Sis20241127

Andrea

27 Novembre 2024

Contents

1 rgestione memoria

memoria risorsa limitata, condivisa tra i vari processi.
La memoria è una risorsa limitata, esiste una textbdgerarchia della memoria, gestita dal **gestore della memoria**.
Sulle slide abbiamo tipologie di memoria in base a tempo di accesso e capacità.
Distinguiamo tra due tipi di organizzazione:

- organizzazione logica:
- organizzazione fisica: tener conto a chi è allocato e cosa, risolve problemi tipici come la rilocazione (spostamento dei processi), protezione e implementa condivisione, tra diversi codici aumentando efficienza (es. librerie)

La memoria è divisa tra memoria riservata al sistema operativo e un unico processo utente. Ci sono diversi approcci:
esempio spazio riservato al so può essere indirizzi bassi, oppure indirizzi alti oppure alti e bassi (quindi estremi).
Nei sistemi monoprogrammati la cpu non è sfruttata quando l'unico processo è in attesa di I/O.
Nei sistemi multiprogrammati: avere più processi permette di sfruttare la cpu. Tanti più processi, tanto è maggiore **grado di multiprogrammazione**

calcolo percentuale: $utilizzocpu = 1 - p^n$ permette di conoscere percentuale, p percentuale di tempo in attesa di I/O processo , con n numero di processi (= grado multiprogrammazione). Maggiore è n, minore è p^n, quindi maggiore è l'utilizzo della cpu.
La formula ci dà soltanto un'indicazione, i processi non sono indipendenti. Serve per fare stima di opportunità di upgrade. (slide multiprogrammazione, nell'esempio i processi usano 4MB di memoria).

$Mem = 16, grado = 4, utilizzocpu$

slide (grafico) ovviamente primi miglioramenti hanno un effetto maggiore sulle prestazioni

2 multiprogrammazione

Ogni programma deve essere portato in memoria per essere eseguito.
Uso della coda in input per i processi in attesa di esecuzione.
Sistema operativo dovrà gestire 8(:::)
spazio indirizzo logico, indirizzi delle istruzioni indipendentemente dalla loro allocazione in memoria, ovvero indirizzi relativi all'inizio del codice del programma.
Quando avviene assocazione tra indirizzi logici e fisici? può avvenire in tempi diversi:

- tempo di compilazione, le locazioni di memoria sono note a priori, con indirizzi assoluti. Per spostare tocca ricompilare il codice
- tempo di caricamento: compilatore deve generare **codice rilocabile**, ovvero con codice che viene allocato. NOn si può spostare durante esecuzione.
- tempo di esecuzione: il programma può essere spostato in memoria durante esecuzione, tuttavia per gestire rilocazione serve supporto hardware, ovvero registri base e limite, utili per ricalcolare indirizzi.

(slide) caso binding
si parte con il programma sorgente, viene passato attraverso diversi stadi, alcuni opzionali.
Programma sorgente, rappresentazione indirizzi con indirizzi simbolici
compilatore associa indirizzi rilocabili (es 14 byte dall'inizio di un certo modulo)
poi avviene ricaricamento e collegamento con librerie di sistema
per migliorare caricamento dinamico, ovvero collegamento delle librerie solo al momento dell'esecuzione effettiva in modo da migliorare uso memoria e permettere condivisione. **collegamento dinamico:** ovvvero collegamento a runtime, all'intero del codice vengono messe piccole porzioni per localizzare la routine
stub in caso di esecuzione stub viene rimpiato da (... slide)
Fondamentale separazione tra spazio indirizzo logico e fisico, vengono a coincidere quando avviene binding. Nel caso di binding dinamico serve supporto

hw:
per esempio il **MMU** associazione al runtime gli indiriizi logici e fisici.

2.1 tipo di allocazione

ALLOCAZIONE CONTIGUA, ovvero memoria dedicata a un processo è contigue.
due blocchi, uno riservato al so e uno riservato ai processi utente.
due approcci possibili (almeno):
PARTIZIONAMENTO STATICO: memoria divisa a priori in partizioni, ogni volta che un processo, si alloca una partizione libera al processo.
Le partizioni possono essere tutte uguali oppure diverse, importante è che sia fatto a priori.
Possibile utilizzo di algoritmi.
problema della FRAMMENTAZIONE INTERNA, ovvero dello spazio sarà inutilizzato (all'interno di ogni singola partizione).
per gestire processi in attesa ci può essere una coda oppure due code. Una coda per tutte partizione oppure più code per ogni partizione.
Nel caso di coda singola c'è problema di come distribuire e allocare le partizione, o algoritmo di FIRST FIT (per ogni buco, primo che ci entra) oppure BEST FIT: più grande ci entra, per ridurre sprechi. ma penalizza job piccoli
Nel caso di più code c'è il rischio di un utilizzo non efficiente di memoria, più coda su alcune partizione.

partizionamento dinamico: memoria non è divisa a priori in partizioni, ma si divide in runtime.
In memoria solo il sistema operativo, poi a ogni processo alloco la memoria che gli serve.
vado a riempire così la memoria, il problema è che quando i processi terminano si creano dei buchi che poi devo andare a riempire. // Non si crea frammentazione interna, ma FRAMMENTAZIONE ESTERNA, ovvero regioni vuote tra un processo e l'altro
Tramite RICOMPATTAZIONE posso limitare il problema.
altro aspetto da risolvere è come scegliere regione libera / dove scegliere regione libera.
sloide (..)
best fit: alloco il più piccolo possibile, first fit: primo che ci sta, worst fit: il più grande possibile.
worst fit, alloco più grande, in modo da massimizzare i buchi di scarte, in modo che siano più facilmente riutilizzabili.
i migliori in termini di efficienza sono first fit e next fit, worst fit più lento mentre best fit tende a frammentare

2.2 swapping

permette di spostare da memoria principale a secondaria in maniera momentanea, riportato successivamente per continuare l'esecuzione.
Può essere utile abbassare temporaneamente ill grado di multiprogrammazione, in caso di sovraccarico. per migliorare temporaneamente i tempi di risposta
Gestito dallo scheduler di medio termine.
ROLL OUT, ROLL IN: combina con algoritmi di scheduling a priorità, per permettere accesso ad alta priorità
(... costoso...)
OVERLAY: in caso di programmi molto grandi, si caricavano in memoria solo le parti necessarie per la computazione in ogni istante
programmazione complessa dal punto di vista del programmatore senza supporto particolare dal sistema operativo
soppiantata dalla memoria virtuale, è il sistema operativo che si occupa di caricare in memoria le parti necessarie, non è più responsabilità del programmatore.

3 ALLOCAZIONE NON CONTIGUA

Aspetto importante che viene normalmente utilizzata nei sistemi operativi moderni, alla base della tecnica memoria virtuale.
Si permette che lo spazio di indirizzi riservato a un processo sia non contiguo, spazio indirizzo logico può essere diviso in part divise in parti diverse e non contigue della memoria fisica.
PAGINAZIONE: memoria fisica viene divisa a priori in FRAME, memoria logica in PAGINE. Sono di dimensioni uguali.
Le pagine logiche verranno caricate in memoria principale non in frame contigui.
FRAMMENTAZIONE INTERNA limitata a una pagina per processo (ultima pagina).
COME TRADUCO:
cpu genera indirizzo logico in numero binario che viene spezzato in due parti:
parte dei bit più a destra indicano offset (bit meno significativi) interno della pagina logica individuata dagli m bit più significativi.
offset indica posizione all'interno pagina fisica, mentre m bit più significativi indicano posizione in page table con cui risalire al numero di pagina fisica.
PAGINAZIONE permette condivisione tra più processi allocando condice condiviso inn determinate pagine fisiche condivise.
PROTEZIONE: ogni frame ha un bit di protezione in page table, indica se pagina associata è proprietà del processo.
SEGMENTAZIONE: spazio indirizzo logico diviso in segmenti di dimensioni diverse