sblocca il prossimo processo lettore o scrittore in attesa sul semaforo scrittura

Attendi se sei il primo lettore e c'è uno scrittore che sta scrivendo sul buffer

Se sei l'ultimo lettore sblocca l'eventuale scrittore in attesa sul semaforo scrittura

```
void lettore() {
    wait(mutex);
    numerolettori++;
    if (numerolettori == 1)
        wait(scrittura);
    esegui l'operazione di lettura;
    signal(mutex);
    numerolettori--;
    if (numerolettori == 0)
        signal(scrittura);
    signal(mutex);
}
```

Codice processo Lettore



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

77

Lettori e Scrittori

(4)



- **Problema:** La soluzione appena proposta può causare *starvation*
- **Domanda:** Perché?
- **Suggerimento:** solo gli scrittori possono trovarsi nello stato di attesa indefinita

Esempio di starvation

Tempo	Int 1	Int 2	Int 3	Int 4	Int 5	Int 6	Int 7	Int 8	Int 9	Int 10	Int 11	Int 12	Int 13	Int 14	Int 15	Int 16	Int 17
Processo Scrittore	Green	Green	Green	Green	Green	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
Processo Lettore 1	Yellow	Yellow	Yellow	Yellow	Yellow	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green
Processo Lettore 2	Yellow	Yellow	Yellow	Yellow	Yellow	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green
Processo Lettore 3	Yellow	Yellow	Yellow	Yellow	Yellow	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green

Legenda	
Operazione riuscita:	Green
Attesa:	Yellow
Starvation:	Red



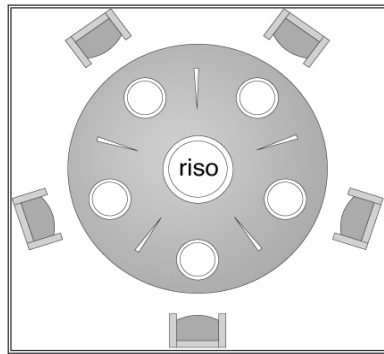
Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

78

Problema dei 5 filosofi

(1)



Dati condivisi:

semaforo bacchetta[5];

Inizialmente tutti i valori di bacchetta[5] sono 1



Problema dei 5 filosofi

(2)



```
void filosofo() {  
    do {  
        wait(bacchetta[i]);  
        wait(bacchetta[(i+1) % 5]);  
        mangia;  
        signal(bacchetta[i]);  
        signal(bacchetta[(i+1) % 5]);  
        pensa;  
    } while (1);  
}
```

Codice del filosofo *i*-esimo

Attendi se la bacchetta *i*-esima non è disponibile

Attendi se la bacchetta (*i*+1)-esima non è disponibile

Sblocca il processo in attesa di prendere la bacchetta *i*-esima

Sblocca il processo in attesa di prendere la bacchetta (*i*+1)-esima



Problema dei 5 filosofi

(3)



- **Problema:** si possono avere situazioni di stallo, come ad esempio quando tutti i filosofi hanno fame e prendono contemporaneamente la bacchetta alla loro sinistra
- **Possibili soluzioni:**
 - Un filosofo prende le sue bacchette solo se sono entrambe disponibili (operazione da eseguire in sezione critica)
 - Un filosofo dispari (pari) prende prima la bacchetta di sinistra (destra) e poi quella di destra (sinistra)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

81

Threads



- Un **processo** è un programma in esecuzione con un unico percorso di controllo e può essere visto come un modo per raggruppare risorse correlate
- La nozione di **thread** estende il concetto di processo attraverso la possibilità di avere più percorsi di controllo che consentono al processo stesso di svolgere più di un compito alla volta
- La maggiore parte dei SO moderni (Windows XP, Linux) permette l'uso dei threads



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

82

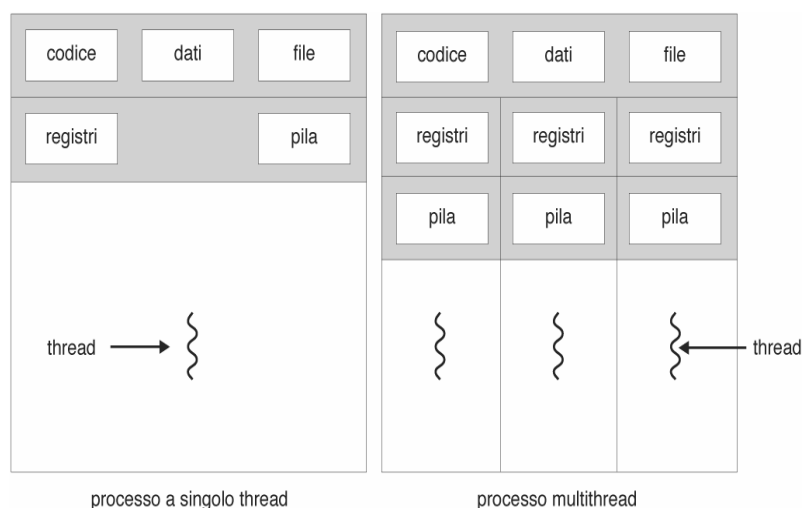
Concetto di thread



- Il **thread** o **processo leggero** è l'unità di base di utilizzo della CPU e consiste di:
 - Identificatore di thread
 - Program counter
 - Insieme di registri (variabili di lavoro correnti)
 - Stack (contiene la storia dell'esecuzione del thread)
- Un processo tradizionale è dotato di un unico thread
- Un processo **multi-thread** ha tanti thread e può compiere più di una operazione per volta
- Un thread condivide con i thread ad esso associati (cioè appartenenti allo stesso processo) le sezioni codice e dati e le risorse fornite dal SO (file aperti)



Processi single e multi thread



Motivazioni

(1)



- Molti programmi per i moderni PC sono predisposti per essere eseguiti da processi multi-thread
- Ad esempio un text editor ha un thread per ognuna delle seguenti attività:
 - Rappresentazione grafica
 - Lettura dati immessi da tastiera
 - Correzione ortografica in sottofondo
 - Salvataggio di sicurezza su disco
- Avere 4 processi separati non funzionerebbe perché i vari thread operano sullo stesso documento in quanto possono condividere dati



Motivazioni

(2)



- Una singola applicazione può dover gestire molti compiti simili tra loro
- Ad esempio:
 - Un Web server accetta da svariati client richieste di pagine Web, immagini, suoni, etc.
 - Se fosse eseguito come un programma tradizionale a singolo thread potrebbe soddisfare un client alla volta
 - Ogni client dovrebbe aspettare un tempo enorme per avere la propria richiesta servita
- In un modello multi-thread viene generato un thread per ogni richiesta da parte dei client



Vantaggi dei threads



- Tempo di risposta più veloce (un'applicazione interattiva può continuare la sua esecuzione anche se una parte di essa è bloccata per qualche motivo)
- Condivisione delle risorse e dello spazio degli indirizzi
- Economicamente è più conveniente avere processi multi-thread che creare nuovi processi (grazie alla condivisione di risorse e memoria)
- I thread possono essere creati e distrutti più velocemente perché non hanno alcuna risorsa associata
- Il passaggio del controllo da un thread ad un altro da parte della CPU è un'operazione meno costosa del context switch tra processi tradizionali



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

87



Scheduling della CPU

- **Concetti** di base
- **Criteri** di scheduling
- **Algoritmi** di scheduling

Concetti di base



- L'obiettivo della multi-programmazione è avere sempre processi in esecuzione al fine di massimizzare l'utilizzo della CPU
- Lo **scheduling** è una funzione fondamentale dei SO che si applica a quasi tutte le risorse scarse di un elaboratore ed in particolare alla CPU
- Essendo la CPU la risorsa scarsa più importante di un elaboratore, il suo scheduling è alla base della progettazione dei SO



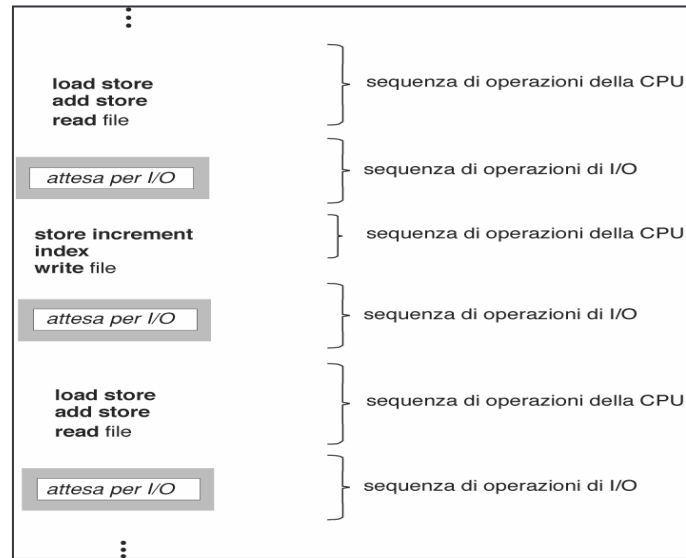
Ciclicità dell'elaborazione



- Il successo dello scheduling dipende dal fatto che i processi soddisfano la seguente proprietà del ciclo di *CPU-I/O burst*
 - L'esecuzione di un processo consiste di un ciclo di elaborazione svolto dalla CPU (*CPU burst*) e di attesa del completamento di una operazione di I/O (*I/O burst*)
 - I processi si alternano tra questi due stati
 - L'ultimo ciclo di CPU burst si conclude con una richiesta al sistema di terminare l'esecuzione



Ciclo di CPU-I/O burst



Scheduling della CPU



- Le decisioni sullo scheduling della CPU vengono prese dallo **scheduler a breve termine** (o *scheduler della CPU*) quando un processo:
 1. Passa dallo stato di *esecuzione* a quello di *attesa* (richiesta I/O)
 2. Passa dallo stato di *esecuzione* a quello di *pronto* (interruzione)
 3. Passa dallo stato di *attesa* a quello di *pronto* (completamento I/O)
 4. Termina
- Se lo scheduling interviene solo nei casi 1 e 4, allora lo schema di scheduling è **non-preemptive** (*senza diritto di prelazione*) ed è tipico dei primi sistemi Windows (la CPU rimane in possesso di un processo fino al suo rilascio)
- Altrimenti lo schema di scheduling è **preemptive** (*con diritto di prelazione*) ed è tipico dei sistemi UNIX (la CPU può essere tolta di autorità ad un processo che la detiene)



Dispatcher



- Il **dispatcher** è il modulo del SO che passa il controllo della CPU al processo selezionato dallo scheduler a breve termine
- Questa funzione riguarda:
 - Il context switch
 - Il passaggio alla modalità utente
 - Il salto alla posizione giusta del programma utente per riavviarne l'esecuzione
- Il tempo impiegato dal dispatcher per fermare un processo ed avviare l'esecuzione di un altro è noto come *latenza di dispatch*



Criteri di scheduling

(1)



- Diversi **algoritmi** per lo scheduling della CPU hanno in generale caratteristiche diverse e possono quindi favorire diverse classi di processi
- Esistono quindi vari **criteri** per il confronto tra algoritmi di scheduling:
 - **Utilizzo della CPU** – la CPU deve essere più attiva possibile
 - **Throughput** o **produttività** – numero di processi completati nell'unità di tempo
 - **Tempo di turnaround** o **tempo di completamento** – quantità di tempo necessaria per completare un particolare processo
 - **Tempo di attesa** – quantità di tempo trascorsa da un processo nella coda dei processi pronti
 - **Tempo di risposta** – quantità di tempo che intercorre tra la sottomissione di una richiesta e la prima risposta prodotta



- I **criteri** utilizzati per progettare algoritmi di scheduling sono in definitiva i seguenti:
 - Massimizzare l'utilizzo della CPU
 - Massimizzare il throughput
 - Minimizzare il tempo di turnaround
 - Minimizzare il tempo di attesa
 - Minimizzare il tempo di risposta



- Lo scheduling della CPU determina il modo in cui vengono selezionati i processi in attesa nella coda dei processi pronti
- Esistono vari algoritmi di scheduling:
 - Scheduling in **ordine d'arrivo** (First-Come First-Served - FCFS)
 - Scheduling per **brevità** (Shortest-Job-First - SJF)
 - Scheduling per **priorità**
 - Scheduling **circolare** (Round-Robin - RR)
 - Scheduling con **code multiple**
 - Scheduling con **code multiple e feedback**



Scheduling FCFS

(1)



- Lo scheduling FCFS assegna la CPU al processo che la richiede per primo
- La realizzazione dello scheduling FCFS è basata sull'implementazione della coda dei processi pronti tramite una coda FIFO
- Lo scheduling FCFS è senza prelazione e quindi non è adeguato ai sistemi a partizione di tempo



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

97

Scheduling FCFS

(2)



Esempio di scheduling FCFS

Processo	Tempo di CPU
P_1	24
P_2	3
P_3	3

Se i processi arrivano nell'ordine P_1, P_2, P_3 , e sono serviti secondo lo schema FCFS, allora il risultato dello scheduling è illustrato dal seguente diagramma di Gantt



- Tempi di attesa: $P_1 = 0, P_2 = 24, P_3 = 27$
- Tempo medio di attesa: $(0 + 24 + 27)/3 = 17$



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

98

Scheduling FCFS

(3)



- Se i processi arrivano nell'ordine: P_2, P_3, P_1 , allora lo schema di Gantt diventa:



- Tempi di attesa: $P_1 = 6, P_2 = 0, P_3 = 3$
- Tempo di attesa medio: $(6 + 0 + 3)/3 = 3$
- Molto meglio del caso precedente



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

99

Scheduling SJF

(1)



- L'algoritmo SJF associa ad ogni processo la lunghezza del suo prossimo CPU burst
- La CPU è allocata al processo con tale lunghezza più breve
- Se ce ne sono due con la stessa lunghezza allora FCFS
- SJF senza prelazione** – una volta che la CPU è assegnata a un processo, esso ne rimane in possesso fino al termine del suo CPU burst
- SJF con prelazione** – se arriva un nuovo processo il cui prossimo CPU burst ha lunghezza più breve del tempo di esecuzione rimanente del processo corrente, allora c'è prelazione (*Shortest Remaining Time First*)
- Si dimostra che SJF è *ottimo* rispetto al tempo medio di attesa, cioè minimizza il tempo medio di attesa per un dato insieme di processi



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

100

Scheduling SJF

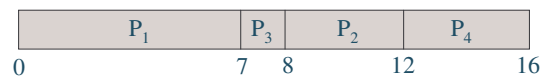
(2)



Esempio di non-preemptive SJF

Processo	Tempo di arrivo	Tempo di Burst
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- Con SJF non-preemptive l'ordine migliore è P_1, P_3, P_2, P_4 che determina il seguente schema di Gantt



- Tempo medio di attesa = $(0 + 6 + 3 + 7)/4 = 4$



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

101

Scheduling SJF

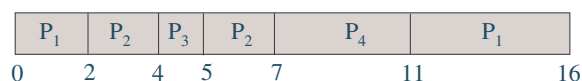
(3)



Esempio di preemptive SJF

Processo	Tempo di arrivo	Tempo di Burst
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- Il diagramma di Gantt diventa il seguente



- Tempo medio di attesa = $(9 + 1 + 0 + 2)/4 = 3$



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

102

Scheduling SJF

(4)



Problema: L'algoritmo SJF non può sapere a priori la lunghezza del prossimo CPU burst di un processo

Soluzione: La lunghezza del prossimo CPU burst può essere stimata con la *media esponenziale* delle lunghezze effettive dei CPU burst precedenti:

- Sia t_n la lunghezza effettiva dell' n -esimo CPU burst (storia recente)
- Sia τ_{n+1} la lunghezza prevista del prossimo CPU burst (il valore di τ_n rappresenta quindi la storia passata)
- Sia α tale che $0 \leq \alpha \leq 1$ (peso relativo della storia recente e passata)
- Definiamo: $\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$



Scheduling SJF

(5)

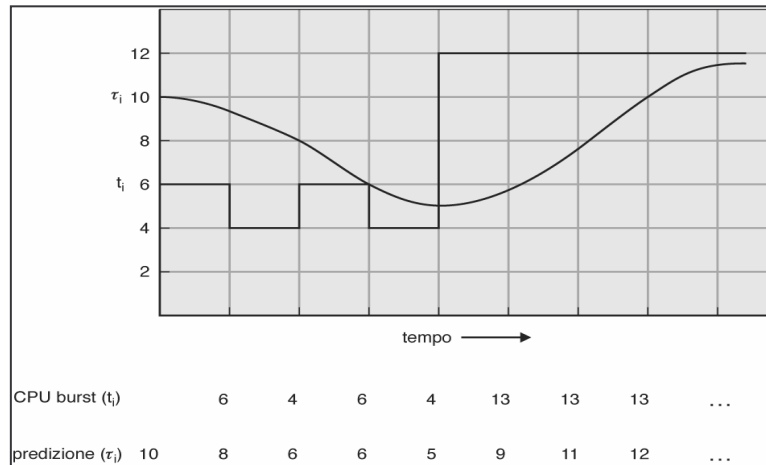


- Se $\alpha = 0$ allora $\tau_{n+1} = \tau_n$ e la storia recente non conta
- Se $\alpha = 1$ allora $\tau_{n+1} = t_n$ e conta solo l'ultimo CPU burst
- Se $\alpha = 1/2$ allora la storia recente e quella passata hanno lo stesso peso
- Se espandiamo la formula $\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$ otteniamo:
$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \alpha t_{n-1} + (1 - \alpha)^2 \alpha t_{n-2} + \dots$$
$$+ (1 - \alpha)^j \alpha t_{n-j} + \dots + (1 - \alpha)^{n+1} \tau_0$$
- Siccome sia α che $(1 - \alpha)$ sono minori o uguali a 1, allora ogni termine successivo ha un peso inferiore a quello del suo predecessore



Scheduling SJF

(6)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

105

Scheduling per priorità

(1)



- Una **priorità** (numero intero) è associata ad ogni processo
- La CPU è allocata al processo con la priorità più alta (intero più piccolo equivale ad una priorità maggiore)
- Lo scheduling con priorità può essere preemptive o non-preemptive
- Lo schema SJF è uno scheduling con priorità dove la priorità è la lunghezza prevista del prossimo CPU burst
- Un problema dello scheduling con priorità è il fenomeno denominato **starvation** (*morte per fame*) in base al quale processi a bassa priorità possono aspettare all'infinito
- Una possibile soluzione è costituita dalla tecnica denominata **invecchiamento** (*aging*), secondo la quale il tempo che passa aumenta la priorità dei processi in attesa



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

106

Scheduling per priorità

(2)



Esempio scheduling per priorità non-preemptive

Processo	Tempo di burst	Priorità
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Se supponiamo che i processi sono arrivati tutti al tempo 0, allora il diagramma di Gantt risultante è il seguente



- Tempo medio di attesa = $(6 + 0 + 16 + 18 + 1)/5 = 8.2$



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

107

Scheduling Round Robin

(1)



- Detto anche scheduling **circolare**, è stato progettato appositamente per i SO di tipo time-sharing
- Simile a FCFS ma con in più il diritto di prelazione
- Ogni processo riceve la risorsa CPU per un **quanto di tempo** (10-100 milli-secondi)
- Al termine del quanto di tempo il processo è prelazonato e aggiunto alla fine della coda dei processi pronti (coda circolare)
- La coda dei processi pronti è una coda FIFO



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

108

Scheduling Round Robin

(2)



- Se ci sono n processi nella coda dei processi pronti e il quanto di tempo è q , allora ogni processo riceve $1/n$ -esimo del tempo di CPU in frazioni di al più q unità di tempo
- Nessun processo aspetta più di $(n - 1)q$ unità di tempo
- Le prestazioni di RR dipendono dal quanto di tempo
 - Se q è grande allora RR si riduce a FCFS
 - Se q è piccolo allora q deve essere più grande rispetto al tempo di context switch, altrimenti l'overhead sarebbe troppo elevato



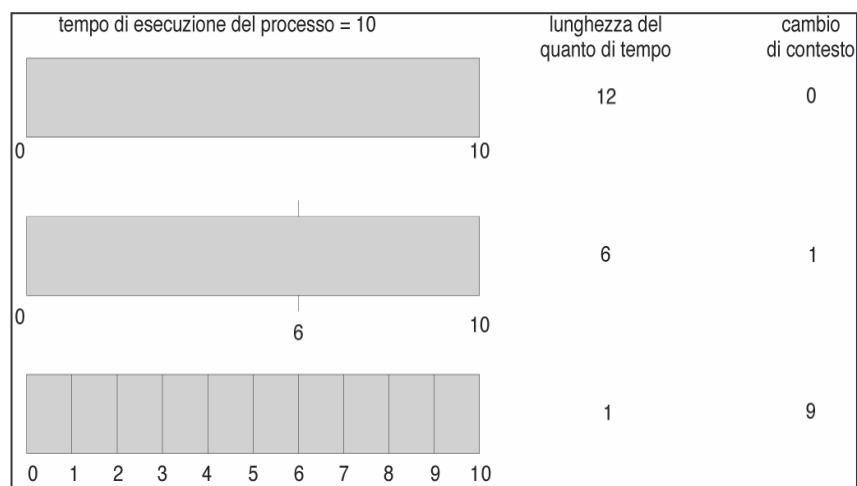
Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

109

Scheduling Round Robin

(3)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

110

Scheduling Round Robin

(4)

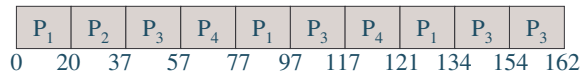


Esempio di Scheduling RR

Processo	Tempo di CPU
P_1	53
P_2	17
P_3	68
P_4	24

Supponiamo che i processi sono arrivati nell'ordine $P_1 P_2 P_3 P_4$

- Se $q = 20$ il diagramma di Gantt è il seguente



- Tipicamente RR ha turnaround medio e tempo medio di attesa più alti rispetto a SJF ma tempo di risposta migliore



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

111

Scheduling a code multiple

(1)



- Algoritmi di scheduling adatti a situazioni in cui i processi possono essere classificati in gruppi diversi
- La coda dei processi pronti è partizionata in code separate a seconda delle necessità dei processi:
 - Processi in primo piano (*foreground*) – processi interattivi
 - Processi in sottofondo (*background*) – processi batch
- Un processo è assegnato a una coda in modo permanente
- Ogni coda ha il proprio algoritmo di scheduling:
 - Coda dei processi foreground – RR
 - Coda dei processi background – FCFS



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

112

Scheduling a code multiple

(2)



- Lo scheduling deve essere fatto anche tra le varie code:
 - *Scheduling a priorità fissata* – serve prima i processi foreground e poi quelli background (possibilità di *starvation*)
 - *Quanto di tempo* – ogni coda riceve una certa quantità di tempo di CPU che può schedulare tra i suoi processi (ad esempio, 80% alla coda dei processi foreground in RR, e 20% alla coda dei processi background in FCFS)



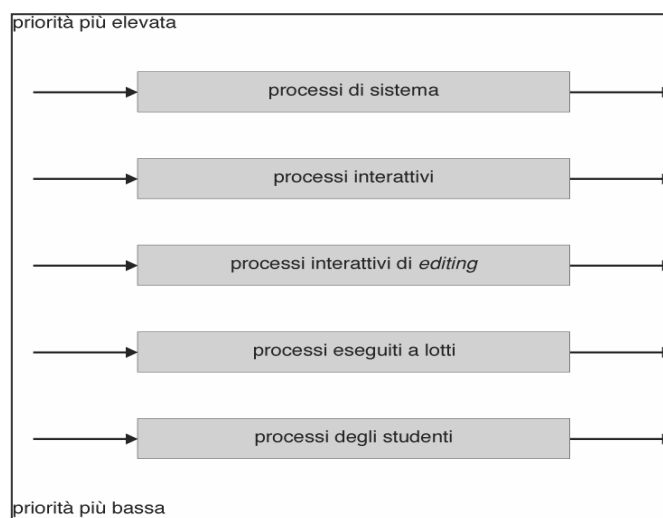
Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

113

Scheduling a code multiple

(3)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

114

Code multiple con feedback

(1)



- Rispetto al caso di code multiple un processo può spostarsi tra le varie code
- Uno scheduler a code multiple con feedback è definito dai seguenti parametri:
 - Numero di code
 - Algoritmo di scheduling per ogni coda
 - Metodo usato per determinare quando spostare un processo in una coda a priorità più alta
 - Metodo usato per determinare quando spostare un processo in una coda a priorità più bassa
 - Metodo usato per determinare in quale coda posizionare un processo nel momento in cui richiede un servizio
- Criterio di scheduling più generale e più complesso



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

115

Code multiple con feedback

(2)



- Tre code:
 - Q_0 – FCFS con quanto di tempo di 8 milli-secondi
 - Q_1 – FCFS con quanto di tempo di 16 milli-secondi
 - Q_2 – FCFS
- Scheduling
 - Un nuovo job entra in Q_0 ed è servito con politica FCFS. Quando il job ottiene la CPU, riceve 8 milli-secondi. Se non finisce in 8 milli-secondi, viene spostato in Q_1
 - In Q_1 il job è servito con politica FCFS e riceve 16 milli-secondi addizionali. Se non riesce ancora a terminare, viene prelazonato e spostato in Q_2



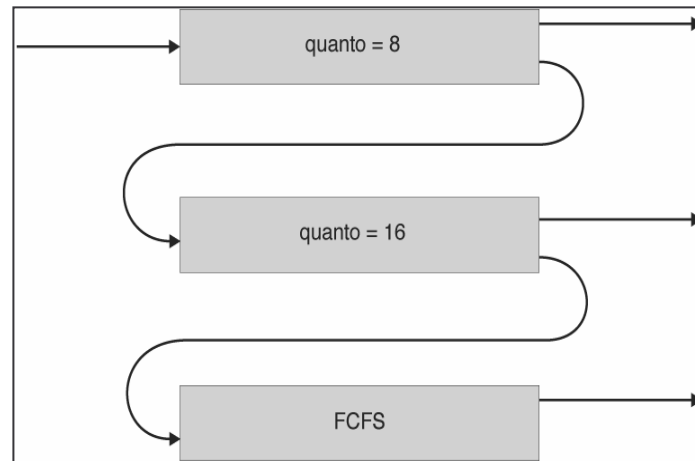
Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

116

Code multiple con feedback

(3)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

117



Gestione della RAM

- Allocazione **contigua**
- **Paginazione**
- **Segmentazione**
- Memoria **virtuale**

Background



- La memoria centrale (RAM) è un vettore di parole ognuna con il proprio indirizzo
- La CPU preleva le istruzioni dalle RAM sulla base del contenuto del program counter
- Le istruzioni possono determinare ulteriori letture (load) e scritture (store) in memoria
- Il tipico ciclo di esecuzione di una istruzione prevede
 - Prelievo** dalla RAM
 - Decodifica** (con eventuale prelievo degli operandi)
 - Esecuzione** (con eventuale utilizzo degli operandi)
- La memoria vede solo un flusso di indirizzi ma non sa come sono generati dalla CPU e a cosa servono
- Possiamo quindi ignorare come un programma in esecuzione genera un indirizzo di memoria e prestare attenzione solo alla sequenza degli indirizzi generati dal programma stesso

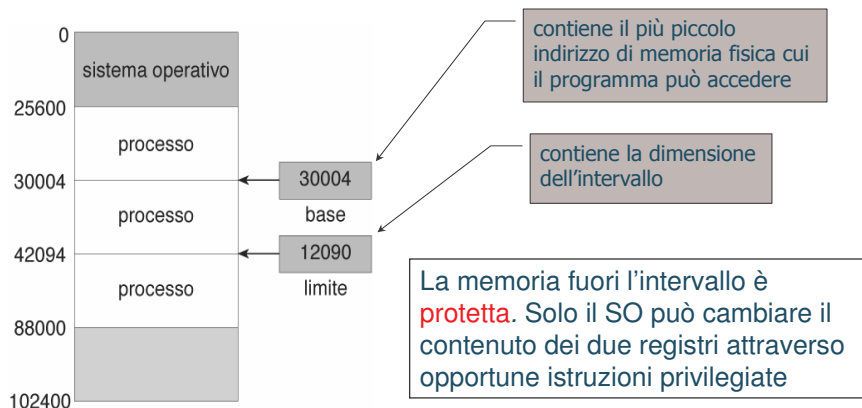


Protezione della memoria

(1)

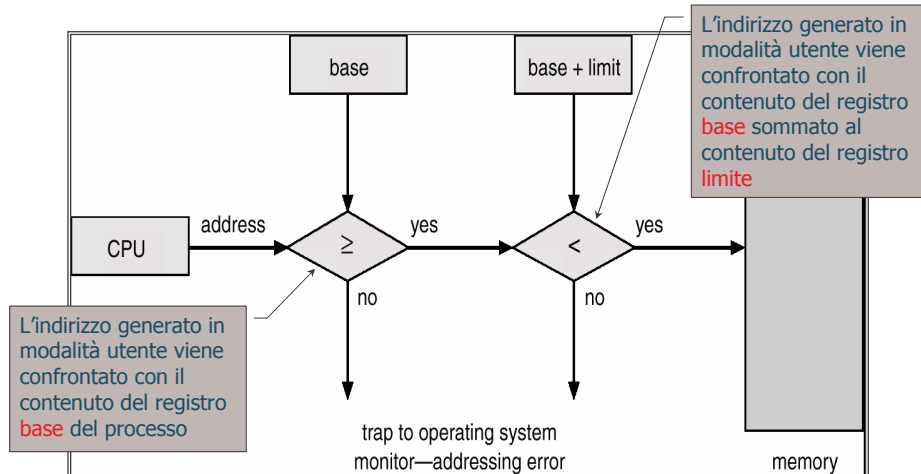


- Bisogna proteggere la memoria associata ad ogni processo in modo da garantire che i processi presenti nel sistema abbiano spazi di memoria separati. Per fare ciò vengono usati due registri per ogni processo che determinano l'intervallo degli indirizzi legali cui il processo può accedere



Protezione della memoria

(2)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

121

Associazione degli indirizzi

(1)



- Un programma in generale risiede su disco in forma di un file binario eseguibile
- Un programma per poter essere eseguito deve essere portato in RAM e inserito all'interno di un processo
- Durante la sua esecuzione un processo può essere trasferito da memoria centrale a disco e viceversa
- L'insieme dei processi su disco in attesa di essere trasferiti in memoria centrale forma la **coda di input**
- Il SO sceglie uno dei processi nella coda di input e lo carica in RAM
- Al termine dell'esecuzione del processo lo spazio di memoria da esso occupato diventa di nuovo disponibile



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

122

Associazione degli indirizzi

(2)



- I SO consentono ai processi utente di risiedere in qualsiasi parte della memoria fisica. Questo influisce sugli indirizzi che un programma utente può usare
- I programmi utente passano attraverso diverse fasi prima di essere eseguiti e tutte queste fasi coinvolgono la memoria
- In generale gli indirizzi di un programma sorgente sono **simbolici** o **logici** (ad esempio, i nomi delle sue variabili)
- Il **binding** o **associazione** di istruzioni e dati a indirizzi di memoria avviene nei moderni SO in fase di esecuzione del programma



Spazi di indirizzi logici e fisici



- Il concetto di **spazio logico degli indirizzi** che è associato ad un separato **spazio fisico degli indirizzi** è centrale per una adeguata gestione della memoria
 - **Indirizzo logico**: generato dalla CPU (chiamato anche *indirizzo virtuale*)
 - **Indirizzo fisico**: indirizzo visto dall'unità di memoria, cioè caricato nel MAR (*Memory Address Register*)



Unità di gestione della memoria



- Il programma utente lavora con indirizzi **logici** e non vede mai i reali indirizzi **fisici**
- L'**unità di gestione della memoria** (Memory Management Unit - MMU) è il dispositivo hardware che associa indirizzi logici a indirizzi fisici nella fase di esecuzione
- La trasformazione da indirizzi logici in indirizzi fisici può essere eseguita in vari modi (che vedremo nel seguito)



Allocazione contigua

(1)



- La memoria centrale è di solito suddivisa in due partizioni:
 - **Sistema operativo**: generalmente residente nella memoria bassa
 - **Processi utente**: generalmente residenti nella memoria alta
- Come assegnare la memoria ai vari processi utente?
- **Allocazione contigua** con partizioni di dimensione fissa
 - Ogni processo è contenuto in una singola sezione contigua della memoria
 - Per proteggere i processi utente l'uno dall'altro e per proteggere il sistema operativo dai processi utente vengono aggiunti al hardware del sistema un **registro di rilocalizzazione** e un **registro limite**
 - Il registro di **rilocalizzazione** contiene il valore del più *piccolo* indirizzo *fisico* di un processo
 - Il registro **limite** contiene invece l'*intervallo* degli indirizzi *logici* di un processo

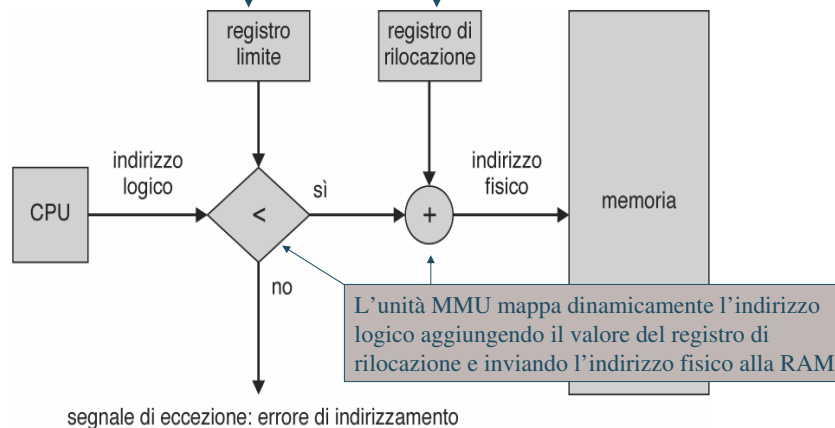


Allocazione contigua

(2)



Quando lo scheduler seleziona un processo da eseguire, il dispatcher carica i registri di rilocalizzazione e limite in fase di context switch con i valori corretti



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

127

Allocazione contigua

(3)



- La memoria è divisa in **partizioni** di dimensione *fissa*
- Ogni partizione contiene **esattamente** un processo
- Il grado di multi-programmazione è limitato dal numero di partizioni
- Quando un processo deve entrare in memoria si verifica se esiste una partizione libera e lo si carica
- Quando un processo termina, la partizione da lui occupata diventa libera per qualche altro processo
- Usato in passato nel SO IBM OS/360 ma attualmente non più in uso



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

128

Allocazione contigua

(4)

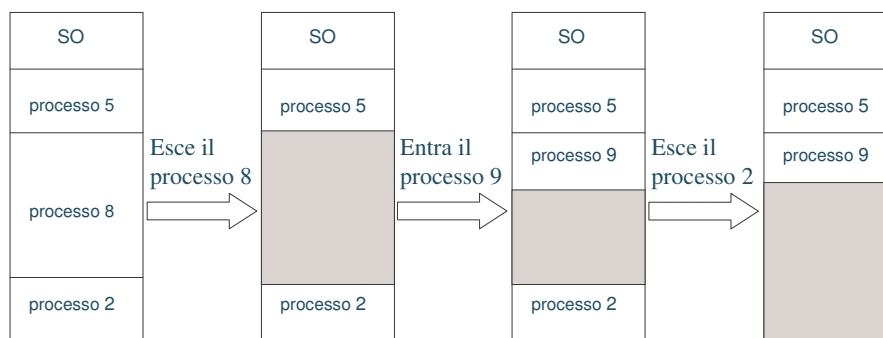


- Una variante dell'allocazione contigua con partizioni di dimensione fissa è l'**allocazione contigua con partizioni di dimensioni multiple**
 - Il sistema operativo mantiene una tabella con informazioni circa:
 - partizioni occupate
 - partizioni libere (**buchi**)
 - Inizialmente tutta la memoria è libera e costituisce un unico buco
 - In generale, buchi di varie misure sono presenti in memoria
 - Quando un processo deve essere caricato in memoria viene posizionato in un buco sufficientemente grande da contenerlo (se esiste), la parte rimanente del buco va a formare un nuovo buco
 - Quando un processo esce dalla memoria rilascia il buco che lo conteneva che diventa disponibile per altri processi. Se tale buco è adiacente ad altri buchi, essi vengono uniti per formare un buco più grande



Allocazione contigua

(5)



Allocazione contigua

(6)



- Data una lista di buchi liberi come viene soddisfatta una richiesta di memoria di dimensione n ?
 - **First-fit**: Alloca il **primo** buco che è grande abbastanza
 - **Best-fit**: Alloca il **più piccolo** buco che è grande abbastanza; deve cercare l'intera lista, a meno che i buchi non siano ordinati in base alla taglia. Produce le più piccole parti di buchi inutilizzate
 - **Worst-fit**: Alloca il buco **più grande**; deve cercare tutta la lista. Produce le più grandi parti di buchi inutilizzate
- È stato dimostrato attraverso simulazioni che first-fit e best-fit sono migliori di worst-fit in termini di velocità e memoria utilizzata. In generale il metodo first-fit risulta essere il più veloce



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

131

Allocazione contigua

(7)



- Il meccanismo dell'allocazione contigua genera due diverse problematiche di **frammentazione della memoria RAM**
 - **Frammentazione esterna** – lo spazio totale di memoria libera è sufficiente a soddisfare una richiesta ma non è contiguo (buchi residui generati dalle tecniche di allocazione dinamica)
 - **Frammentazione interna** – la memoria allocata può essere leggermente più grande di quella richiesta; questa differenza è memoria interna a una partizione ma non viene usata
 - Si può ridurre la frammentazione esterna tramite **compattazione**:
 - Riordinare il contenuto della memoria per riunire la memoria libera in un unico grosso blocco
 - La compactazione è possibile solo se la rilocalizzazione è dinamica e viene fatta a tempo di esecuzione e può essere costosa
 - Ridurre la frammentazione consentendo la non contiguità dello spazio fisico (**paginazione** e **segmentazione**)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

132

Paginazione della memoria

(1)



- Metodo di gestione della memoria che consente che lo spazio degli indirizzi fisici di un processo non sia contiguo
- La **memoria fisica** viene divisa in blocchi di dimensione fissa (compresa tra 512 byte e 16 MB) chiamati **frames**
- La **memoria logica** è divisa in blocchi della stessa dimensione detti **pagine**
- Bisogna tenere traccia di tutti i frames liberi
- Per eseguire un programma di taglia pari a n pagine, bisogna trovare n frames liberi e caricare il programma
- Il programma per poter essere eseguito deve risiedere interamente in memoria centrale



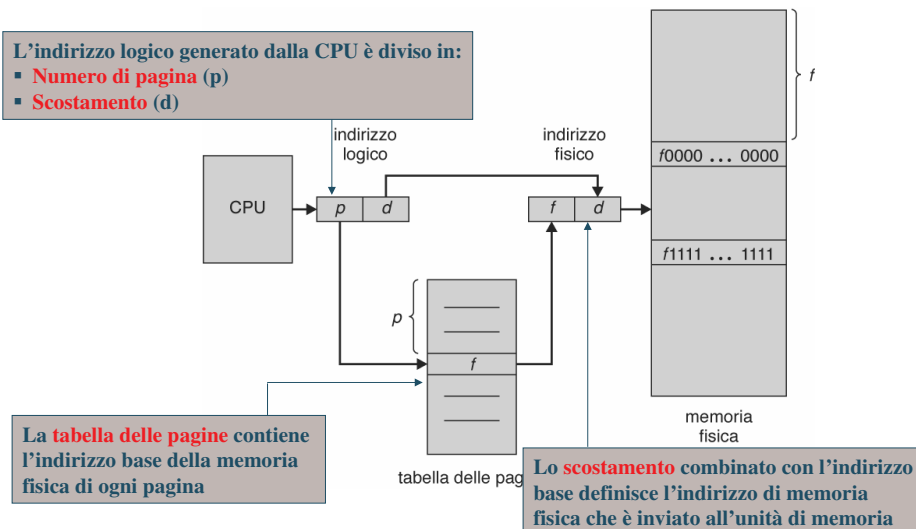
Paginazione della memoria

(2)



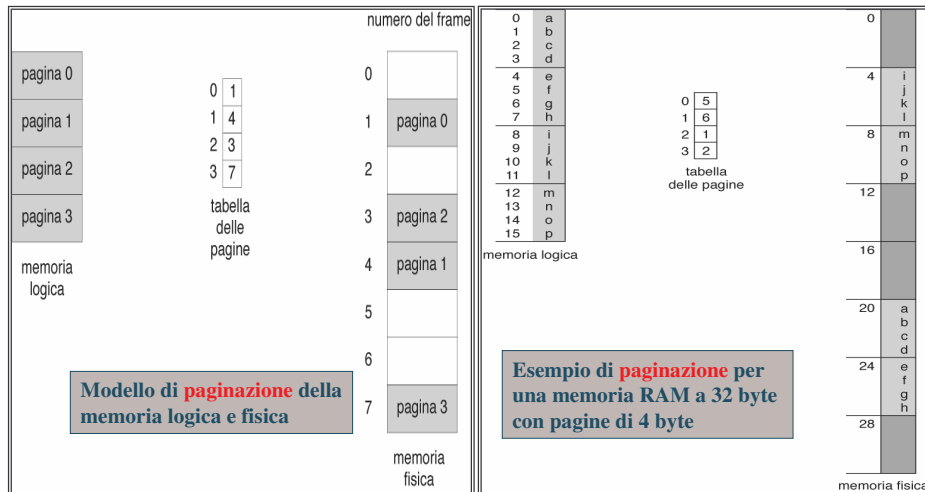
L'indirizzo logico generato dalla CPU è diviso in:

- **Numero di pagina (p)**
- **Scostamento (d)**



Paginazione della memoria

(3)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

135

Paginazione della memoria

(4)



- **Vantaggi/svantaggi**
 - La paginazione evita la frammentazione esterna, in quanto qualsiasi frame di memoria libera si può assegnare ad un processo che ne ha bisogno
 - La paginazione causa frammentazione interna, in quanto i frames di memoria sono unità indivisibili e l'ultimo frame assegnato ad un processo può non essere completamente pieno
 - Si può rimediare al problema della frammentazione interna facendo in modo che la dimensione delle pagine sia piccola. Come vedremo successivamente questa soluzione aumenta il numero trasferimenti tra memoria centrale e disco

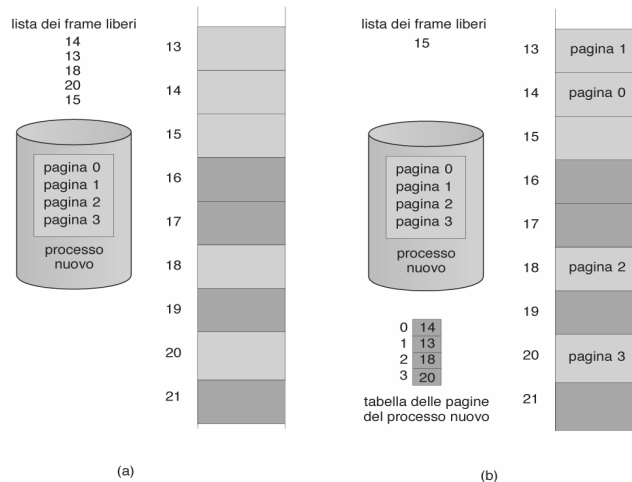


Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

136

Come vengono assegnati i frames liberi ad un nuovo processo?



- Il SO deve sapere in ogni istante
 - Il numero totale dei frames di memoria
 - Quali frames di memoria sono assegnati
 - Quali frames di memoria sono liberi
- In generale queste informazioni sono contenute in una struttura dati chiamata **tabella dei frames di memoria**
 - Contiene un elemento per ogni frame di memoria che indica se il frame è libero oppure assegnato, e se è assegnato a quale pagina di quale processo



Segmentazione della memoria

(1)



- Schema di gestione della memoria che supporta la visione della memoria da parte dell'utente che *vede* un programma come una collezione di **segmenti** di lunghezza variabile
- Un *segmento* è una unità logica come ad esempio:
 - Programma principale
 - Procedura o funzione
 - Metodo
 - Oggetto
 - Tabella, matrice, lista ecc.
- Uno spazio di indirizzi logici è quindi una raccolta di **segmenti** ognuno con un nome e una lunghezza



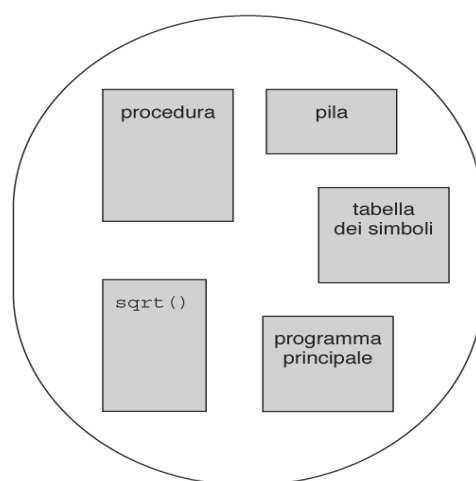
Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

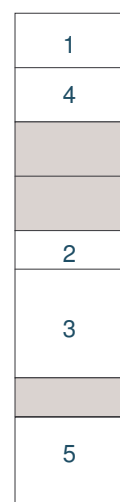
139

Segmentazione della memoria

(2)



Spazio utente



Spazio memoria fisica



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

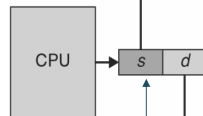
140

Segmentazione della memoria

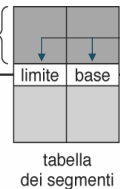
(3)



La traduzione dell'indirizzo logico in indirizzo fisico avviene tramite la **tabella dei segmenti**



L'indirizzo logico generato dalla CPU è una coppia:
 ▪ **Numero di segmento (s)**
 ▪ **Scostamento (d)**



Ogni elemento della **tabella dei segmenti** è una coppia:
 ▪ **base**: indirizzo fisico di partenza della memoria in cui risiede il segmento
 ▪ **limite**: lunghezza del segmento



si



no

Lo scostamento **d** viene confrontato con il valore **limite** del segmento

segnale di eccezione: errore di indirizzamento

memoria fisica



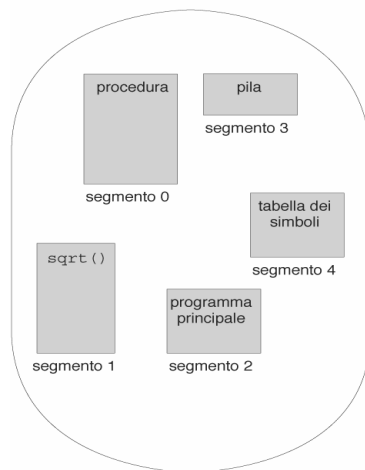
Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

141

Segmentazione della memoria

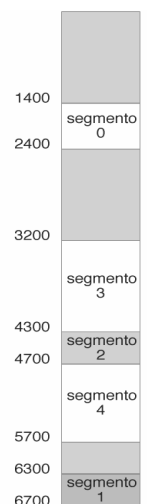
(4)



spazio degli indirizzi logici

	limite	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

tabella dei segmenti



memoria fisica



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

142

- Vantaggi/svantaggi
 - Associazione della protezione con i segmenti attraverso i valori di base e limite per i segmenti
 - Possibilità di condivisione di codice e dati attraverso la condivisione di segmenti
 - Come nel caso dell'allocazione contigua con partizioni variabili l'assegnazione della memoria ai segmenti è un problema di allocazione dinamica della memoria
 - Può causare frammentazione esterna se tutti i blocchi di memoria liberi sono troppo piccoli per contenere un segmento. In questi casi si può ad esempio rimandare l'esecuzione di un processo finché la compattazione non genera un buco sufficientemente grande



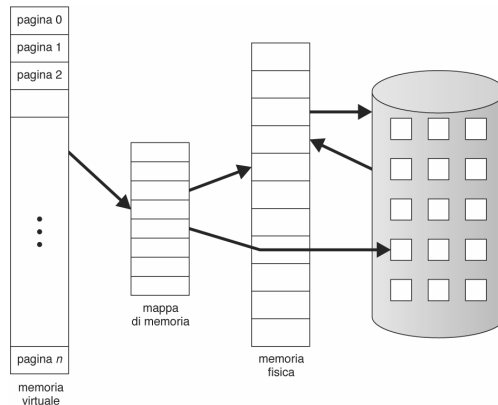
- I meccanismi di gestione della memoria centrale visti finora cercano di garantire la presenza del maggior numero di processi in RAM per permettere la multiprogrammazione
- Essi richiedono però che un processo per poter essere eseguito risieda completamente in memoria centrale
- Cosa accade se dobbiamo eseguire un processo che non può stare completamente in memoria centrale?



Memoria virtuale



- La **memoria virtuale** è quel meccanismo che consente di eseguire processi che sono più grandi della memoria fisica realizzando una vera separazione tra la memoria logica vista dall'utente e la memoria fisica stessa
- In generale solo una parte di un programma ha bisogno di stare in RAM per l'esecuzione
- La memoria logica (vista dall'utente) può essere molto più grande della memoria fisica
- Libera gli utenti dal problema dei limiti della memoria fisica
- La memoria virtuale può essere realizzata tramite:
 - **Paginazione su richiesta**
 - Segmentazione su richiesta



Paginazione su richiesta

(1)



- I processi risiedono su disco e per eseguirli bisogna caricarli in memoria centrale
- Un processo è un insieme di pagine e non uno spazio di indirizzi di memoria contigui
- Una pagina viene caricata in memoria centrale solo quando è necessario
 - Meno operazioni di I/O
 - Meno memoria fisica richiesta
 - Risposte più veloci
 - Più utenti
- Il modulo del SO che si occupa del caricamento delle pagine da disco è il **paginatore**



Paginazione su richiesta

(2)



- La corrispondenza tra indirizzi logici e indirizzi fisici viene realizzata tramite la **tabella delle pagine**
- Il paginatore però ha bisogno di poter distinguere tra pagine presenti in RAM e pagine su disco
- Ad ogni elemento della tabella delle pagine viene associato un **bit di validità** ($1 \Rightarrow$ in-RAM, $0 \Rightarrow$ non-in-RAM)
- Inizialmente il bit di validità è 0 per ogni pagina
- Se un processo tenta di accedere ad una pagina il cui bit di validità nella tabella delle pagine è 0 allora si verifica un cosiddetto **page fault** (eccezione di pagina mancante al SO)



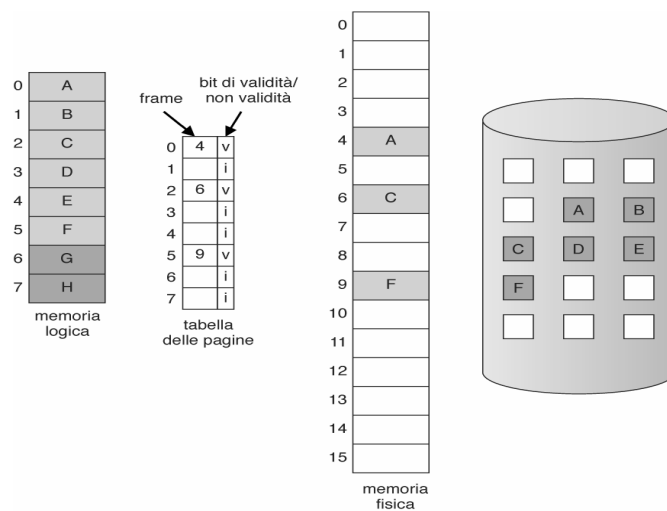
Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

147

Paginazione su richiesta

(3)



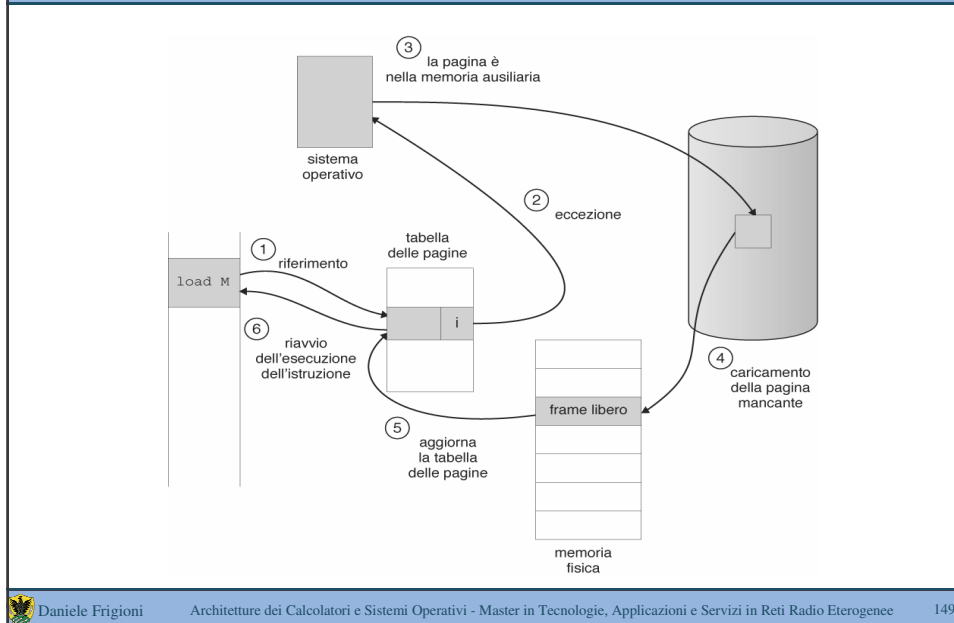
Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

148

Paginazione su richiesta

(4)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

149

Paginazione su richiesta

(5)



- La paginazione su richiesta ha un effetto rilevante sulle prestazioni di un sistema in quanto un page fault determina le seguenti azioni
 - Segnale di eccezione al SO
 - Salvataggio dello stato del processo
 - Determinazione della natura del segnale di eccezione
 - Determinazione della locazione della pagina su disco
 - Lettura dal disco e trasferimento in un frame di memoria libero
 - Durante l'attesa, scheduling della CPU
 - Segnale di interruzione del controllore del disco (I/O completato)
 - Salvataggio dello stato del processo corrente
 - Determinazione della natura del segnale proveniente dal disco
 - Aggiornamento della tabella delle pagine
 - Attesa che la CPU sia nuovamente assegnata a questo processo
 - Recupero dello stato del processo e ripresa dell'istruzione interrotta



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

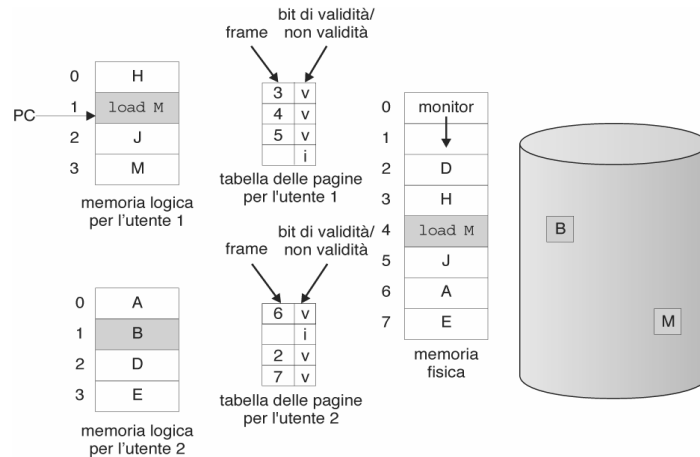
150

Sostituzione delle pagine

(1)



- Cosa accade se al verificarsi di un page fault non ci sono frames liberi in RAM?



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

151

Sostituzione delle pagine

(2)



- In questi casi è necessario eseguire un algoritmo di **sostituzione delle pagine** che cerca qualche pagina in memoria centrale che non è attualmente in uso ed effettua lo swap con la pagina richiesta
- Abbiamo bisogno di un algoritmo di sostituzione che tenti di minimizzare il numero di page faults
- La sostituzione delle pagine completa la separazione tra memoria logica e memoria fisica
- Una grande memoria virtuale può essere fornita su una piccola memoria fisica



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

152

Sostituzione delle pagine

(3)



Algoritmo base di sostituzione

1. Trova la locazione su disco della pagina richiesta
2. Cerca un frame libero in memoria centrale:
 - se esiste, usalo
 - altrimenti, usa un algoritmo di sostituzione di pagine per scegliere un frame **vittima**
 - scrivi la pagina **vittima** sul disco
3. Trasferisci la pagina richiesta nel frame appena liberato
4. Aggiorna le tabelle delle pagine e dei frames
5. Riavvia il processo



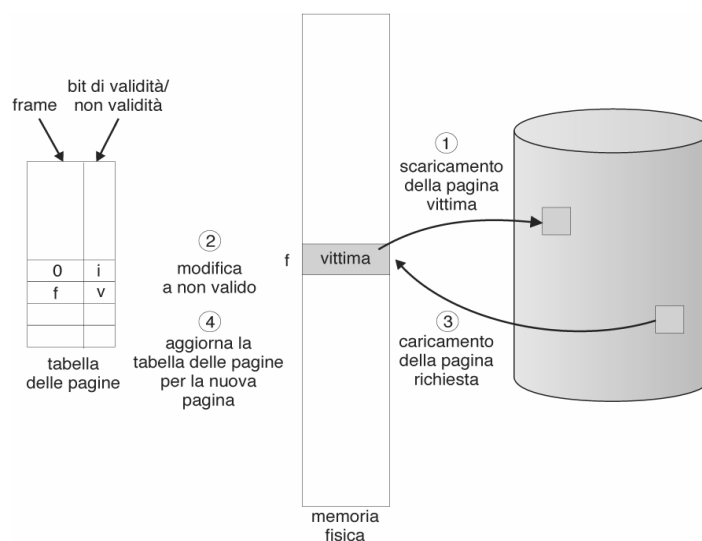
Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

153

Sostituzione delle pagine

(4)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

154

Sostituzione delle pagine

(5)



- Per realizzare la paginazione su richiesta bisogna risolvere due problemi principali
 - Sviluppare un **algoritmo di sostituzione delle pagine** (quando è richiesta la sostituzione di una pagina occorre decidere i frames di memoria da sostituire)
 - Sviluppare un **algoritmo di assegnazione dei frames di memoria** (se nella memoria sono presenti più processi occorre decidere quanti frames di memoria assegnare a ciascun processo)



Algoritmi di sostituzione delle pagine



- La sostituzione delle pagine è fondamentale per la realizzazione della paginazione su richiesta e bisogna quindi progettare algoritmi efficienti per essa
- Viene scelto in generale l'algoritmo con **minima frequenza di page fault**
- Un algoritmo viene valutato eseguendolo su una particolare stringa di riferimenti alla memoria (*successione dei riferimenti*) e calcolando il numero di page faults su quella stringa



Algoritmo di sostituzione FIFO (1)

(1)



- Associa ad ogni pagina l'istante di tempo in cui tale pagina è stata portata in memoria
- Se si deve sostituire una pagina, si seleziona quella presente in memoria da più tempo
- Alternativamente, si possono strutturare le pagine presenti nella memoria fisica secondo una coda FIFO e sostituire ogni volta la pagina in testa alla coda



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

157

Algoritmo di sostituzione FIFO (2)

(2)



Esempio

- Consideriamo la seguente stringa dei riferimenti:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- 3 frames

1	1	1	1	4	4	4	1	3	3	9 page faults
2		2	2	2	2	2	2	2	4	
3			3	3	3	5	5	5	5	

- 4 frames

1	1	1	1	1	5	5	5	5	4	4	10 page faults
2		2	2	2	2	1	1	1	1	5	
3			3	3	3	3	2	2	2	2	
4				4	4	4	4	3	3	3	

- **Anomalia di Belady** – più frames \Rightarrow più page faults

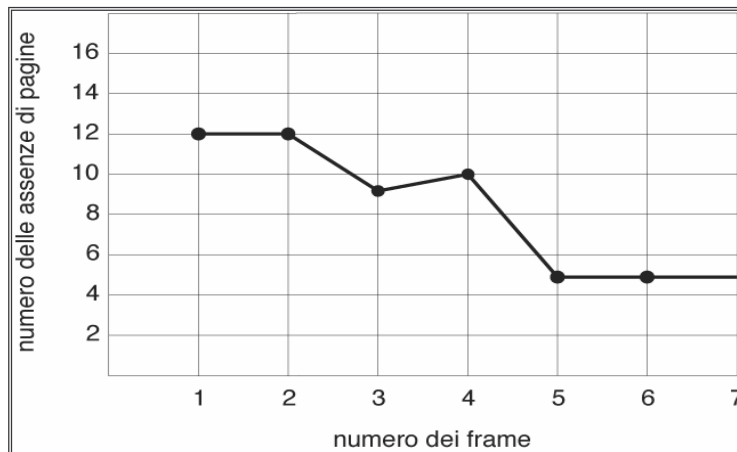


Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

158

Algoritmo di sostituzione FIFO (3)



Algoritmo di sostituzione ottimale (1)



- Sostituisce la pagina che non verrà utilizzata per il periodo di tempo più lungo
- Vantaggi/svantaggi
 - Non presenta l'anomalia di Belady
 - Frequenza di page fault minima
 - Presuppone la conoscenza a priori della stringa dei riferimenti
 - Usato per misurare le prestazioni di altri algoritmi



Algoritmo di sostituzione ottimale (2)



■ Esempio

- Supponiamo di avere a disposizione una memoria fisica con 4 frames
- Consideriamo la seguente stringa di riferimenti

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	1	1	1	1	1	1
2		2	2	2	2	2
3			3	3	3	3
4				4	4	4

6 page faults



Algoritmo di sostituzione LRU (1)



- LRU = Least Recently Used
- Associa ad ogni pagina l'istante in cui è stata usata per l'ultima volta
- Quando si verifica un page fault l'algoritmo LRU sostituisce quella che non è stata usata per il periodo più lungo
- Approssimazione dell'algoritmo ottimale



Algoritmo di sostituzione LRU

(2)



■ Esempio

- Supponiamo di avere a disposizione con 4 frames
- Consideriamo la seguente stringa di riferimenti

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	1	1	1	1
2		2	2	2
3			3	3
4				4

1
2
5
4

1	1	5
2	2	2
5	4	4
3	3	3

8 page faults



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

163

Algoritmi di allocazione dei frames



- Come viene allocata la memoria libera ai vari processi?
 - Ogni processo ha bisogno di un numero *minimo* di frames (definito nell'architettura)
 - Non si possono allocare più frames di quelli disponibili
- Esistono vari algoritmi di allocazione:
 - Allocazione **uniforme**
 - Allocazione **proporzionale**
 - Allocazione **con priorità**
 - Allocazione **globale** e **locale**



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

164

Algoritmi di allocazione

(1)



- **Allocazione uniforme:** se ci sono 100 frames e 5 processi, dai a ogni processo 20 frames
- **Allocazione proporzionale:** alloca i frames in accordo con la taglia dei processi
 - s_i = dimensione del processo p_i
 - $S = \sum s_i$
 - m = numero totale di frames
 - a_i = allocazione per $p_i = (s_i/S) m$

Esempio

$$m = 64$$

$$s_1 = 10$$

$$s_2 = 127$$

$$a_1 = (10/137) 64 \approx 5$$

$$a_2 = (127/137) 64 \approx 59$$



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

165

Algoritmi di allocazione

(2)



Allocazione con **priorità**

- Usa uno schema di allocazione proporzionale con priorità piuttosto che con dimensione
- Se il processo P_i genera un page fault, allora:
 - Seleziona per la sostituzione uno dei suoi frames
 - Seleziona per la sostituzione un frame di un processo con priorità più bassa



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

166

- Allocazione **globale**
 - un processo seleziona un frame per la sostituzione dall'insieme di tutti i frames; un processo può quindi prendere un frame di un altro processo
- Allocazione **locale**
 - ogni processo seleziona un frame per la sostituzione solo dall'insieme di frames ad esso allocati



File System

- Organizzazione
 - Concetto di file
 - Metodi di accesso
 - Struttura delle directory
- Realizzazione
 - Metodi di allocazione
 - Gestione spazio libero

Concetto di file



- I calcolatori possono memorizzare informazioni su diversi supporti di memorizzazione come dischi, nastri, ecc.
- Il **file** è l'unità di memorizzazione logica con cui il SO astrae le caratteristiche fisiche dei dispositivi di memorizzazione fisica
- Un *file* è un insieme di informazioni identificate in modo univoco da un nome e registrate in memoria secondaria
- Le informazioni contenute in un *file* sono definite dal suo creatore e possono essere di vari tipi:
 - *Dati*: numerici, caratteri, binari
 - *Programmi*: sorgente, oggetto
- Dal punto di vista dell'utente il *file* è la più piccola unità di memorizzazione logica su memoria secondaria



Struttura dei file



- Un file ha una struttura definita dal suo creatore e che dipende dal suo tipo
- In generale è una sequenza di bit, byte, righe o *record*
- Un file di *testo* è una sequenza di caratteri organizzati in righe
- Un file *sorgente* è un insieme di procedure e funzioni organizzate in dichiarazioni e istruzioni eseguibili
- Un file *oggetto* è una sequenza di byte comprensibile al collegatore (*linker*)
- Un file *eseguibile* è una sequenza di sezioni di codice in formato rilocabile cioè comprensibile al caricatore (*loader*)



Attributi dei file



- Un file ha molti **attributi** che possono variare a seconda del SO, ma che in generale includono i seguenti
 - **Nome**: unica informazione in forma umanamente leggibile
 - **Identificatore**: etichetta (numero) che identifica univocamente il file nel file system
 - **Tipo**: necessario per sistemi che supportano tipi diversi
 - **Locazione**: puntatore alla locazione del file sul dispositivo
 - **Dimensione**: taglia attuale del file (in byte o blocchi)
 - **Protezione**: informazione di controllo su chi può leggere/scrivere/eseguire un file
 - **Ora, data, e identificazione utente**: relative alla creazione o ultima modifica e sono utili per protezione e monitoraggio dell'utilizzo



Struttura delle directory



- Le informazioni sui file sono mantenute nella **struttura delle directory** che risiede sul disco
- Un elemento di una directory consiste di un nome di file e di un identificatore unico che individua gli altri attributi del file
- Un elemento di una directory può richiedere più di un Kbyte per contenere queste informazioni e quindi in un sistema con molti file la dimensione di una directory può essere molto grande
- Le directory sono registrate nella memoria secondaria e caricate in memoria centrale quando serve



Operazioni sui file



- Per definire adeguatamente la nozione di file bisogna anche specificare le **operazioni** che su di esso si possono eseguire
 - **Creazione**: trova spazio nel FS e crea un nuovo elemento nella directory dove registra il nome del file
 - **Scrittura**: chiamata di sistema che specifica il nome del file e le informazioni da scrivere. Il FS mantiene un puntatore di scrittura alla locazione del file dove eseguire la prossima operazione di scrittura
 - **Lettura**: chiamata di sistema che specifica il nome del file e la locazione della memoria centrale dove collocare le informazioni lette. Il FS mantiene un puntatore di lettura alla locazione del file dove eseguire la prossima operazione di lettura
 - **Riposizionamento**: cerca l'elemento appropriato nella directory e assegna un nuovo valore al puntatore alla posizione corrente nel file
 - **Cancellazione**: cerca il file nella directory, rilascia lo spazio associato al file e lo elimina dalla directory
 - **Troncamento**: consente di cancellare il contenuto di un file mantenendo i suoi attributi



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

173

Metodi di accesso a file



- I file memorizzano informazioni. Al momento dell'uso è necessario accedere a queste informazioni e caricarle in RAM
- Esistono diversi metodi di accesso a file
 - Accesso **sequenziale**
 - Accesso **diretto**
 - Accesso **diretto con indice**
- La scelta del metodo appropriato è un importante problema di progettazione



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

174

Accesso sequenziale



- Le informazioni di un file sono elaborate un record dopo l'altro in maniera ordinata
- È il metodo di accesso più comune ed è utilizzato ad esempio dagli editor e dai compilatori
- Le più comuni operazioni da eseguire sui files sono quelle di lettura e scrittura che vengono eseguite nel meccanismo di accesso sequenziale come segue:
 - *read next*: legge una porzione del file e fa avanzare il puntatore del file che tiene traccia della locazione di I/O
 - *write next*: aggiunge alla fine del file e avanza fino alla fine delle informazioni appena aggiunte (nuova fine del file)



Accesso diretto



- Un file è formato da un insieme di elementi logici di lunghezza fissa chiamati **record**
- Il file è quindi considerato come una sequenza numerata di record che è possibile leggere e scrivere senza un ordine stabilito (Es: leggi il record 10, leggi il record 45 e scrivi il record 23)
- È ispirato al modello di file dei dischi magnetici (dispositivi ad accesso diretto)
- Utili quando si deve accedere velocemente a grandi quantità di informazioni
- Le basi di dati usano un metodo di accesso diretto (quando si deve eseguire una query, bisogna capire quale record contiene la risposta e quindi leggere direttamente quel record)
- Le più comuni operazioni sono *read(n)* e *write(n)*, dove *n* è il numero del record



Accesso diretto con indice (1)



- Sulla base del metodo di accesso diretto se ne possono costruire altri che generalmente prevedono la definizione di un **indice** per un file
- L'indice contiene puntatori ai vari record fisici del file
- Per trovare un elemento logico del file bisogna prima cercare nell'indice e poi usare il puntatore trovato nell'indice per accedere direttamente al file e trovare il record fisico desiderato
- L'obiettivo dell'uso degli indici è quello di compiere ricerche veloci limitando il numero di operazioni di input/output

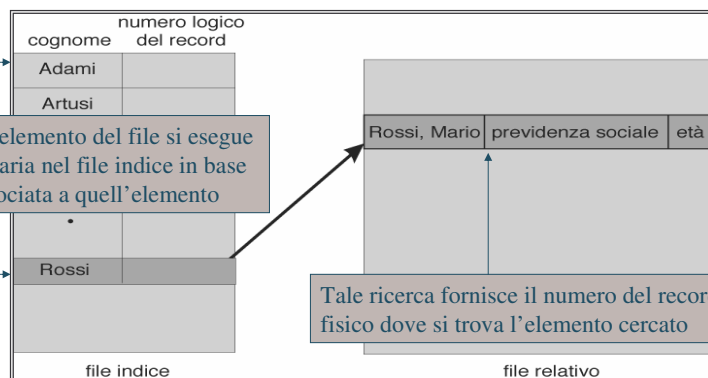


Accesso diretto con indice (2)



Le informazioni del file sono memorizzate nel file indice in ordine alfabetico in base ad una chiave

Per trovare un elemento del file si esegue una ricerca binaria nel file indice in base alla chiave associata a quell'elemento



Tale ricerca fornisce il numero del record fisico dove si trova l'elemento cercato



Struttura del file system



- Il file system di un calcolatore può essere molto ampio (milioni di file memorizzati su dischi)
- Per gestire tutti questi dati è necessario che essi siano **organizzati**
- Tale organizzazione viene in generale effettuata in due parti
 - I dischi vengono suddivisi in un certo numero di **partizioni**. Una partizione è una struttura a basso livello in cui risiedono i file e le directory
 - Ogni partizione contiene le informazioni sui file presenti in essa. Tali informazioni sono memorizzate in una struttura nota come **directory del dispositivo** o **indice del volume**



Informazioni in una directory



- Una directory di dispositivo registra le seguenti informazioni per ogni file nella partizione:
 - Nome
 - Tipo
 - Indirizzo
 - Dimensione attuale
 - Data ultimo accesso
 - Data ultimo aggiornamento
 - Identificativo del proprietario
 - Informazioni di protezione



Operazioni su una directory



- Per poter definire la **struttura logica del sistema di directory** bisogna tener conto delle operazioni che su di essa si possono eseguire:
 - **Ricerca** di un file
 - **Creazione** di un file
 - **Cancellazione** di un file
 - **Elencazione** di una directory
 - **Ridenominazione** di un file
 - **Attraversamento** del file system



Struttura logica delle directory (1)



- Le directory possono essere organizzate a livello logico in vari modi che possono essere valutati in base ai seguenti parametri
 - **Efficienza**: localizzare un file velocemente
 - **Naming**: conveniente per gli utenti
 - Due utenti possono avere lo stesso nome per files diversi
 - Lo stesso file può avere differenti nomi per diversi utenti
 - **Grouping**: raggruppamento logico di files per proprietà, (ad esempio, tutti i programmi Java, tutti i giochi, ...)



Struttura logica delle directory (2)



- Consideriamo i seguenti modi per organizzare le directory a livello logico
 - Directory a **livello singolo**
 - Directory a **due livelli**
 - Directory con **struttura ad albero**
 - Directory con **struttura a grafo aciclico**
 - Directory con **struttura a grafo generale**

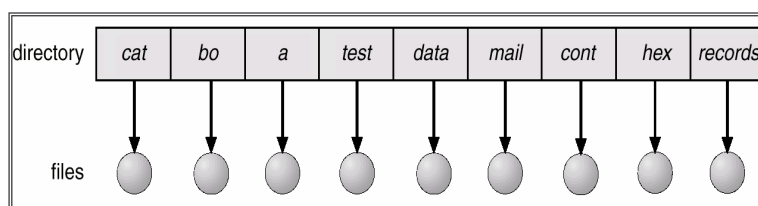


Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

183

Directory a livello singolo



- Tutti i file sono contenuti nella stessa directory
- Presenta limiti notevoli che aumentano all'aumentare del numero di file
 - Problemi di naming (nel caso di tanti file)
 - Problemi di grouping (nel caso di tanti utenti)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

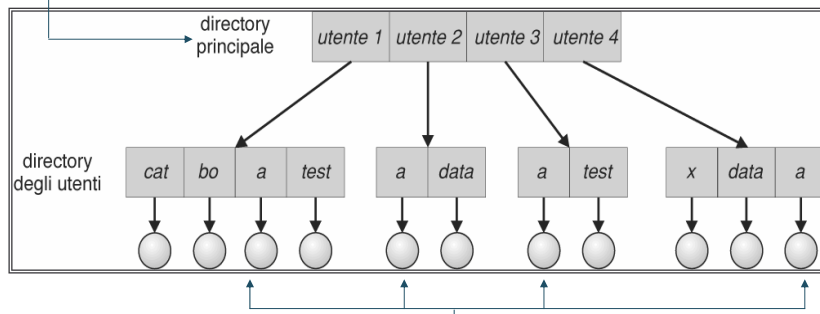
184

Directory a due livelli

(1)



Ogni utente ha la propria directory dei file utente.
Le directory degli utenti sono elencate nella
directory principale (master file directory)



Quando un utente fa riferimento ad un dato file, esso viene cercato nella sua
user file directory. Utenti diversi possono avere file con lo stesso nome



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

185

Directory a due livelli

(2)



■ Vantaggi

- Risolve il problema della collisione dei nomi
- Diversi utenti possono avere file con lo stesso nome
- Si accede ad un file tramite il suo *path name*
- Ricerca più efficiente rispetto alla struttura a livello singolo

■ Svantaggi

- Impossibilità di grouping (e quindi di condivisione di file) in quanto ogni utente è separato dagli altri



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

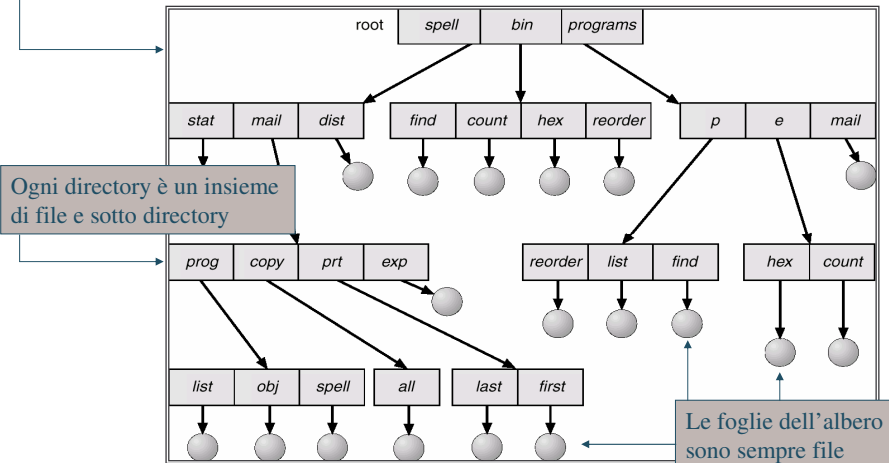
186

Directory con struttura ad albero (1)

(1)



Estensione della struttura a due livelli che permette agli utenti di creare proprie sotto directory e di organizzare in esse i propri file



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

187

Directory con struttura ad albero (2)

(2)



- Ogni utente dispone di una **directory corrente** (directory di lavoro)
- Quando viene fatto un riferimento ad un file si cerca nella directory corrente
- Ogni file ha un path name **assoluto** (parte dalla root) e un path name **relativo** (parte dalla directory corrente)
- La creazione di un nuovo file e la creazione di una nuova subdirectory vengono fatte nella directory corrente
- La cancellazione di un file viene fatta nella directory corrente (se si vuole cancellare un file presente in un'altra directory bisogna specificarne il path name assoluto)
- Se cancelliamo una directory cancelliamo tutto l'albero con radice in quella directory
- Ricerca efficiente
- Diversi utenti possono avere file con lo stesso nome (naming)
- Impossibilità di grouping



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

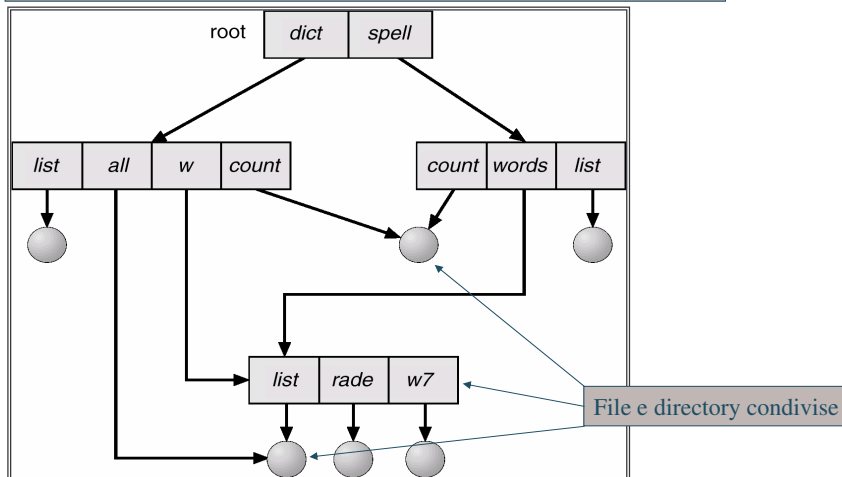
188

Struttura a grafo aciclico

(1)



Estensione della struttura ad albero utile per consentire a programmatori che lavorano ad un progetto comune di condividere directory e file



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

189

Struttura a grafo aciclico

(2)



- Vantaggi
 - Semplicità degli algoritmi di visita e ricerca
 - Utenti diversi possono avere gli stessi nomi di file
 - Possibilità di grouping
- Svantaggi
 - Un file può avere più path name assoluti (*aliasing*) e quindi si deve stare attenti nella cancellazione del file
 - Mantenere aciclica la struttura durante la creazione di nuovi file



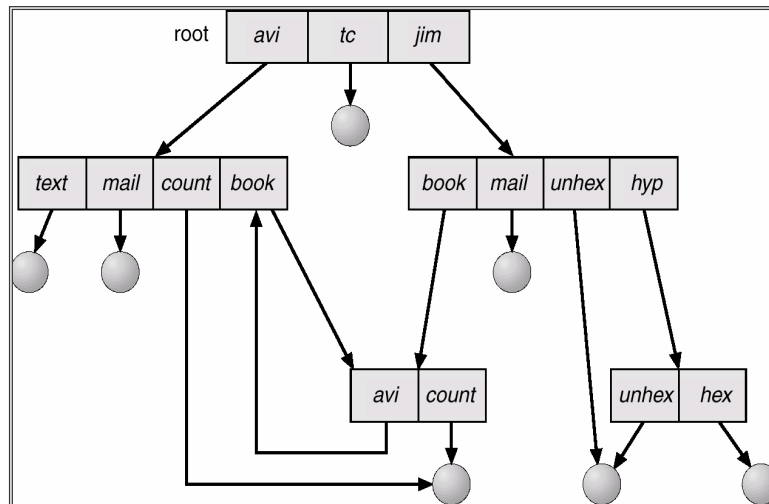
Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

190

Struttura a grafo generale

(1)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

191

Struttura a grafo generale

(2)



- La struttura a grafo aciclico è desiderabile grazie alla semplicità degli algoritmi di visita e ricerca
- Come garantiamo l'assenza di cicli man mano che la struttura del file system cresce/diminuisce?
 - Consenti solo collegamenti a file e non a subdirectories
 - Garbage collection (ripulitura)
 - Ogni volta che un nuovo collegamento viene aggiunto usa un algoritmo di cycle detection per verificare se è il grafo rimane aciclico



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

192

Realizzazione del file system



- Finora abbiamo studiato le caratteristiche che un file system deve avere da un punto di vista logico
- Il file system però risiede in maniera permanente nella memoria secondaria (disco) di un sistema di calcolo
- Risulta pertanto necessario capire come:
 - Viene allocato lo spazio su disco
 - Le componenti del SO si interfacciano con il disco
- I dischi rappresentano la maggior parte della memoria secondaria in cui si conserva il file system ed hanno due caratteristiche importanti
 - Si possono riscrivere localmente
 - Si può accedere direttamente a qualsiasi blocco di informazioni



Realizzazione delle directory



- La scelta degli algoritmi di gestione delle directory ha un impatto decisivo sulle prestazioni e sull'affidabilità di un file system
 - **Lista lineare** di nomi di file
 - Semplice da programmare
 - Ricerca lineare di un file
 - Dispendioso in termini di tempo di esecuzione
 - **Lista lineare ordinata**
 - Ricerca binaria
 - Difficoltà nella creazione e cancellazione di file
 - **Tabella hash**
 - **B-alberi**



Metodi di allocazione



- Un metodo di allocazione riguarda il modo in cui i blocchi su disco sono allocati per i file e deve garantire che:
 - Lo spazio su disco è usato efficientemente
 - L'accesso ai file è rapido
- Esistono tre metodi principali:
 - Allocazione **contigua**
 - Allocazione **concatenata**
 - Allocazione **indicizzata**



Allocazione contigua

(1)

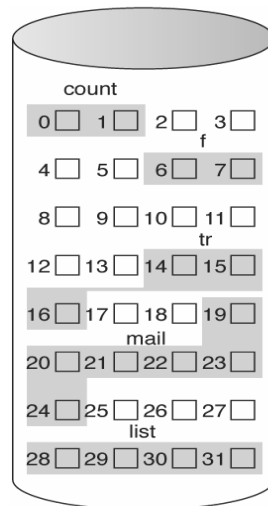


- Ogni file occupa un insieme di blocchi contigui su disco
- Per l'allocazione contigua di un file su disco solo la locazione di partenza (numero del blocco) e la lunghezza (numero di blocchi) sono richiesti
- L'accesso ad un file memorizzato in modo contiguo su disco è facile
 - **Sequenziale**: il file system memorizza l'indirizzo dell'ultimo blocco cui è stato fatto riferimento
 - **Diretto**: nel caso di accesso diretto al blocco i di un file che inizia al blocco b si accede direttamente al blocco $b+i$



Allocazione contigua

(2)



directory		
file	blocco iniziale	lunghezza
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

197

Allocazione contigua

(3)



- L'allocazione contigua presenta alcuni problemi
 - Individuare la disponibilità di spazio libero per un nuovo file
 - Problema di allocazione dinamica della memoria (soddisfare una richiesta di dimensione n data una lista di buchi liberi)
 - Vari criteri di scelta (first-fit, best-fit, worst-fit)
 - I file non possono crescere
 - Frammentazione esterna
 - Necessità di compattamento



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

198

Allocazione concatenata

(1)

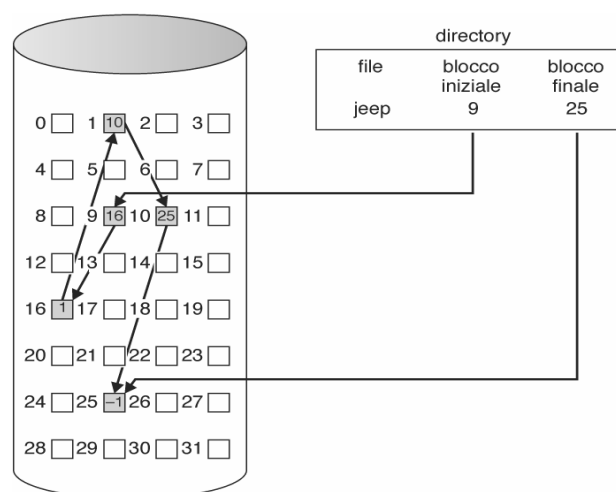


- Ogni file è una lista concatenata di blocchi su disco
- I blocchi possono essere sparsi ovunque sul disco
- Ogni blocco contiene un puntatore al blocco successivo
- La directory che contiene il file ha un puntatore al primo blocco del file stesso
- Risolve tutti i problemi dell'allocazione contigua
- Non esiste frammentazione esterna
- I file possono crescere finché esistono blocchi liberi
- Supporta efficientemente l'accesso sequenziale ma non quello diretto



Allocazione concatenata

(2)

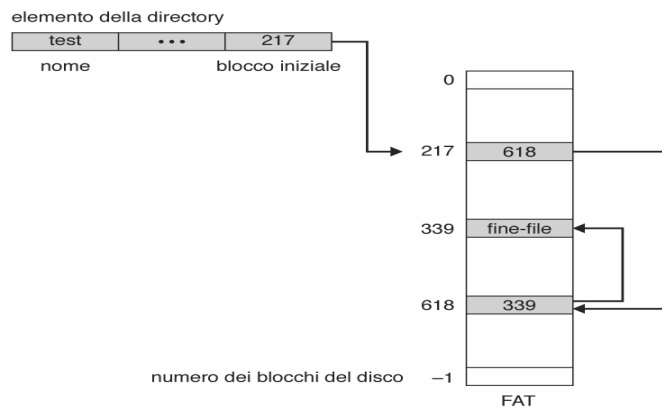


Allocazione concatenata

(3)



- Una variante dell'allocazione concatenata consiste nell'uso della tabella di allocazione dei file (File Allocation Table – FAT)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

201

Allocazione indicizzata

(1)



- Risolve il problema dell'inefficiente accesso diretto dell'allocazione concatenata raggruppando tutti i puntatori ai blocchi di un file in una sola locazione chiamata **blocco indice**
- Ogni file ha il proprio blocco indice che è un vettore di indirizzi a blocchi del disco
- L' i -esimo elemento del blocco indice punta all' i -esimo blocco del file
- La directory contiene il puntatore al blocco indice



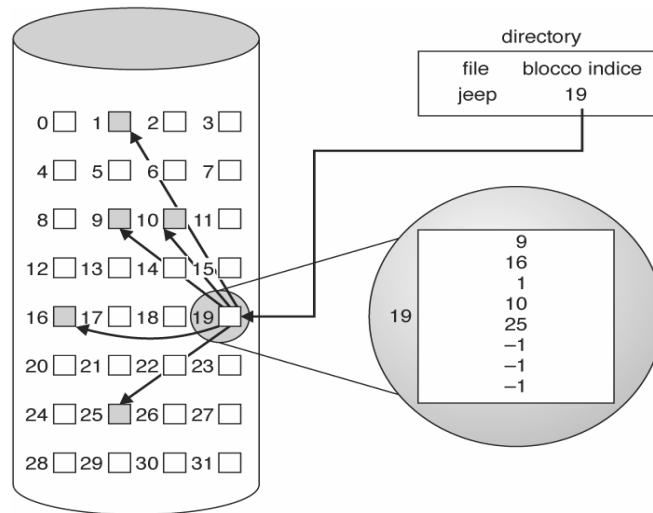
Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

202

Allocazione indicizzata

(2)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

203

Allocazione indicizzata

(3)



- Supporta l'accesso diretto
- Non genera frammentazione esterna ma ha l'overhead del blocco indice (uno per ogni file)
- Decidere la dimensione del blocco indice è di fondamentale importanza e ci sono vari modi:
 - Schema concatenato
 - Indice a più livelli
 - Schema combinato



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

204

Allocazione indicizzata

(4)



- Schema concatenato
 - Il blocco indice è formato da un solo blocco del disco
 - Ciascun blocco indice può essere letto/scritto con un solo accesso a disco
 - Per permettere la presenza di grandi file è possibile collegare tra loro più blocchi indice
- Indice a più livelli
 - Si usa un blocco indice di primo livello che punta a blocchi indice di secondo livello
 - I blocchi di secondo livello puntano ai blocchi dei file
- Schema combinato
 - *inode* in UNIX



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

205

Gestione spazio libero

(1)



- Lo spazio su disco è limitato
- Necessita un efficiente riutilizzo dello spazio lasciato libero dai file cancellati
- Un modo è quello di utilizzare l'**elenco dei blocchi liberi** che tiene traccia dello spazio libero su disco
- Viene modificato ogni volta che viene creato/cancellato un file
- Esistono due modi per implementare l'elenco dei blocchi liberi
 - Vettore di bit
 - Lista concatenata



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

206

Gestione spazio libero

(2)



- Nell'implementazione tramite **vettore di bit** ogni blocco è rappresentato da un bit come segue

$$\text{libero}[i] = \begin{cases} 1 & \Rightarrow \text{block}[i] \text{ è libero} \\ 0 & \Rightarrow \text{block}[i] \text{ è occupato} \end{cases}$$

- Semplice ed efficiente se mantenuto in memoria centrale
- Ciò è possibile solo per dischi piccoli ma non è praticabile per dischi molto grandi



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

207

Gestione spazio libero

(3)



- L'implementazione tramite lista concatenata presenta vantaggi e svantaggi
 - Poco efficiente perché per attraversare la lista occorre leggere ogni blocco e le operazioni di I/O sono costose
 - Si usa in generale il primo blocco libero incontrato nella lista
 - Non c'è spreco di spazio



Daniele Frigioni

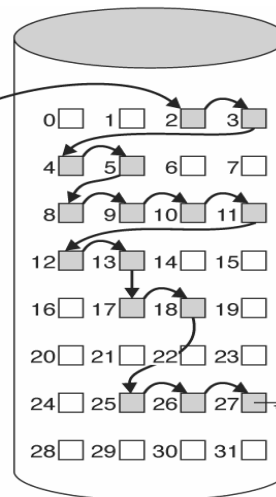
Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

208

Lista dei blocchi liberi su disco



primo elemento della lista
dello spazio libero



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

209

Efficienza e prestazioni



- L'uso efficiente del disco dipende da:
 - Algoritmi usati per l'allocazione dei blocchi del disco
 - Algoritmi usati per la gestione delle directory
- Una volta scelti tali algoritmi, le loro prestazioni possono essere migliorate attraverso meccanismi quali ad esempio il caching dei blocchi usati più frequentemente



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

210



Sistemi di I/O

- Introduzione
- Architetture e dispositivi di I/O
- Interfaccia di I/O per le applicazioni
- Sottosistema di I/O del kernel

Introduzione

(1)



- Il controllo dei dispositivi connessi ad un calcolatore è una delle questioni più importanti che riguardano i progettisti di SO
- Esiste infatti una incredibile varietà di dispositivi di I/O largamente diversi per funzioni e velocità
- I metodi di controllo di tali dispositivi sono quindi altrettanto diversi
- L'insieme di tali metodi costituisce il **sottosistema di I/O** del nucleo



- La tecnologia dei dispositivi di I/O mostra due tendenze tra loro in contrasto:
 - Sempre crescente uniformazione a standard delle interfacce fisiche e logiche
 - Sempre crescente varietà di dispositivi, difficili da integrare in calcolatori e sistemi operativi esistenti
- Questo problema si risolve strutturando il sottosistema di I/O del nucleo in moduli chiamati **driver dei dispositivi** in modo da incapsulare le particolarità e i dettagli dei vari dispositivi
- I driver dei dispositivi offrono al sottosistema di I/O una interfaccia uniforme per l'accesso ai dispositivi allo stesso modo in cui le chiamate di sistema forniscono una interfaccia comune tra le applicazioni e il sistema operativo



- Nonostante l'incredibile varietà dei dispositivi di I/O bastano pochi concetti per capire come tali dispositivi sono connessi al sistema di calcolo e come il sistema operativo li controlla
 - *Porta* (punto di connessione)
 - *Bus* (meccanismo di connessione)
 - *Controllore* (controlla porte, bus o dispositivi)
- I *driver dei dispositivi* offrono al sottosistema di I/O una interfaccia uniforme per l'accesso ai dispositivi



Porta di I/O



- Una porta di I/O è formata tipicamente da quattro registri:
 - **status**: insieme di bit che possono essere letti dalla CPU e indicano lo stato della porta (errore, fine operazione corrente, etc.)
 - **control**: insieme di bit di controllo che può essere scritto dalla CPU per attivare un comando (lettura, scrittura, etc.)
 - **data-in**: letto dalla CPU per ricevere dati
 - **data-out**: scritto dalla CPU per emettere dati



Controllore di dispositivo



- Un controllore è un insieme di componenti elettronici che può far funzionare una porta, un bus o un dispositivo
- La comunicazione tra la CPU e i dispositivi avviene tramite la lettura o la scrittura di uno o più registri che contengono dati e segnali di controllo
- Tale comunicazione può avvenire in due modi:
 - Istruzioni di I/O diretto: specificano il trasferimento di un byte o una parola a un indirizzo di porta di I/O
 - Istruzioni di I/O mappato in memoria: i registri di controllo del dispositivo corrispondono ad un sottoinsieme dello spazio degli indirizzi della CPU



Interrogazione ciclica

(1)



- Il *protocollo* di interazione tra la CPU e un controllore è basato sulla nozione di **negoziiazione** (*handshaking*) che può essere visto come un meccanismo Produttore/Consumatore come segue:
 - Il controllore specifica il suo stato tramite il bit *busy* del registro *status* (1 = occupato, 0 = libero)
 - La CPU comunica le sue richieste tramite il bit *command-ready* del registro *control* (viene posto a 1 quando un controllore deve eseguire un comando)



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

217

Interrogazione ciclica

(2)



- Esempio di negoziazione per la scrittura su una porta da parte della CPU:
 - La CPU legge ripetutamente il bit *busy* finché esso non assume il valore 0 (*attesa attiva* o *interrogazione ciclica*)
 - La CPU pone a 1 il bit *write* del registro *control* e scrive un byte nel registro *data-out*
 - La CPU pone a 1 il bit *command-ready*
 - Quando il controllore si accorge che il bit *command-ready* vale 1 pone a 1 il bit *busy*
 - Il controllore legge il registro *control*, trova il comando *write* ed esegue l'operazione di scrittura nel dispositivo leggendo dal registro *data-out*
 - Il controllore pone a 0 il bit *command-ready*, pone a 0 il bit *error* nel registro *status* per segnalare che l'I/O ha avuto esito positivo e pone a 0 il bit *busy* per indicare che l'operazione è terminata
- La sequenza descritta si ripete per ogni byte



Daniele Frigioni

Architetture dei Calcolatori e Sistemi Operativi - Master in Tecnologie, Applicazioni e Servizi in Reti Radio Eterogenee

218

Interruzioni

(1)

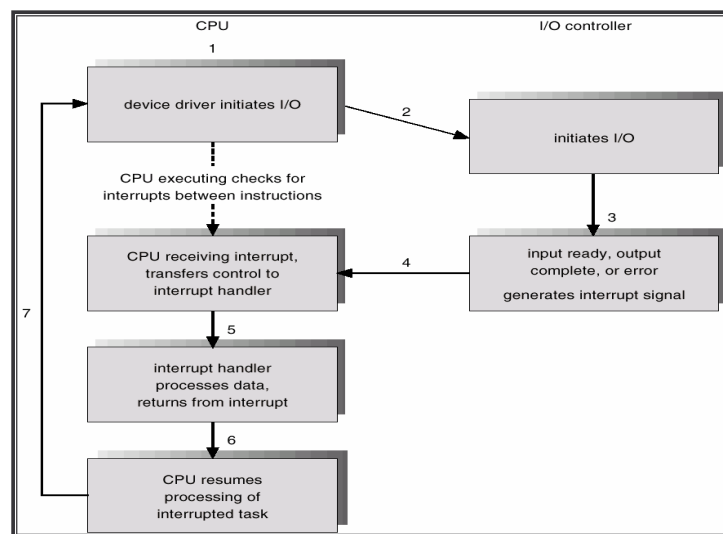


- L'interrogazione ciclica può diventare inefficiente se il dispositivo viene trovato pronto raramente consumando inutilmente cicli di CPU
- Risulta pertanto più efficiente far sì che il controllore del dispositivo possa segnalare alla CPU quando il dispositivo stesso è pronto per una operazione di I/O attraverso un segnale di **interruzione**
- La CPU ha un contatto detto *linea di richiesta di interruzione* del quale controlla lo stato dopo l'esecuzione di ogni istruzione
- Quando viene rilevata una interruzione viene passato il controllo alla routine di gestione delle interruzioni che gestisce l'interruzione e ritorna il controllo alla CPU
- Il vettore delle interruzioni contiene gli indirizzi di memoria degli specifici gestori delle interruzioni
- Il sistema di *livelli di priorità* permette alla CPU di gestire le interruzioni in base alla loro priorità



Interruzioni

(2)



Interfaccia di I/O per applicazioni (1)



- Le interfacce di I/O di un sistema operativo permettono un trattamento uniforme dei dispositivi di I/O. In particolare:
 - Consentono ad una applicazione di aprire un file residente su disco senza sapere di che disco si tratti
 - Consentono di aggiungere al calcolatore nuove unità disco o altri dispositivi senza che si debba modificare il sistema operativo
- Le differenze tra i vari dispositivi sono incapsulate in moduli del nucleo detti **driver dei dispositivi** che sono specializzati per dispositivi specifici ma comunicano con l'esterno per mezzo delle interfacce uniformi



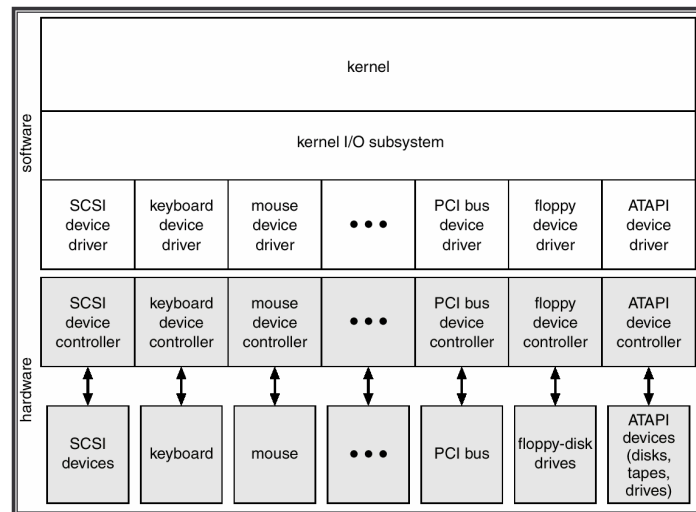
Interfaccia di I/O per applicazioni (2)



- I dispositivi possono differire in molti aspetti:
 - Trasferimento a flusso di caratteri o a blocchi
 - Accesso sequenziale o diretto
 - Dispositivi sincroni o asincroni (tempi di risposta prevedibili o non prevedibili)
 - Dispositivi condivisibili o riservati (uso concorrente o riservato)
 - Velocità di funzionamento
 - Lettura e scrittura, sola lettura o sola scrittura



Struttura dell'I/O di un kernel



Sottosistema di I/O del nucleo



- Il sottosistema di I/O del nucleo fornisce molti servizi riguardanti l'I/O:
 - Scheduling
 - Buffering (memorizzazione transitoria)
 - Caching
 - Spooling (gestione delle code)
 - Gestione degli errori
- Sono realizzati a partire dai dispositivi e dai relativi driver



Scheduling di I/O



- Meccanismo che stabilisce l'ordine di esecuzione più efficace delle richieste di I/O
 - L'ordine di arrivo delle richieste non è in generale la scelta migliore
 - L'accesso dei processi ai dispositivi di I/O va distribuito equamente in modo da ridurre il tempo medio di attesa per il completamento di una operazione di I/O
 - Uso di code di richieste per ogni dispositivo
 - Uso di tecniche eque di servizio delle richieste
- Lo scheduling del I/O è uno dei meccanismi con cui il sottosistema di I/O migliora l'efficienza del sistema



Buffering di I/O



- Meccanismo di memorizzazione temporanea di dati in un area di memoria (*buffer*) durante il trasferimento tra due dispositivi o tra una applicazione e un dispositivo
- Viene usato per tre ragioni principali
 - Gestire la diversa velocità dei dispositivi
 - Gestire dispositivi che trasferiscono dati in blocchi di dimensioni diverse (tipico delle reti di comunicazione dove è spesso necessario frammentare e ricomporre messaggi)
 - Realizzare la **semantica delle copie** nell'ambito del I/O delle applicazioni per garantire la coerenza delle copie degli stessi dati presenti ai vari livelli della gerarchia delle memorie



Caching e spooling di I/O



- *Caching* – memorizzazione su supporti veloci di *copie* di dati importanti
 - Distinta dalla memorizzazione transitoria
 - Fondamentale per le prestazioni
- *Spooling* – memorizzazione di transito contenente dati per un dispositivo che non può accettare flussi di dati intercalati
 - Si usa quando il dispositivo può servire al più una richiesta alla volta (stampante)
 - Alcuni SO concedono l'uso esclusivo di un dispositivo e usano le chiamate di sistema per l'allocazione e la deallocazione gestendo situazioni di deadlock



Efficienza e prestazioni



- L'I/O è uno dei principali fattori che influiscono sulle prestazioni di un sistema
 - Richiede tempo di CPU per eseguire il codice dei driver dei dispositivi, e per realizzare uno scheduling equo delle richieste di I/O
 - I conseguenti cambi di contesto sfruttano a fondo la CPU e la sua memoria cache
 - I trasferimenti di dati tra i controllori dei dispositivi e la memoria impiegano la CPU e il bus
 - Il traffico di rete può risultare estremamente pesante dal punto di vista del tempo di CPU
- Soddisfare queste esigenze in modo efficiente è una delle principali questioni che riguardano i progettisti di sistemi operativi

