Esame sistemi operativi

1. Descrivere l'allocazione dinamica della memoria di un sistema di archiviazione di massa. Identificare le tecniche più utilizzate, descrivendone i comportamenti nei casi specifici, e compararle dal punto di vista dell'efficacia e dell'efficienza. Nello specifico delle tecniche sopra introdotte, discutere il problema della frammentazione ed identificare le soluzioni più comunemente adottate.

processo di gestione e distribuzione dello spazio di archiviazione tra file e app che richiedono memorizzazione dati.

Nome tecnica	Descrizione caso	funzionamento	efficacia	efficienza
Best fit	- ottimizzazione ed approssimazione - alloca memoria a blocchi di dim(variabile) - trova blocco che si adatta meglio a richiesta	- algoritmo trova min_block tale che dim(richiesta) ≤ dim(disponibile) - porzione del blocco viene assegnata al processo - if (spazio_residuo < 0) new_free_block[] = spazio_residuo - update_list == true	- Riduce frammentazione interna - aumenta frammentazione esterna long term	- costa più tempo del first fit - per allocazione (deve esaminare tutto l'elenco ogni volta) - per deallocazione (deve ordinare l'elenco ogni volta)
Worst fit	- cerca blocco più grande disponibile - cerca di lasciare liberi blocchi più grandi possibili - cerca dall'inizio	- processo richiede memo - algoritmo cerca MAX_slot_avalable - porzione del blocco viene assegnata al processo - if (spazio_residuo < 0) new_free_block[] = spazio_residuo - update_list == true	- riduce frammentazione esterna - rischio frammentazione interna - allocando da blocchi grandi si può lasciare molto spazio inutilizzato	- facilita allocazione di memoria residua - se block_list[] è grande la ricerca di MAX_slot_avalable richiede tanto tempo - t(deallocazione) simile a best fit → inefficiente se elenco è ordinato
First fit	- cerca primo blocco abbastanza grande - cerca dall'inizio ogni volta	- processo richiede memo - algoritmo cerca primo blocco tale che free_dim > requested_dim - if (spazio_residuo < 0) new_free_block[] = spazio_residuo - update_list == true	- frammentazione esterna (molti buchi di memoria difficilmente riutilizzabili)	- più veloce all'inizio - facile da implementare - diventa lento più a lungo viene usato
next fit	- variante del first fit - riprende la ricerca da dove si era fermato l'ultima volta	Procedura identica a first fit	- simile a first fit ma più prevedibile nel t	- distribuisce allocazioni nella memo - molto veloce - può richiedere fusione blocchi

Esame sistemi operativi

allocazione dinamica della memoria → tecniche comuni:

- allocazione contigua → assegna blocchi di memoria adiacenti per un file (efficiente inefficace) → crea frammentazione esterna → soluzione: garbage collection

- allocazione a lista concatenata → divide file in blocchi posizionabili ovunque → collegati con puntatori → efficiente uso memoria – riduce frammentazione – accesso diretto lento

- allocazione indicizzata → usa struttura di indice separata con puntatori a tutti i blocchi di un file → bilancia efficienza accesso e gestione frammentazione → aggiunge overhead di gestione

problema frammentazione:

- avviene quando memoria libera divisa in blocchi piccoli la cui dimensione non è sufficiente per soddisfare una richiesta di memoria anche se la sum(dim_free_block[]) potrebbe essere sufficiente.

- si divide in due tipi:

1. frammentazione esterna → spazio inutilizzabile tra blocchi assegnati
2. frammentazione interna → spazio all'interno di un blocco assegnato non è completamente utilizzato

- best fit → cerca di ridurre frammentazione → min_block tale che dim(richiesta) ≤ dim(disponibile)
  - → riduce frammentazione interna
  - → aumenta frammentazione esterna → può lasciare liberi blocchi troppo piccoli
  - → soluzione → compattazione periodica (riorganizza blocchi di memoria)

- first fit → assegna primo spazio libero sufficiente cercando dall'inizio
  - → approccio semplice e veloce
  - → può provocare frammentazione esterna → può lasciare blocchi liberi tra quelli usati
  - → soluzione → periodic garbage collection (pulizia della memoria che recupera spazi liberi)

- worst fit → assegna spazio più grande disponibile
  - → più lento di best fit
  - → riduce frammentazione esterna
  - → aumenta frammentazione interna
  - → soluzione → allocazione a blocchi di dim[variabile] (adatta dim(block) riduce f. interna)

- next fit → simile a first fit
  - → inizia ricerca da dove aveva interrotto
  - → riduce leggermente frammentazione esterna rispetto a first fit
  - → soluzione → periodic garbage collection (riduce frammentazione esterna)

Esame sistemi operativi

2. Introdurre il concetto di Job-Shop Scheduling per i processi, con particolare riguardo al problema della semantica della concorrenza per processi senza controllo dell'interleaving. Per il più comune algoritmo di gestione della concorrenza con tecniche di scope control, descrivere il metodo di gestione dello stato dell'assegnazione mediante semafori e discutere gli effetti sul throughput e sul waiting time del time sharing.

Job-shop scheduling per i processi → modello di scheduling gestisce l'assegnazione dei processi alle risorse disponibili, ottimizzando l'utilizzo delle risorse e minimizzando il tempo di completamento.

Il problema della semantica della concorrenza per processi senza controllo dell'interleaving

si tratta del problema della gestione di più thread che vengono eseguiti simultaneamente usando risorse comuni in modo non deterministico e non controllato, concetto critico in ambienti multi-thread e/o job-shop scheduling.

I problemi che ne possono risultare sono:

Nome	Descrizione
race conditions	Almeno 2 processi accedono alla stessa risorsa simultaneamente → output dipende da ordine esecuzione operazioni
deadlock	Entrambi i processi aspettano che l'altro inizi → stallo
livelock	I processi cambiano di stato senza fare nulla → simile a deadlock
starvation	Un processo non ottiene mai accesso a risorse perché continuamente prevaricato

descrivere metodo di gestione dello stato dell'assegnazione mediante semafori

semafori → strumenti comuni per gestione concorrenza e controllo accesso alle risorse condivise → permettono sincronizzazione ed esecuzione processi evitando i problemi sopra elencati

scope control con semafori → delimita sezioni critiche di codice che accedono a risorse condivise → usa funzioni di wait (quando acquisisce semaforo) e signal (quando rilascia → esce da critical section)

throughput → = n processi completati in un unità di tempo → semafori possono avere effetti mutevoli

Positivi	Negativi
Sincronizza accesso risorse → evita race conditions (evita system crash) → migliora stabilità e throughput	Uso di wait() e signal() introduce overhead → if (system_type == competitive) throughput --
	Se non usati correttamente → deadlock → throughput decrease harshly

discutere effetti del waiting time del time sharing

waiting time -> tempo medio che un processo aspetta prima di accedere alla risorsa -> effetti dei semafori

Positivi	Negativi
Se usati bene → evitano starvation → assegnando risorse in tempi ragionevoli	Se più processi competono per lo stesso semaforo → waiting time aumenta → specie con sezioni critiche lunghe o risorse limitate

Esame sistemi operativi

time sharing → tecnica di gestione della CPU → alterna processi rapidamente → da illusione simultaneità
uso semafori ha due effetti:

1. efficienza nel contesto dello switching → migliroano gestione del contesto di switch → riduce tempi morti da sync + migliorauso CPU
2. preemption in time-sharing systems → preemption deve gestire corretamente i semafori per evitare di lasciare sezioni critiche incomplete → rischio inconsistenze → tuttavia migliora waiting time

Esame sistemi operativi

3. Analizzare le modalità di soluzione del problema del coordinamento di processi concorrenti in presenza di un sistema reader-writer a due processi. Mostrare la tecnica di eliminazione dei side effects dell'interleaving mediante semafori e provare con pseudocodice specifico come provocare il coordinamento mediante priorità, o mediante altre tecniche di controllo, se conosciute. Non è necessario implementare codice corrispondente.

problema:

- processo legge -> lettori possono leggere insieme -> accesso condiviso
- processo che scrive -> un solo scrittore alla volta -> processo esclusivo

soluzioni:

- priorità ai lettori → scrittori ammessi solo quando 0 lettori attivi
- precedenza a scrittori → lettori ammessi solo quando 0 scrittori attivi
- soluzione effettiva → utilizzo semafori → coordina accesso sicuro a risorsa condivisa → garantisce + reader sync e writer exclusive

  → semafori usati:

  - `read_count`: Conta il numero di lettori attivi.
  - `write_count`: Indica se uno scrittore è attivo.
  - `mutex`: Protegge l'accesso a `read_count`.
  - `wrt`: Garantisce mutua esclusione per gli scrittori.

eliminazione dei side effects dell'interleaving in sistemi concorrentiù

sync accesso risorse condivise → operazioni critiche eseguite atomicamente -> usa semaphore (un solo processo esegue operazioni critiche sulla risorsa) → utilizzo:

- semaphore ‘mutex’ & ‘wrt’ → assicurano che i reader possano leggere contemporaneamente se 0 writer attivi
- read_count → incrementa ogni volta che un lettore inizia a leggere
- (wait(wrt))++ → quando first_reader inizia a leggere → blocca accesso per gli scrittori
- read_count-- → quando reader termina
- (signal(wrt)) →quando l’ultimo reader termina rilascia il blocco per writer

altrimenti:

- priorità ai writers → assegna priorità a writers → evita starvation processi scrittura

Esame sistemi operativi

pseudo-codice coordinamento mediante priorità

```
void reader() {
    while (true) {
        wait(mutex);
        read_count++;
        if (read_count == 1) {
            wait(rw_mutex); // Il primo lettore blocca l'accesso agli scrittori
        }
        signal(mutex);

        // Sezione critica per leggere la risorsa
        read();

        wait(mutex);
        read_count--;
        if (read_count == 0) {
            signal(rw_mutex); // L'ultimo lettore rilascia il blocco
        }
        signal(mutex);
    }
}
```

```
void writer() {
    while (true) {
        wait(rw_mutex); // Lo scrittore blocca l'accesso a lettori e altri scrittori

        // Sezione critica per scrivere la risorsa
        write();

        signal(rw_mutex);
    }
}
```