www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

E se la politica ***second chance*** fosse stata locale? Per caricare C o A devo rimpiazzare SOLO le pagine di C o A.

Basta isolare la lista di prima *per lettera*

FILE SYSTEM

I dati hanno durata più lunga delle applicazioni che li usano

ESIGENZE PRINCIPALI
- persistenza dei dati
- possibilita' di condivisione dei dati tra applicazioni distinte
- nessun limite di dimensione fissato a priori

File system e' il servizio di S/O progettato per soddisfare questi bisogni

in particolare
- → li organizza
- → li gestisce
- → li realizza
- → li accede

FILE: insieme di dati correlati residenti in memoria secondaria e trattati unitariamente

Progettazione
1. modalita' di accesso a file, struttura logica e fisica, opzioni ammesse ⇒ **cosa offrire all'utente**
2. **come realizzarlo**

file= concetto logico realizzato tramite meccanismi di astrazione

aka insieme di dati percepiti dall'utente come unitari, può nominarli, vuole poterli spostare
↳ **nomi logici**

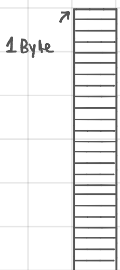
CARATTERISTICHE DISTINTIVE

attributi (tra cui anche il nome logico):

nome, estensione (tipo di file), data di creazione, dimensione corrente, data di ultima modifica, creatore e possessore, permessi di accesso, protezione, password, creatore, [proprietario, uso, visibilità, livello, archiviazione, tipo di contenuto, tipo di accesso, permanenza] → Flag 0 o 1

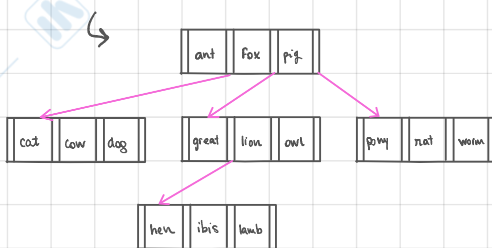
Strutture dati all'interno di un file può essere considerata da 3 punti di vista distinti:
1. LIVELLO UTENTE: programma applicativo associa autonomamente significato al contenuto grezzo del file
2. LIVELLO di STRUTTURA LOGICA organizzazione i dati in strutture logiche per facilitarne il trattamento
3. LIVELLO di STRUTTURA FISICA il sistema operativo mappa le strutture logiche sulle fisiche della memoria secondaria (tipo i settori di un disco)

possibili strutture logiche: a sequenza di byte, a record di lunghezza e struttura interna fissa, a record di lunghezza e struttura interna variabile

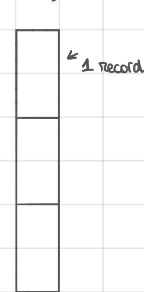


ovvero NON c'e' una struttura logica
E' la più rudimentale ma anche la più flessibile

L'accesso dei dati utilizza un puntatore relativo all'inizio del file



Modo di organizzare i dati complesso, ogni struttura interna di un record e' identificata univocamente da una key in postazione fissa
Queste keys vengono raccolte in una tabella
L'accesso avviene per chiave
Abbastanza diffuso in sistemi mainframe



Non e' detto che i record siano riempiti e il S/O deve conoscerne la struttura interna

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Modalita' di accesso = SEQUENZIALE
Un puntatore indirizza il ***record*** corrente e avanza a ogni lettura e scrittura

- può avvenire in qualsiasi posizione del file, la quale però deve essere aggiunta sequenzialmente
- può avvenire solo in coda al file (append)

DIRETTO opera su record di dati posti in posizione arbitraria nel file
INDICIZZATO per ogni file una tabella di coordinate contenenti gli offset dei rispettivi record nel file
↳ le informazioni non sono più nei record ma in una struttura a parte ad accesso veloce
I.S.A.M. index sequential access method, consente accesso sia indicizzato che sequenziale

IL FILE SYSTEM può trattare diversi tipi di file
file regolari sui quali si può operare normalmente
file catalogo tramite i quali il file system permette di descrivere l'organizzazione di gruppi di file
file speciali con i quali il FS rappresenta logicamente dispositivi orientati a carattere o a blocco
↓
tipo USB

Operazioni ammesse:
creazione file inizialmente vuoto, inizializzazione attributi
apertura deve precedere il primo uso, predispone le informazioni utili all'accesso
cerca posizione solo per accesso **casuale**
cambia nome che considera anche lo spostamento nella struttura logica
distruzione rilascio della memoria occupata
chiusura rilascio delle strutture di controllo usate per l'accesso ed il salvataggio dei dati
lettura/scrittura read, write, append
trova attributi make
modifica attributi permessi

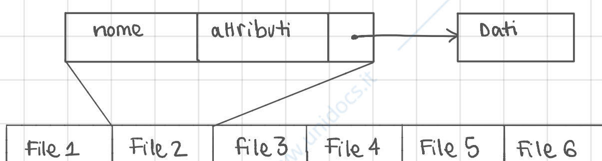
Sessione d'uso di un file:
si può accedere solo ad un file già aperto tramite uno strumento specifico di accesso chiamato **handle**
Dopo l'uso il file viene chiuso

S/O mappa un file in memoria virtuale: il file continua a risiedere in memoria secondaria ma all'indirizzo di memoria virtuale (base + offset)
↳ uguale nel caso di segmentazione
Tutte le modifiche apportate al file, devono essere riportate anche in memoria secondaria
Questa soluzione riduce gli accessi ma porta a problemi con la condivisione: dove trovo la versione corrente?

DIRECTORY

Ogni file system usa le directory o folder per tenere traccia dei suoi file

Sono classificate rispetto all'organizzazione di file che consentono, tipo:
a livello singolo, a due livelli, ad albero, ...



Tutti i file sono elencati su un'unica lista lineare

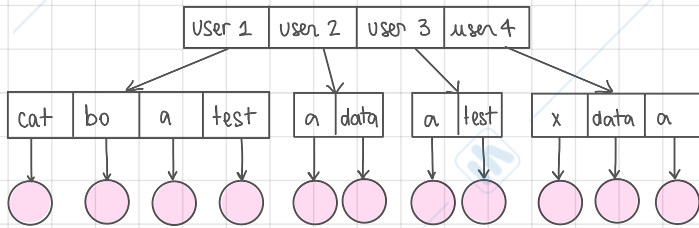
REQUISITI FONDAMENTALI a livello utente:
efficienza ovvero trovare/creare un file deve essere facile,
libertà di denominazione, diversi nomi per stesso file e viceversa stesso nome per file diversi (a seconda dell'utente)
libertà di raggruppamento, opportunità di creare gruppi logici sulla base di proprietà significative per l'utente

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Directory a due livelli

Una root directory che contiene una user File directory per ciascun utente, e ogni utente può guardare SOLO la propria UFD.
I File sono localizzati tramite percorso path name
È possibile porre certi File in una directory di sistema condivisa in modo da essere accessibili da tutti gli utenti

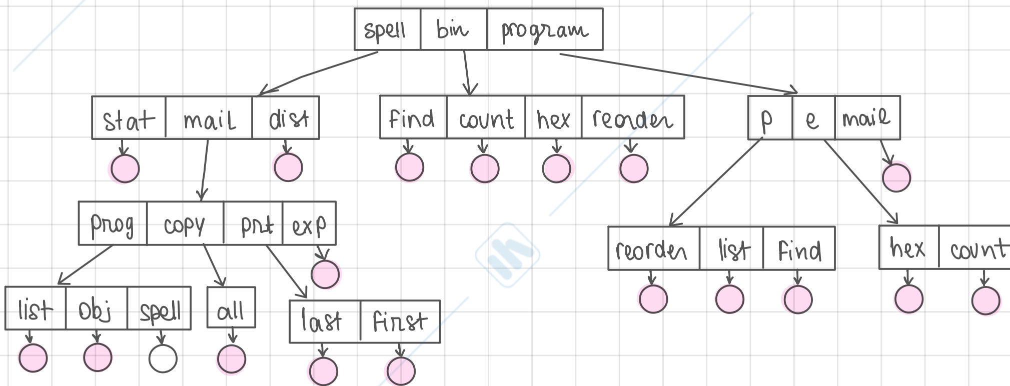


Requisiti parzialmente soddisfatti come l'efficienza di ricerca e la libertà di denominazione ma per esmepio non permette la libertà di raggruppamento

DIRECTORY AD ALBERO

il numero di livelli è arbitrario, il livello superiore viene sempre detto **root**
Ogni directory contiene file regolare o anche altre directory a sua volta di livello inferiore.
ogni utente ha la sua *directory* che può cambiare con comandi di sistema. Se non si specifica il path si assume come riferimento la *directory corrente*
Il cammino può essere **assoluto** oppure **relativo**

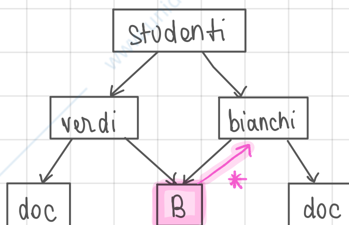
- assoluto → rispetto alla radice di FS
- relativo → rispetto alla posizione corrente



Anche in questo caso, come nella directory a due livelli, non viene soddisfatto il requisito di libertà di raggruppamento

DIRECTORY A GRAFO

Avviene quando un file è raggiungibile da due cartelle distinte



La forma generalizzata consente collegamenti ciclici e quindi riferimenti circolari
Attenzione nel caso di ricerche ricorsive all'interno di un file system con riferimenti circolari, rischio che la ricerca **non termini mai**
Un S/O potrebbe duplicare gli identificatori di accesso al file, ma questo rende più difficile assicurare la **coerenza** del file (come per esempio di chi sia quel file, o quale è la versione più recente, modifiche ecc)

* aggiungendo questa freccia ho reso il grafo ciclico

HARD LINK		SYMBOLIC LINK
Collegamento tramite un puntatore ad un File regolare che viene inserito nella directory ad esso remota ↳ **crea due vie di accesso dirette ad uno stesso File** MA: Nel momento in cui cancello quel File, cosa sto davvero facendo? Problemi di coerenza dei File	VS	Viene creato un File speciale il cui contenuto è il cammino del File originario. Si mantiene **1 sola via di accesso** tramite un collegamento falso, è un po' più lento ma non ho problemi di coerenza.

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

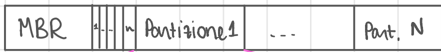
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

File system contenuti nel disco, ma i dischi possono essere partizionati.

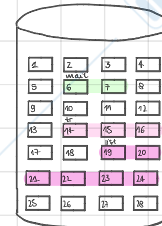
Il settore 0 contiene le informazioni di inizializzazione di sistema: Master Boot Record (MBR) contiene in 512B una descrizione eseguita dal BIOS ↙ delle partizioni che identifica quella attiva
↓
specifiche per l'inizializzazione di **una partizione** = boot block

Unità informativa del disco = settore
ma un disco viene letto e scritto a blocchi o cluster

MBR		Partizione 1	...	Part. N



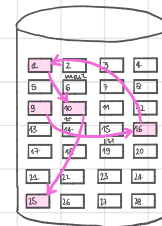
File = insieme di blocchi sul disco (non necessariamente contigui)
Infatti ci sono 3 metodi di allocazione
1. allocazione contigua
2. linked list
3. lista indicizzata



directory

file	start	length
mail	6	2
tr	14	3
list	19	6

(1) Mi basta dire l'inidrizzo di inizio del file e dire quanto è grande. Consente accesso sequenziale e anche diretto (perché ho tutte le informazioni necessarie).
MA ogni modifica comporta il rischio di frammentazione esterna (e un'eventuale ricompattazione costerebbe tempo)



start 9
end 25

(2) Il file è identificato dal puntatore al suo primo blocco (o anche dall'ultimo per esempio per operazioni di aggiunta)
Ciascun blocco deve contenere il puntatore al blocco successivo (e questo sottrae spazio ai dati)
Possibile l'accesso sequenziale, difficile quello diretto.
Un solo guasto corrompe l'intero file

(3) Simile a quella concatenata, ma si pongono i puntatori ai blocchi in una tabella
apposita. In questo modo ogni blocco contiene **solo** dati.
2 strategie di organizzazione:
- forma tabulare, **FAT** *File Allocation Table*
- forma indicizzata, **nodo indice** *i-node*

Consente accesso sequenziale e diretto, non causa frammentazione interna, non richiede di conoscere la dimensione massima di ogni nuovo file

FAT, File Allocation Table

Base di Windows, tabella ordinata di puntatori
(puntatore ∀ blocco del disco)
La porzione di FAT relativa ai file deve risiedere interamente in RAM
⇒ grandezza dipende dalla dimensione del disco
File = catena di indici

Nodi indice

UNIX → GNU | Linux
Struttura indice i-node ∀ file, con gli attributi di quel file e i puntatori ai suoi blocchi
In RAM risiede una i-node solo per **i file in uso**
⇒ la grandezza dipende dai file in uso
Un' i-node contiene un numero limitato di puntatori a blocchi
(E se un file ha bisogno di più puntatori?)

↙ File di piccola dimensione, sono ampiamente contenuti in un singolo i-node ⇒ frammentazione interna

↓ File di media dimensione, non gli basta un i-node, sacrifico un campo per renderlo puntatore a un altro blocco i-node

↘ File di grandi dimensioni si usa un puntatore a un blocco i-node che punta a un i-node dati

Per file ancora più grandi basta aggiungere altri livelli di indirezione

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

GESTIONE FILE CONDIVISI

Abbiamo visto che un file è condiviso, se esistono più percorsi che portano allo stesso file.
Per preservarne la coerenza, abbiamo detto che si possono creare due 'collegamenti' (*symbolic* e *hard link*)

puntatore speciale, si mantiene una sola via di accesso che è l'i-node del file originale

due vie di accesso: nella directory remota si pone il puntatore diretto al descrittore del file originale = i-node

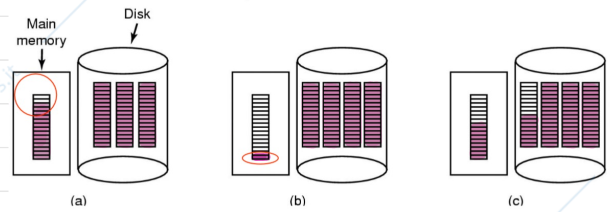
Nel caso di **hard link** si ha che esiste un proprietario effettivo ma più possessori e dunque il file non può venire eliminato fino a che esistono descrittori (collegamenti i-node) remoti a quel file

GESTIONE DEI BLOCCHI LIBERI

- array du i **bit map** dove ogni bit indica lo stato del corrispondente blocco (0 libero, 1 occupato)
- lista concatenata di blocchi liberi sfruttando i campi puntatore al successivo (architettura FAT)

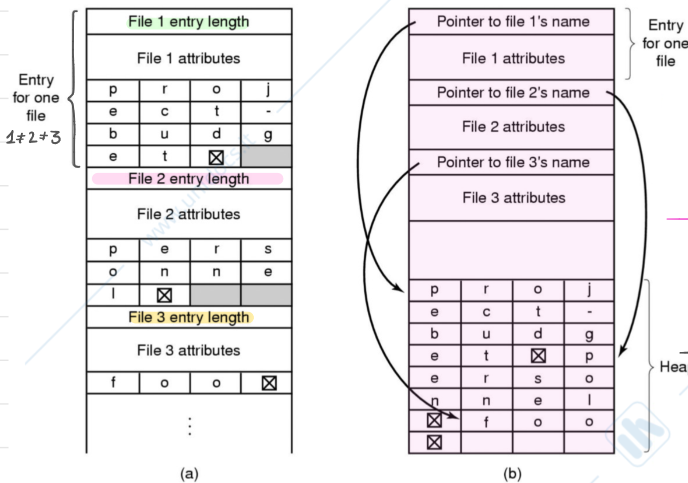
↳ equivalente di un file ma i suoi blocchi sono liberi

Nel momento in cui ho una struttura dati che mi indica i blocchi liberi, e volessi caricarla in RAM...mi accorgo che sono rimasti pochi spazi liberi. Potrei avere un file che consuma sti spazi liberi e anzi ha bisogno di altro spazio.
Carico un altro blocco in RAM ma dopo l'aggiunta di un file mi trovo nella stessa posizione di prima.
soluzione alternativa: una volta riempito il blocco, si dividono i suoi puntaori **in due parti: metà in blocco RAM, metà in blocco su disco**



REALIZZAZIONE DELLE DIRECTORY

File e directory, risiedono in aree logiche distinte per garantire la corretta gestione di entrambi (hanno usi diversi)
Per minimizzare la gestione della directory, l'ideale è utilizzare una **struttura fissa** anche se le istruzioni contenute NON solo necessariamente a dimensione fissa.
Possiamo avere per esempio [nome+attributo] oppure anche [nome+puntatore ad un i-node con attributi]



Per accedere ad una informazione del file 3, devo prima sapere dove finisce l'1 e dove finisce il 2 e **poi posso accedere** al file 3
(con entry length so già di quanto saltare)

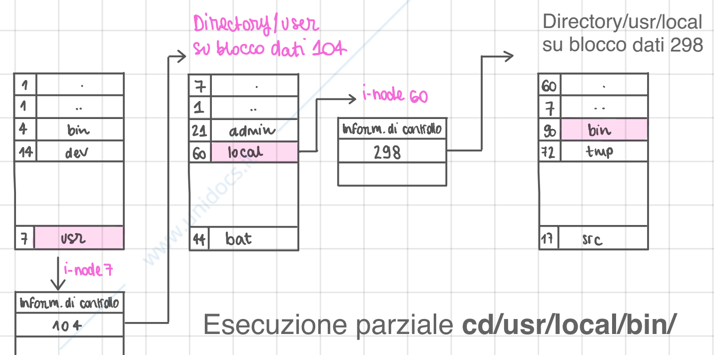
→ Entry a dimensione fissa, niente frammentazione interna

→ Rischio di page fault nella lettura del nome (perché magari il resto è in un'altra pagina)

Slide 291 – 294 Vecchi compilatori

UNIX v7

Concepito e realizzato tra il 1969 e il 1979.
Struttura ad albero e condivisione di file (grafo aciclico)
Nomi dei file fino a 14 caratteri ASCII
La ***directory*** contiene solo nome e puntatore al suo i-node per un massimo di 64K file per file system (2 i-node)
Un i-node (64B) contiene gli attributi del file **incluso il contatore di directory che puntano al file tramite hard link**
Se il contatore = 0 vengono liberati i blocchi



Esecuzione parziale **cd/usr/local/bin/**

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

GESTIONE DEI BLOCCHI DANNEGGIATI

Nel momenti in cui mi rendo conto che un blocco e' stato danneggiato, devo assicurarmi che venga rimosso e non utilizzato da altri utenti.
Ci sono due modi: **hardware** e **software**

- hardware → creo e mantengo in un settore del disco un elenco di blocchi danneggiati e dei loro sostituti
- software → ricorro ad un falso file che occupa tutti i blocchi danneggiati

SALVATAGGIO DEL FILE SYSTEM

Posso salvarlo sul nastro (tempi lunghi anche se costa poco) oppure sul disco,o generando una partizione di backup oppure con l'**architettura RAID**

Come faccio a verificare la consistenza di un file system?
Se per esempio un mentre modifico un file, si spegne il sistema? Quando si riaccende ho ancora la versione vecchia
E la consistenza dei blocchi?
Basta usate 2 puntatori, uno alla lista dei blocchi in uso, uno alla lista dei blocchi liberi:

✅ ciascun blocco appartiene **ad una e una sola lista**
❌ un blocco non appartiene a **nessuna lista**
🔁 il contatore del blocco è >1 in una delle due liste

Per ridurre la frequenza di accesso ai dischi, è utile dedicare una porzione di memoria principale viene usata come ***cache*** di blocchi. Occorre però garantire la consistenza dei dati:

MS-DOS tecnica write through
UNIX → GNU/Linux processo periodico detto sync effettua l'aggiornamento dei blocchi su disco

Da UNIX a GNU/Linux

Processo principale attività nel sistema, concorrenza a livello di processi
↳ processi attivati direttamente dal sistema (=daemon) o creati mediante **fork ()** (clone con stessa memoria all'inizio e accesso ai file aperti)

I processi figli hanno all'inizio la stessa memoria identica a quella del processo genitore, poi alla prima modifica i due diventano **completamente indipendenti** (copy on write)

I processi possono avere più flussi di controllo detti **thread** , che condividono tutte le risorse logiche e fisiche del processi genitore (anche dopo la modifica)

res = pthread_create (&tid ①, attr ②, fun ③, arg ④)   ① identità ③ compito, funzione ② attributi ④ argomenti

Cosa succede se **fork()** include più thread? Due possibilità:
- tutti i thread del padre vengono clonati → aggiunge gradi di difficoltà nell'uso concorrente + nella comunicazione tra le entità attiva
- solo un thread del genitore viene clonato
  ↓ problemi di inconsistenza per le thread non clonate

I thread sono gestiti dal kernel, l'ordinamento (che sia si thread o di processi) avviene per **task** (↳ selezionati a seconda delle priorità ecc.)

Tre classi di priorità dei **task:**
tempo reale con politica FCFS a priorità senza prerilascio, tempo reale con politica RR a priorità, divisione di tempo RR a priorità
Anche se in realtà non sono proprio a tempo reale
↓ Priorità dinamica

GESTIONE MEMORIA UNIX

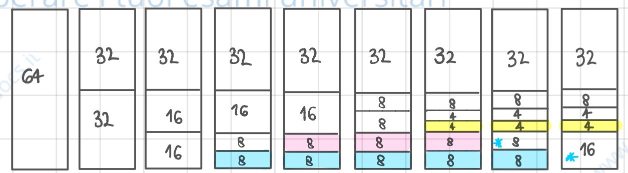
In origine l'allocazione avveniva mediante ***swap*** di processi e lo *swapper* (gestore) creava lo spazio necessario salvando sul disco i processi sospesi con più tempo d'esecuzione recente e minor priorità.
In questo modo si permetteva di avere il minor numero di spostamenti da RAM e memoria secondaria.
Successivamente venne introdotta la ***paging on demand*** ovvero mano a mano che il processo è in esecuzione vengono emesse delle richieste, senza **preallocare** anticipatamente le pagine.

RAM algoritmo di allocazione primario ***buddy***
Ogni richiesta N viene arrotondata alla prima potenza di 2 > N
La memoria disponibile viene frazionata in metà successive fino a frazioni di ampiezza 2^m : una singola fazione viene assegnata al richiedente e viene creata anche una struttura ausiliaria che contiene la testa di liste predefinite di frazioni (**per velocizzare la ricerca)**

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

La memoria disponibile usata per l'allocazione è sempre la frazione di minore dimensione e appena una frazione viene rilasciata di unisce con la sua frazione vicina (il suo buddy)
Vengono inoltre utilizzate delle strutture ausiliarie dette **slab** per evitare di usare 128 pagine per allocarne 65.



insieme contiguo di 64 pagine→richiesta di 6 pagine contigue (2^3=8) → richiesta di 5 pagine (2^3=8) → richiesta di 4 pagine (2^2=4) → rilascio e ricompattazione*

FILE SYSTEM

vengono usati gli i-node.
Un i-node per ogni file, prima ci sono tutti gli attributi e poi i vari puntatori (10/12 di default intanto e poi fino a 3 livelli di indirezione).

Esempi

ipotesi: blocco dati ha capienza 4KB, l'i-node è ampio 64B, gli indici di blocco sono espressi su 4B

esempio 1 single-indirect

$(12 + 64/4) \times 4kB = (12+16) \times 4 = 112kB$

* puntatori di prima indirezione

esempio 2 double indirect

$112kB + 16^2 \times 4kB = 1MB + 112kB$

esempio 3 triple indirect

$1MB + 112kB + 16^3 \times 4kB = 17MB + 112kB$

Dimensioni massima di un file

Inizialmente basato su MINIX, poi venne abbandonata quella versione e venne introdotta la versione di riferimento **ext2**
Maggiore innovazione = divisione della partizione di disco in gruppi di blocchi in modo da poterli usare per file più grandi e per velocizzarne l'accesso

Dimensione dell'i-node venne estesa a 128B, con indirizzi di blocco ampi 4B
Blocchi di dimensione pari a 1, ,e 4 KB scelta in fase di configurazione del file system

Ogni aggiunta a file viene realizzata quanto più **localmente** possibile ovvero avendo un gruppo di blocchi, se voglio aumentare un file preferibilmente cerco di farlo all'interno di quello stesso gruppo (*contiguità*)

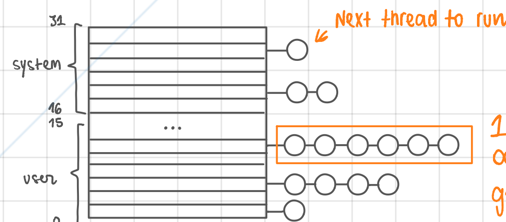
WINDOWS

Gestione dei processi

job = gruppo di processi gestito come singola entità

processo = possessore di risorse con ***thread*** > 1 (thread = flusso di controllo gestito dal kernel)
4GB di spazio di indirizzamento (2 utente e 2 nucleo) ed inizialmente viene creato il suo singolo thread
non ha uno stato di avanzamento (infatti è il thread che avanza)
fiber = suddivisione di thread ignota al nucleo, esegue nell'ambiente del thread e viene gestita dla sottosistema Win32
non hanno un contest switch, è un ulteriore suddivisione molto più leggera da gestire

6 classi di priorità per processo: realtime, high, above-normale, normale, bellw-normale, idle
7 classi di priorità per i thread: Time-critical, highest, above-normal, normal, below-normal, lowest, idle
32 livelli di priorità (0.....31)



0-15 = ordinarie
16-31 = di sistema

↳ dato dalla combinazione tra priorità processo e priorità thread
Ciascuno è associato a una coda di *thread* pronti, e questi non sono distinti per processo di appartenenza

Ciascun thread ha una priorità base iniziale e una corrente che **varia** nel corso dell'esecuzione
La priorità corrente si eleva quando il thread o completa l'operazione di I/O oppure ottiene un semaforo/riceve segnale d'evento.
La priorità decresce a ogni quanto consumato (che può essere più ampio nel caso della finestra in primo piano)
Per **mitigare il problema di priorità** si usa una tecnica particolare ovvero se un thread pronto non è selezionato per un certo quanto di tempo, riceve un incremento di priorità per 2 quanti

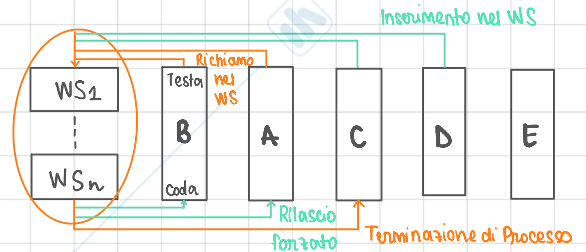
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

Gestione della memoria

Ciascun *page frame* può essere:
- in **uso** ovvero appartenente ad un working set
- **rilasciata** ovvero appartiene ad una e ad una sola lista tra le seguenti:
  - **A** in **attesa** può essere riassegnata e sovrascritta senza problemi, è ancora associata ad un certo processo ma non sta venendo usata, posso rimpiazzarla
  - **B** da **copiare su disco** una pagina rimpiazzata ma che essendo stata rimossa deve essere riportata la modifica sul disco
  - **C libera** fase successiva di A, pagina non associata ad alcun processo
  - **D azzerata** come C, ma il contenuto è stato obliterato a 0, tipo resettata (per evitare che si possa in quale modo risalire al contenuto precedente)
  - **E difettosa** fisicamente: pagina che non può essere più utilizzata

Lo **swapper thread** *(daemon)* del **memory manager** si occupa di verificare l'effettivo uso delle pagine, porta le pagine dei processi i cui *thread* siano stati recentemente inattivi in A o B.
Altri 2 daemon assicurano che vi siano abbastanza pagine in C salvando quelle in B e accodandole in A.



Un WS che cresce preleva le pagine libere da C oppure da D (daemon dedicato che periodicamente **azzera le pagine di C** e le pone in D)

Gestione del file system

Venne adottato il sistema NTFS (New Technology File System) molto più efficiente.
Ha la **nuova concezione** di indici espressi su 64bit.
Principale struttura dati = **MFT Master File Table** fisicamente realizzata come un file, logicamente strutturata come una sequenza di 2^{48} record di ampiezza da 1 a 4KB.
Ciascun record descrive 1 file identificato da 48bit, i restanti 16 servono come numero di sequenza.
Questi record sono gli analoghi degli **i-node** per i sistemi GNU/Linux

Ciascun record contiene un numero variabile di coppie **< descrittore di attributo, valore >**

- descrittore di attributo: esistono 13 attributi di base con una struttura predefinita, possono poi essere aggiunti attributi a struttura libera
- valore: se il valore è piccolo lo inserisco direttamente (residente), altrimenti inserisco un puntatore al suo record remoto (NON residente)

A differenza degli i-node, se un record ha piccole dimensioni, può essere **interamente contenuto nel record** senza ricorrere a puntatori ==> semplifica l'accesso

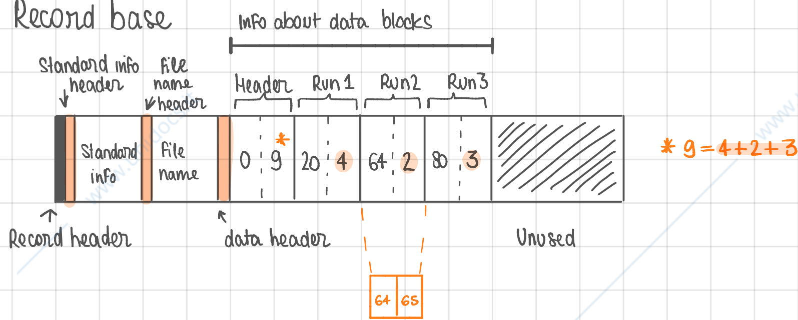
I primi 16 record del MFT sono riservati per file trascendenti di sistema (**metadata**) e descrivono l'organizzazione dell'intero volume.
Il primo (0) record descrive l'MFT stesso, il secondo (1) replica i primi 16 in modo non residente ponendone il contenuto in fondo al volume (***mirror file),*** il quarto (3) caratterizza il volume (nome logico, versione, data di creazione...), il quinto (4) descrive gli atrributi.
Inoltre abbiamo:
- il puntatore alla radice del FS
- bitmap dei blocchi liberi
- copia del codice di boot di volume il suo puntatore ecc..

Non tutti gli attributi di sistema si applicano a tutti i file, gli attributi previsti per i file sono quelli che corrispondono a quelli che Linux mette negli i-node con l'aggiunta dell'identificatore dell'oggetto corrispondente
Il contenuto dati di file **con ampiezza minore ad 1KB** viene memorizzato interamente entro un record di MFT (ma questi sono ***immediate file***)
Per file più grandi l'attributo dati diventa una lista ordinata di blocchi su disco

NTFS cerca di assegnare allo stesso file sequenze di blocchi contigui. Nel caso peggiore i dati adiacenti di un file possono trovarsi su sequenze di blocchi singoli non adiacenti.
Esiste un record base per ogni file sequenziale presente nel volume (come in Linux esiste un i-node).
File con zone interne non utilizzate vengono chiamati ***file sparsi*** e vengono gestiti diversamente

www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari

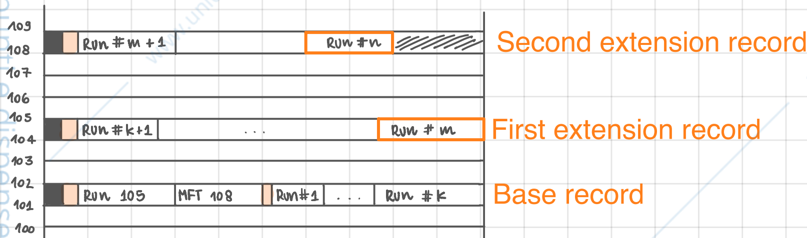
www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari



Un singolo descrittore basta a contenere l'intera lista di sequenze di blocchi contigui. L'intestazione dell'attributo dati specifica il totale di sequenze presenti, la prima coppia di attributi specifica l' offset del file e l'ampiezza

La strategia NTFS permette virtualmente di rappresentare un file di ampiezza **illimitata**
Il numero di record necessari dipende dalla contiguità piuttosto che dalla sua ampiezza: un file da 20GB costituito da sequenze di 1M blocchi da 1KB ciascuno, richiede 20+1 coppie di valori espressi su 64bit ovvero 21x2x8B = 336B
Un file da 64KB costituito da 64 sequenze di 1 blocco ciascuna richiede (64+1)x2x8B = 1040B

La rappresentazione di file può richiedere anche più di un record. Viene usata una tecnica a continuazione simile alle varie indirezioni degli i-node..
Se non vi fosse abbastanza spazio in MFT per i record secondari di un file, l'intera lista viene considerata come ***attributo non residente*** e posta in un file dedicato denotato da un record posto in MFT



www.unidocs.it - Appunti e dispense per superare i tuoi esami universitari