Concorrenza e Sincronizzazione con Semafori di Dijkstra

Concorrenza

La concorrenza si riferisce alla capacità di più processi o thread di eseguire istruzioni in modo simultaneo all'interno di un sistema. Questa caratteristica è essenziale per sfruttare al massimo le risorse del sistema e migliorare le prestazioni.

Semafori di Dijkstra

- Proposti da Edsger W. Dijkstra, i semafori sono un meccanismo di sincronizzazione utilizzato per gestire l'accesso concorrente alle risorse condivise.
- Un semaforo è una variabile intera che può essere utilizzata per consentire o vietare l'accesso a una risorsa condivisa.
- Due operazioni fondamentali sui semafori sono `wait()` e `signal()`.
  - `wait()` decrementa il valore del semaforo e blocca il processo se il risultato è negativo.
  - `signal()` incrementa il valore del semaforo, potenzialmente risvegliando uno o più processi in attesa.

Semafori Interi

- Oltre ai semafori binari, che possono essere solo 0 o 1, esistono anche semafori interi che possono assumere valori positivi o nulli.
- I semafori interi sono utilizzati per contare le risorse disponibili o il numero di processi che possono accedere a una risorsa condivisa.
- Le operazioni `wait()` e `signal()` sui semafori interi vengono utilizzate per decrementare e incrementare il contatore, rispettivamente.

Sincronizzazione

La sincronizzazione è il processo di coordinamento tra processi o thread per garantire l'ordine corretto delle operazioni e prevenire problemi come la corsa critica e il deadlock.

- I semafori sono uno strumento fondamentale per la sincronizzazione, poiché consentono ai processi di cooperare e comunicare tra loro.

Esempio di Problema Produttore-Consumatore

Il problema del produttore-consumatore è un classico esempio di sincronizzazione.

- Un produttore genera dati e li inserisce in un buffer condiviso.
- Un consumatore preleva i dati dal buffer e li utilizza.
- Utilizzando i semafori, è possibile sincronizzare l'accesso al buffer in modo che il produttore non inserisca dati quando il buffer è pieno e che il consumatore non prelevi dati quando il buffer è vuoto.

Implementazione con Semafori

markdown

Copy code

```
### Produttore ### 1. produci_dato() 2. wait(spazio_disponibile) 3. wait(mutex) 4. inserisci_dato_nel_buffer() 5. signal(mutex) 6. signal(elementi_disponibili) ### Consumatore ### 1. wait(elementi_disponibili) 2. wait(mutex) 3. preleva_dato_dal_buffer() 4. signal(mutex) 5. signal(spazio_disponibile) 6. consuma_dato()
```

Questi sono gli appunti principali sulla concorrenza, i semafori di Dijkstra, i semafori interi e la sincronizzazione.