

# 1 Introduzione 1

- Programma di sistema: gestisce le operazioni del computer.
- Programma applicativo: eseguono i compiti impartiti dall'utente.
- Sistema operativo: il più importante programma di sistema che gira in Kernel mode. Ha il compito di dare all'utente un insieme più appropriato di istruzioni con cui lavorare.
- Macchina virtuale: si interpone uno strato software a quello hardware per nascondere all'utente tutte le peculiarità e problematiche tecniche, i dettagli relativi all'hardware e fornisce un'immagine comoda, elegante e consistente.
- Microprogramma: software, solitamente residente nella ROM, che gestisce direttamente i dispositivi. Al boot del sistema carica le istruzioni base quali ad es. ADD, MOVE, JUMP etc.
- Ciclo di fetch-execute: è composto da i seguenti passi:
  - I. Caricare dalla memoria la prossima istruzione che deve essere eseguita;
  - II. Decodificare l'opcode;
  - III. Leggere gli operandi dalla memoria principale;
  - IV. Eseguire l'istruzione e salvare i risultati;
  - V. Ripartire dal passo 1;
- Linguaggio macchina: insieme delle istruzioni interpretate dal microprogramma.
- RISC (Reduced Instruction Set Computers): non hanno un alto livello di microprogrammazione e le istruzioni in linguaggio macchina sono eseguite direttamente dall'hardware.

•

## 1.1 Cos'è un sistema operativo 3

- S.O. come macchina estesa o virtuale: nasconde all'utente tutte le peculiarità e problematiche tecniche, i dettagli relativi all'hardware e fornisce un'immagine comoda, elegante e consistente.
- S.O. come gestore di risorse hardware/software del sistema: tiene conto di chi sta utilizzando una risorsa e quale, di fornire diritti d'accesso, di registrarne l'uso e risolvere problemi di richieste conflittuali tra programmi ed utenti, anche diversi e numerosi.

## 1.2 Storia dei sistemi operativi 5

- Prima generazione (1945-1955) valvole e pannelli di collegamento.
- Seconda generazione (1955-1965) transistor e sistemi batch:
  - ✓ Job: programma o insieme di programmi. La struttura è:
    - I. \$JOB = contiene il massimo tempo di calcolo espresso in minuti, il numero dell'account sul quale far girare il lavoro, nonché informazioni relative al programmatore;
    - II. \$FORTRAN = indica al S.O. di caricare il compilatore Fortran;
    - III. Programma in Fortran;
    - IV. \$LOAD = indica al S.O. di caricare il codice oggetto del programma appena compilato;
    - V. \$RUN = indica al S.O. di mandare in esecuzione il programma con i dati successivi;
    - VI. Dati del programma;
    - VII. \$END = segnala la fine del JOB;
  - ✓ Sistema batch: formato da vari job che contengono vari programmi. Si ha un computer specializzato per l'I/O (l'IBM 1404) ed uno per i calcoli (l'IBM 7094). I S.O. tipici sono FMS (Fortran Monitor System) e l'IBSYS.
- Terza generazione (1965-1980) circuiti integrati e multiprogrammazione:
  - ✓ IBM System/360: linea di macchina software compatibili. La debolezza coincideva con il suo punto di forza, ossia l'idea che una grande singola famiglia potesse servire per ogni tipo di compito. Il S.O. (OS/360) è enorme, complesso e pieno di bug.
    - I. Multiprogrammazione: introdotta da OS/360, è la possibilità di passare da un processo ad un altro tramite la suddivisione della memoria in partizioni assegnate ognuna ad un processo. In questo modo si utilizza il tempo di non utilizzo della CPU quando si hanno operazioni di I/O.
    - II. Spooling (Simultaneous Peripheral Operation On Line): si interpone un software, il demone, che è il solo ad avere accesso diretto alla risorsa;
  - ✓ Timesharing: variante della multiprogrammazione che permette di gestire più job contemporaneamente condividendo il tempo di utilizzo della CPU. Si ha quindi una gestione on-

line delle richieste.

✓ **MULTICS** (Multiplexed Information and Computing Service): sviluppata da MIT, Bell Labs e General Electric, è una computer utility, ossia una macchina capace di supportare contemporaneamente centinaia di utenti in timesharing, così come avviene per la distribuzione dell'energia elettrica.

✓ **UNIX**: S.O. sviluppato da Thompson a seguito dell'esperienza avuta con MULTICS.

✓ **POSIX**: standard per UNIX che definisce un'interfaccia minima di sistema.

• Quarta generazione (1980-oggi) i personal computer:

✓ **Workstation**

✓ **User-friendly**: software di semplice utilizzo anche per utenti inesperti.

✓ **S.O. di rete**: ogni macchina possiede localmente il proprio S.O. che permette di collegarsi ad altre macchine remote.

✓ **S.O. distribuito**: è un sistema multiprocessore nel quale l'utente non è a conoscenza di dove sono conservati i file e dove girano i programmi.

• **Storia di MINIX.**

### 1.3 Nozioni sui sistemi operativi 15

• **System calls o Chiamate di sistema**: è il set di istruzioni estese fornite dal S.O. per interfacciarlo con i programmi utente.

• **Processo**: vedi 2.1.

• **Process table o Tabella dei processi**: vedi 2.1.

• **Trap**: eccezioni software intercettate dall'hardware.

• **UID**: identificatore di utente.

• **File system**: vedi 5.

• **File speciali**: consentono di trattare dispositivi di I/O come file:

I.a blocchi = I/O a blocchi (ad es. dischi);

II.a caratteri = I/O a caratteri (ad es. stampanti);

• **Pipe**: pseudo-file per collegare due processi.

• **Shell o interprete dei comandi**: interfaccia tra l'utente ed il S.O.; è un processo.

### 1.5 Struttura di un sistema operativo 37

• **Sistemi monolitici**: il S.O. è organizzato in maniera omogenea e non si possono distinguere le varie componenti; può essere sottolineato "la grande confusione" perché, praticamente, non ha struttura ed ogni procedura è visibile da tutte le altre.

✓ **Kernel o Supervisor mode**: modalità che può gestire gli interrupt e può accedere alle tabelle di sistema e all'hardware.

✓ **Kernel o Supervisor call**: trap che permette di passare alla modalità kernel.

• **Sistemi a strati**: S.O. a vari livelli, dove i più bassi nascondono le peculiarità a quelli superiori; realizza a pieno l'idea di S.O. come macchina estesa. La struttura del THE, il primo S.O. a strati, è:

Strato	Funzionalità	
5	Operatore	User Mode
4	Programmi utente	
3	Gestione I/O	Kernel Mode
2	Comunicazione processo-console	
1	Gestione memoria	
0	Allocazione CPU e multiprogrammazione	

• **Macchine virtuali**: dà l'impressione di lavorare su una macchina che in realtà non c'è; sull'hardware si monta un S.O. che permette di emulare varie macchine virtuali. Come es. si ha il VM/370, che permetteva di gestire varie versioni di se stesso.

✓ **Virtual Machine Monitor**: cuore del sistema, gira direttamente sull'hardware, realizza la multiprogrammazione e fornisce al livello superiore varie macchine virtuali. Queste sono copia esatta dell'hardware, e quindi si possono montare sopra anche diversi S.O.

✓ **CMS (Conversational Monitor System)**: sistema interattivo monoutente per utenti in timesharing. Ogni chiamata all'hardware, da parte del S.O., passa per il CMS. Tutte le risorse vengono

condivise da tutti i sistemi e vengono gestite dall'hardware e dal VM/370.

✓Exokernel: programma in esecuzione in modalità nucleo che alloca risorse per le macchine virtuali.

- Modello client-server: la tendenza è quella di snellire il kernel, implementando funzionalità del S.O. all'interno di programmi utente. Si ha quindi il rapporto cliente-servitore. Si adatta molto bene a sistemi distribuiti.

## 2 Processi 49

### 2.1 Introduzione ai processi 49

- Pseudo-parallelismo: è l'idea che indica la rapida commutazione della CPU da un programma all'altro (VS multiprocessore). Si ha una situazione nella quale tutti i processi progrediscono anche se solo uno alla volta sia effettivamente in esecuzione; non c'è quindi bisogno che i processi siano programmati tenendo conto di assunzioni temporali.
- Processo: entità dinamica, programma in esecuzione al quale è associato un insieme di registri che contiene il program counter, il puntatore allo stack, altri registri hardware ed altre informazioni necessarie all'esecuzione del programma. Inoltre vi è uno spazio d'indirizzamento, cioè una lista di locazioni di memoria, tra un minimo ed un massimo, che contiene il codice del programma, i dati statici, quelli dinamici e lo stack. Concettualmente ogni processo ha la sua CPU virtuale.
- Programma: algoritmo espresso in qualche linguaggio; è quindi un'entità statica.
- Stati di un processo: le transizioni possibili sono: I-III, I-II, II-I, III-II:
  - I. Esecuzione o running = il processo sta effettivamente utilizzando la CPU;
  - II. Pronto o ready = il processo è eseguibile ma temporaneamente sospeso per consentire l'esecuzione di un altro processo;
  - III. Bloccato = il processo è impossibilitato a proseguire fino al verificarsi di un evento esterno;
- Scheduler: vedi 2.4.
- Process table o Tabella dei processi: array o lista di strutture, una per ogni processo, che contiene le informazioni relative al processo, ad eccezione del contenuto dello spazio d'indirizzamento. In particolare ha: il program counter, il puntatore allo stack, l'allocazione di memoria, lo stato dei file aperti, il tempo d'utilizzo della CPU e le informazioni di scheduling.
- Vettore di interruzione: locazione di memoria, posizionata in fondo alla memoria, al quale è associata una classe di dispositivi di I/O.
- Context switch o cambio di contesto: quando si deve passare da un processo ad un altro si salva il program counter e il puntatore alla tabella dei processi in variabili globali (in modo tale da poter essere reperiti velocemente), nonché registri e variabili.

### 2.2 Comunicazioni interprocesso 59

- Corse critiche: quando due o più processi leggono o scrivono dati condivisi e il risultato finale dipende da chi va in esecuzione e quando.
- Sezioni o regioni critiche: parte di codice del programma in cui si accede alla memoria condivisa.
  - ✓Mutua esclusione: se un processo usa una risorsa condivisa, gli altri processi non la possono utilizzare in quel momento.
  - ✓Condizioni per le corse critiche: per evitare si presuppone il contemporaneo verificarsi di:
    - I. Due processi non possono contemporaneamente essere nella loro sezione critica;
    - II. Non si possono fare ipotesi sulla velocità dei processi e sulla loro durata;
    - III. Nessun processo in esecuzione al di fuori della sua sezione critica può bloccare altri processi;
    - IV. Nessun processo deve attendere un tempo lungo prima di entrare in sezione critica;
- Mutua esclusione con attesa attiva: vedi foglio:
  - I. Interruzioni disabilitate;
  - II. Variabili di lock;
  - III. Alternanza stretta;
  - IV. Soluzione di Peterson;
  - V. Istruzione TSL;
- Sleep e Wakeup: vedi foglio.

✓ Problema del produttore-consumatore o buffer limitato:

I. Buffer pieno: Produttore = sleep al produttore; Consumatore = wakeup al produttore;

II. Buffer vuoto: produttore = wakeup al consumatore; Consumatore = sleep al consumatore;

• Semafori: vedi foglio.

✓ Problema del produttore-consumatore o buffer limitato = 2 semafori (full ed empty) per sincronizzare, 1 semaforo binario (mutex) per mutua esclusione.

• Monitor: vedi foglio.

• Scambio messaggi: vedi foglio.

✓ Problema del produttore-consumatore o buffer limitato: XXX

I. Indirizzo unico ad ogni processo per i messaggi;

II. Mailbox, send e receive; i messaggi sono in numero fissato e il produttore ne manda uno pieno mentre il consumatore uno vuoto;

## 2.3 Problemi classici di IPC 78

• Problemi IPC: comunicazione interprocesso.

• Problema dei filosofi affamati:

I. Filosofo prende le due forchette una alla volta ? si ha starvation, ossia i programmi procedono senza termine e senza ottenere alcun progresso;

II. Filosofo prende le due forchette una alla volta e se non può mangiare le riposa e le riprende dopo un tempo random ? si ha ancora starvation;

III. Semaforo binario che indica chi sta mangiando ? un solo filosofo alla volta può mangiare;

IV. Array di stati che registra se un filosofo sta mangiando, pensando o cerca le forchette;

• Problema del barbiere addormentato: 2 semafori (customers e barber) per i clienti in attesa e per i barbieri fermi, 1 semaforo (mutex) per la mutua esclusione ed 1 variabile (waiting) che è la copia di customers.

## 2.4 Scheduling dei processi 85

• Scheduler: parte del S.O. che si occupa del management dei processi; è molto snello e semplice. Deve essere:

I. Equo;

II. Efficiente;

III. Minimizzare il tempo di risposta;

IV. Minimizzare il tempo d'attesa da parte degli utenti;

V. Massimizzare il throughput, ossia il numero di job elaborati ogni ora;

ma alcuni punti vanno in contrasto tra loro, Si può dimostrare che uno scheduler che favorisce una classe di job lo fa a spese di altri. Si possono adottare due strategie:

I. Preemptive = sospendere temporaneamente i processi che sono logicamente eseguibili;

II. Nonpreemptive o Run to completion = eseguire i processi fino al loro completamento;

• Algoritmo di scheduling: algoritmo utilizzato dallo scheduler per scegliere tra i processi ready.

I. Round robin: ogni processo ha un quanto di tempo e non si hanno priorità diverse tra processi. Tutto si gestisce con una lista circolare di tipo FIFO. Il problema si ha nella decisione della durata del quanto;

II. Prioritario: i processi hanno una priorità o sono raggruppati in classi di priorità; dopo l'esecuzione la priorità del processo viene diminuita;

III. Code multiple: si ha lo spostamento in basso ogni volta che un processo viene eseguito e consuma per intero il suo quanto. La classe più alta (la 1°) ha 1 quanto, mentre quelle di numero maggiore hanno più quanti;

IV. Shortest job first: si esegue il processo il cui tempo d'esecuzione è il più piccolo. Si può applicare se si conosce prima tale tempo; in alternativa si può fare una stima tramite aging (tecniche di invecchiamento);

## 3 Input/Output 159

### 3.1 Principi dell'hardware di I/O 159

• Dispositivi di I/O: sono grossolanamente divisi in:

I.a blocchi = ad es. dischi;

II.a caratteri = ad es. stampanti;

• Controllori di dispositivo: le unità o dispositivi di I/O sono costituiti da parti:

I. Elettronica ? controllore del dispositivo o adattatore = scheda che si inserisce in un backplane della parentboard e con il quale il S.O. si interfaccia. Il suo compito è anche quello di effettuare l'eventuale correzione d'errore e di convertire l'output del dispositivo nel formato riconoscibile dal sistema;

II. Meccanica ? dispositivo vero e proprio;

• Interrupt o interruzione: è un segnale elettrico che serve ad avvertire che si è verificato un certo evento.

• DMA (Direct Memory Access): la CPU fornisce al controllore l'indirizzo sul disco del blocco, quanti blocchi copiare e dove; successivamente la CPU può continuare a fare i calcoli senza curarsi del trasferimento. Il controllore possiede un buffer interno, per poter conservare i bit che arrivano dal dispositivo a ritmo costante. Per sopperire alla differenza di velocità tra il trasferimento e la rotazione del disco si utilizzano tecniche di interleaving, ossia i blocchi sono memorizzati sul disco a saltare.

### 3.2 Principi del software di I/O 165

• Scopo del software di I/O: si vuole rendere indipendente il software dal dispositivo, dalle sue peculiarità tecniche e progettuali; si cerca inoltre di avere un'uniformità dello spazio dei nomi. Per fare questo si struttura il software di I/O in:

I. Gestore delle interruzioni: serve l'interrupt tramite l'interrupt handler;

II. Device driver o gestore di dispositivo o controllore: impartisce i comandi e vede se sono corretti, imposta i registri del dispositivo, conosce a fondo l'hardware;

III. Software di I/O indipendente dal dispositivo: fornisce:

- 1) Interfaccia uniforme;
- 2) Gestione della protezione;
- 3) Gestione di dimensioni di settore diverse tra dischi diversi;
- 4) Allocazione e rilascio dei dispositivi;
- 5) Gestione degli errori;
- 6) Gestione del buffer;

IV. Software di I/O nello spazio utente: effettua le chiamate di I/O, definisce il formato dell'I/O, effettua lo spooling;

### 3.3 Deadlock 173

• Deadlock: stallo con risorse hardware o software (vedi "Principi del deadlock").

• Risorsa: un qualsiasi oggetto allocato che può essere utilizzato da un solo processo alla volta. Si dividono in:

I. Con prerilascio = può essere sottratta al processo senza causare errori (ad es. Cd-Rom);

II. Senza prerilascio = non può essere sottratta al processo, altrimenti causa errori (ad es. Masterizzatore);

• Principi del deadlock: un insieme di processi è in deadlock se ogni processo dell'insieme è in attesa di un evento che solo un altro processo appartenente allo stesso insieme può causare; in altre parole, ogni membro dell'insieme di processi in deadlock è in attesa di una risorsa posseduta da un altro processo in deadlock.

• Condizioni: si devono verificare contemporaneamente:

I. Mutua esclusione = ogni risorsa o è libera o è allocata a un solo processo;

II. Hold and wait o prendi ed aspetta = i processi che hanno risorse allocate possono richiederne altre;

III. Assenza di prerilascio = un processo non può essere forzato a rilasciare una risorsa;

IV. Attesa circolare = in una lista di processi ognuno è in attesa di una risorsa che è allocata al processo che segue nella lista;

• Scoprire i deadlock: se si verifica un ciclo nel grafo delle risorse, cioè un grafo orientato con:

I. Cerchi = processi;

II. Quadrati = risorse;

III. Arco da un quadrato ad un cerchio = risorsa richiesta ed assegnata al processo;

- IV. Arco da un cerchio ad un quadrato = il processo è bloccato in attesa della risorsa;
- Algoritmo dello struzzo: ignorare il problema visto che non si verificano così spesso rispetto ad altri problemi.
  - Individuare e risolvere: il sistema controlla se c'è un ciclo; se sì si termina un processo del ciclo fino a che non si elimina il deadlock.
  - Prevenzione: vincolare i deadlock in modo da renderli strutturalmente impossibili:
    - I. Mutua esclusione ? va bene per stampanti e spooling, ma non è generale;
    - II. Richiedere tutte le risorse prima ? i processi non sanno esattamente a priori quante risorse hanno di bisogno;
    - III. Prerilascio ? non funziona con risorse senza prerilascio, come ad es. stampanti;
    - IV. Eliminare l'attesa circolare ? numerare le risorse, ma può non esistere un ordinamento soddisfacente;
  - Evitare di deadlock: si cercano algoritmi che evitino sempre i deadlock facendo le scelte giuste:
    - I. Algoritmo del banchiere per risorse singole = considerare ogni richiesta nell'istante in cui si presenta e controllare che il suo soddisfacimento porti a uno stato sicuro (se esiste una sequenza di altri stati che porta tutti i processi ad ottenere le proprie risorse e terminare). Per ogni processo si ha il numero di risorse allocate e quello massimo allocabile;
    - II. Traiettorie di risorse = se si hanno più classi di risorse; con un solo processore gli spostamenti non sono mai diagonali e comunque sempre verso sopra e destra;
    - III. Algoritmo del banchiere per risorse multiple = si realizza con più matrici;

### 3.7 Dischi 207

- Hardware dei dischi: sono organizzati in cilindri, ognuno contiene tante tracce quante sono le testine allineate in verticale. Le tracce sono a loro volta divise in settori che contengono un certo numero di byte. Più ci si avvicina al bordo esterno più i settori si allungano e quindi la densità di dati è maggiore nei cilindri più interni, anche se il tempo di lettura/scrittura rimane sempre lo stesso.
  - ✓ IDE (Integrated Drive Electronics): contengono più settori nelle tracce esterne ed è compito dell'elettronica sofisticata mascherare queste differenze e rendere il sistema omogeneo.
  - ✓ Overlapped seeks o posizionamenti sovrapposti: il controllore può effettuare posizionamenti su più drive contemporaneamente.
- Software del disco: i dischi sono soggetti ad errori, quindi ci deve essere un qualche tipo di controllo d'errore. Il tempo per leggere un dato è la somma di:
  - I. Posizionamento del braccio che sostiene la testina sul cilindro (il maggiore);
  - II. Rotazione del disco per fare trovare allineati testina e settore;
  - III. Trasferimento dati;

## 4 Gestione della memoria 325

- Gestore della memoria: parte del S.O. che gestisce la gerarchia di memoria. Si occupa di allocare e deallocare memoria, di sapere quanta e quale memoria è allocata ad un processo e di effettuare lo swapping.

### 4.1 Concetti base di gestione della memoria 325

- Monoprogrammazione senza swapping o paginazione: si ha un solo processo alla volta in esecuzione. Il S.O. può risiedere in fondo alla RAM, in cima in una ROM oppure nella ROM o BIOS (Basic Input Output System) si hanno i device driver, mentre in fondo alla RAM si ha il S.O.
- Multiprogrammazione con partizioni fisse: si hanno più processi in timesharing e le dimensioni delle partizioni sono diverse e fisse, causando quindi frammentazione interna:
  - I. Partizioni con code d'ingresso multiple = ogni nuovo job è inserito nella coda d'ingresso per la partizione più piccola in grado di contenerlo;
  - II. Partizioni con coda d'ingresso singola = ogni volta che si libera una partizione il job più vicino alla testa della coda, e di dimensioni minore o uguale a quella della partizione, viene caricato nella partizione; questa strategia è a sfavore dei job più piccoli;
  - III. Rilocazione: quando il linker collega le varie parti del programma dovrebbe sapere a quale indirizzo avrà inizio il programma. Questo valore però cambia sempre; calcola quindi degli

indirizzi relativi, interni al codice del programma, che verranno poi modificati dal S.O. tramite:

1) Registro base = indirizzo iniziale della partizione;

2) Registro limite = indirizzo finale della partizione;

IV. Protezione: un programma non deve poter accedere ad aree di memoria che non siano le sue;

#### 4.2 Swapping 329

- Swapping: strategia che consente di caricare un processo per intero, eseguirlo per un po' per poi spostarlo sul disco. In questo caso il numero e la grandezza delle partizioni varia nel tempo.
  - I. Processi con dimensione fissa (non crescono nel tempo): si alloca esattamente lo spazio necessario a contenere il processo;
  - II. Processi con dimensione variabile (crescono nel tempo permettendo l'allocazione dinamica della memoria): ogni volta che un processo richiede più memoria di quanta non sia contenuta nella partizione bisogna cercarne una compatibile e spostare il processo oppure fare lo swapping;
- Gestione della memoria con le bit map: la memoria viene divisa in unità di allocazione al qual è associato un bit che indica se l'unità è libera o occupata. Una memoria di  $32n$  bit richiederà soltanto un numero fisso di  $n$  bit per la bit map. Lo svantaggio si ha nella ricerca (sequenziale) di una sequenza di blocchi liberi capaci di contenere i dati da caricare;
- Gestione della memoria con liste: ogni elemento della lista, eventualmente circolare, è composto da un campo che indica se quello è uno spazio vuoto (H) oppure un processo (P), l'indirizzo d'inizio, la lunghezza ed il puntatore al successivo. Si hanno vari algoritmi per l'allocazione della memoria:
  - I. First fit = si alloca uno spazio vuoto grande a sufficienza;
  - II. Next fit = come il first fit solo che si tiene conto della posizione dell'ultimo spazio libero trovato;
  - III. Best fit = scandisce tutta la lista ed alloca lo spazio vuoto più piccolo per il processo; è lento ed ha frammentazione esterna;
  - IV. Worst fit = scandisce tutta la lista ed alloca lo spazio vuoto più grande per il processo, in modo da crearne uno nuovo non troppo piccolo; è lento;
  - V. Quick fit = gestisce un certo numero di liste separate per dimensioni di allocazione più frequentemente usate; è veloce nell'allocazione ma oneroso nella deallocazione nel cercare i vicini per effettuare un compattamento;
- Gestione della memoria con buddy systems: il gestore della memoria mantiene una lista di blocchi di dimensioni pari alle potenze di due, fino alla dimensione massima della memoria. All'inizio si ha un'unica lista che contiene tutta la memoria; alla prima richiesta la memoria viene divisa in due finché non si trova la più piccola potenza di due sufficiente a contenere il blocco. I due blocchi contigui e di ugual grandezza vengono detti compagni. Quando viene rilasciato un blocco di  $2^n$  byte, il gestore della memoria deve scorrersi la lista dei blocchi di  $2^n$  byte e vedere se può avvenire una fusione. Questo comporta però frammentazione interna.

#### 4.3 Memoria virtuale 335

- Overlay: il programmatore divide il programma in unità che possono essere contenute in memoria.
- Memoria virtuale: miglioramento dell'overlay nel quale è il S.O. che divide il programma; permette ai programmi di essere eseguiti anche quando sono contenuti solo in parte in memoria principale.
- Paginazione: tecnica utilizzata in sistemi con memoria virtuale nel quale il programma è diviso in pagine:
  - ✓ Indirizzi virtuali = indirizzi di  $4+12=16$  bit generati dal programma che formano lo spazio d'indirizzamento virtuale, diviso in pagine;
  - ✓ MMU (Memory Management Unit) = insieme di circuiti che traducono l'indirizzo virtuale in uno fisico di memoria;
  - ✓ Frame o Immagini di pagina = corrispondenti delle pagine nella memoria fisica;
  - ✓ Bit presente/assente = indica se la pagina è allocata in memoria o no;
  - ✓ Page fault: eccezione dovuta al fatto che il bit presente/assente indica che la pagina non è allocata;
- Tabella delle pagine: tabella interna alla MMU che come entry ha i 4 bit dell'indirizzo virtuale, che danno il numero della pagina virtuale, e restituisce, se il bit presente è a "1", 3 bit che indicano il numero di frame di pagina da aggiungere, come bit più significativi, all'offset dell'indirizzo virtuale per dare l'indirizzo fisico. E' molto grande ( $2^{30}$  pagine) e deve essere veloce nella traduzione.

- **Tabelle delle pagine a più livelli:** si organizza la tabella delle pagine in più livelli per evitare di tenerla tutta costantemente in memoria. Al crescere dei livelli si aggiunge flessibilità ma allo stesso tempo complessità e ritardo. Una tipica voce della tabella delle pagine è composta da:

I.Indice del frame di pagina;

II.Presente/assente;

III.Protezione;

IV.Modificato;

V.Riferito;

VI.Cache disabilitata;

VII. Vuoti;

- **Tabella delle pagine inverse:** in sistemi a 64-bit, ci vogliono  $10^{15}$  byte per contenere la sola tabella. In questo caso contiene un elemento per ogni frame di pagina presente nella memoria fisica anziché un elemento per ogni pagina contenuta nello spazio d'indirizzamento virtuale. Risparmia spazio ma la traduzione da indirizzi virtuali ad indirizzi fisici diventa più difficile.

#### 4.4 Algoritmi per il rimpiazzamento delle pagine 348

- Ogni qual volta si ha un page fault bisogna, se non si ha spazio vuoto, cancellare dalla memoria una pagine. Questo diventa più oneroso se la pagina da rimpiazzare è stata modificata, e deve quindi essere salvata. Ci vogliono inoltre degli algoritmi:

I.Ottimale = prevede la pagina che sarà richiesta il più lontano nel tempo. E' specifico a quel programma ed è anche impossibile prevedere;

II.NRU (Not Recently Used) = utilizza i bit R di riferimento (che periodicamente viene impostato a "0" per distinguere le pagine riferite recentemente) ed M di modifica e divide le pagine in quattro classi e rimuovendo una pagina a caso tra quelle della classe non vuota di livello minimo:

Classe	R	M
0 = non riferita, non modificata	0	0
1 = non riferita, modificata	0	1
2 = riferita, non modificata	1	0
3 = riferita, modificata	1	1

III.FIFO (First-In First-Out) = funziona come una coda, dove il primo ad uscire è il primo arrivato;

IV.Second chance = variante del FIFO nella quale si ha un campo che indica l'istante di caricamento.

Al page fault se R = 0 la pagina viene rimossa, altrimenti viene azzerato, l'istante di caricamento viene aggiornato e la pagina viene messa in fondo alla lista;

V.Orologio = variante del Second chance nella quale la lista diventa circolare ed un puntatore punta alla pagina più vecchia. Al page fault se R = 0 la pagina viene rimossa, altrimenti viene azzerato ed il puntatore passa ad indicare la pagina seguente;

VI.LRU (Least Recently Used) = si considera che le pagine più utilizzate saranno in testa, mentre quelle meno utilizzate lo resteranno ancora a lungo. Ci sono due realizzazioni:

✓Hardware = contatore a 64 bit che viene incrementato dopo ogni istruzione; il S.O. non deve far altro che eliminare la pagina con contatore più basso;

✓Software = matrice di  $n \times n$  bit a "0"; se si riferisce la pagina "k" vengono posti ad "1" i bit della riga "k", mentre a "0" quelli della colonna "k". Anche in questo caso il S.O. non deve far altro che selezionare la pagina che ha come valore quello minimo tra le righe;

VII.NFU (Not Frequently Used) = è la simulazione software della soluzione hardware dell'LRU. Ad ogni pagina è associato un contatore software al quale viene sommato, dopo ogni colpo di clock, il valore del bit R;

VIII.NFU con invecchiamento = il bit R viene inserito come incoming bit di uno shift a destra del contatore di 8 bit;

IX.WS clock = variante dell'orologio nel quale viene eliminata una pagina soltanto se non fa parte del working set;

#### 4.5 Problemi progettuali dei sistemi a paginazione 355

- **Anomalia di Belady:** non è vero che più pagine fisiche si hanno a disposizione minori saranno i *page fault*. Belady portò ad esempio che con un algoritmo FIFO:

- ✓ 4 pagine fisiche ? 9 page fault;
- ✓ 3 pagine fisiche ? 10 page fault;
- Algoritmi a pila: gli accessi alla memoria di un processo possono essere caratterizzati attraverso una lista ordinata di numeri di pagina, che viene detta stringa dei riferimenti; XXX
- Stringa delle distanze: è la stringa che contiene le distanze rispetto alla testa dello stack del punto dove la pagina era allocata. Essa dipende non solo dalla stringa dei riferimenti ma anche dall'algoritmo di paginazione.
- Working set: è l'insieme di pagine correntemente usate da un processo. Si possono avere:
  - I. Paginazione a richiesta = le pagine sono caricate solo su richiesta e non in anticipo;
  - II. Località dei riferimenti = normalmente un programma riferisce soltanto una piccola parte delle sue pagine e per di più vicine tra loro;
  - III. Trashing = è un programma che causa molti page fault ogni poche istruzioni;
  - IV. Modello a working set = si tiene memoria del working set di ogni processo in modo da poter effettuare la prepaginazione, ossia caricare le pagine prima che il processo venga eseguito;
- Dimensioni delle pagine: è una scelta che ricade sul S.O.; l'hardware può infatti essere stato progettato con pagine di 512 byte, ma il S.O. può considerare pagine come multipli di tale valore. Si hanno due possibilità per le dimensioni:
  - I. Grande ? si ha una grande frammentazione interna e quindi uno spreco di memoria;
  - II. Piccola ? si perde in prestazioni a causa dei molti accessi;

## 5 File system 423

- File system: gerarchia per ordinare file e directory.
- File: astrazione che ci permette di conservare, in maniera duratura e persistente, informazioni alle quali i processi possono accedere e sono conservati in dischi o altri mezzi fisici.

### 5.1 File 424

- Denominazione dei file: a seconda del file system si possono avere varie strutture. La più comune è quella di assegnare un nome ed un'estensione separati da un punto.
- Struttura dei file: le possibilità più comuni sono:
  - I. Sequenza non strutturata di byte = il S.O. non si preoccupa di cosa ci sia all'interno dei file;
  - II. Sequenza di record di lunghezza fissa = si presuppone che l'unità minima che ogni operazione sui file può compiere sia il record;
  - III. Albero di record = ogni record ha una chiave, e l'ordinamento è dato dal valore della chiave;
- Tipi di file: si possono avere:
  - I. File ordinari = contengono le informazioni;
  - II. Directory = file di sistema utilizzati per contenere file ed altre directory; servono quindi per strutturare il file system;
  - III. File speciali a caratteri = per interfacciarsi con dispositivi di I/O a caratteri;
  - IV. File speciali a blocchi = interfacciarsi con dispositivi di I/O a blocchi;
- Accesso ai file: alcuni vecchi mainframe permettevano, all'atto della creazione del file, di poter scegliere la modalità di accesso tra i due tipi:
  - I. Sequenziale = per poter leggere il byte  $n$  bisogna scorrere i byte precedenti;
  - II. Casuale = i byte possono essere letti in qualsiasi ordine;
- Attributi dei file: solitamente riguardano: protezione, flag, record, indicazioni temporali, dimensione.
- Operazioni sui file: si attuano tramite chiamate di sistema e sono: create, delete, open, close, read, write, append, seek, get attributes, set attributes e rename.

### 5.2 Directory 432

- Sistemi con gerarchie di directory: si possono avere:
  - I. Unica directory = tutti i file sono conservati senza nessuna organizzazione;
  - II. Una directory per ogni utente = per poter distinguere i file di ogni utente;
  - III. Albero di directory = si possono avere directory all'interno di altre directory;
- Nomi di percorso: si distinguono: nome assoluto, nome relativo e directory di lavoro o corrente.
- Operazioni sulle directory: si attuano tramite chiamate di sistema e sono: create, delete, opendir,

closedir, readdir, rename , link e unlink.

### 5.3 Implementazione del file system 437

- Implementazione dei file: S.O. diversi utilizzano metodi diversi:
  - I. Allocazione contigua = i file vengono memorizzati uno dietro l'altro. E' semplice da implementare e le prestazioni sono buone, ma non è fattibile nei sistemi che consentono di variare dinamicamente la grandezza del file e si ha frammentazione;
  - II. Allocazione per liste = la prima parola del blocco contiene un puntatore al blocco successivo. Non spreca spazio a causa della frammentazione ma l'accesso casuale è lento e parte dello spazio è utilizzato per memorizzare il puntatore;
  - III. Allocazione per liste usando un indice = si conserva in memoria un indice dei blocchi, in maniera da ovviare ai problemi prima riscontrati. Questo però comporta che l'intera tabella sia memorizzata costantemente e completamente in memoria;
  - IV. I-node = ad ogni file è associata una tabella che contiene attributi ed indirizzi dei blocchi del file su disco. Ci sono:
    - 1) Blocchi ad indirizione singola = contengono gli indirizzi di blocchi sul disco;
    - 2) Blocchi a doppia indirizione = contengono una lista di blocchi ad indirizione singola;
- Implementazione delle directory: la directory ha una struttura differente per ogni S.O.
- Gestione dello spazio su disco: si hanno due strategie:
  - I. Allocazione contigua = si cerca una sequenza contigua di blocchi liberi per conservare il file;
  - II. Frammentazione = si spezzetta il file in un certo numero di blocchi eventualmente non contigui;
- Dimensione del blocco: vedi 4.5.
- Gestione dei blocchi liberi: si hanno due strategie:
  - I. Serie di blocchi per organizzare una lista = all'interno di uno o più blocchi sono salvati gli indirizzi dei blocchi liberi;
  - II. Bit map = vedi 4.2;
- Affidabilità del file system: i dischi possono contenere dei blocchi difettosi che vengono sostituiti con altri di riserva.
- Consistenza del file system: dopo un crash il file system può essere inconsistente. I blocchi danneggiati possono contenere dati ma possono anche contenere gli i-node, i che comporta la perdita di una quantità di dati. Per controllare la consistenza dei blocchi il S.O. genera due tabelle, inizializzate a "0" che contano quante volte un blocco compare all'interno di un file e quante volte ogni blocco appare nella lista dei blocchi liberi. Se il file system è consistente ogni blocco avrà un uno o nella prima o nella seconda tabella. Si possono verificare:
  - I. Missing block = ha "0" in entrambe le tabelle ? viene aggiunto tra i blocchi liberi;
  - II. Blocchi duplicati = il contatore del blocco segna un numero maggiore di 1 ? viene settato ad 1;
  - III. Un blocco appare in due o più file ? si alloca un nuovo blocco, vi si copia il contenuto del blocco e si inserisce la copia in uno dei file;
- Prestazioni del file system: per ridurre l'accesso al disco si utilizza una cache dei blocchi, ossia una copia in memoria principale di alcuni blocchi. Anche in questo caso si devono implementare delle politiche di rimpiazzamento in memoria dei blocchi.

### 5.4 Sicurezza 457

XXX