sistemi operativi riassunto

Matteo

December 17, 2025

1 processi

Def processo: un attivita controllata da un programma che si svolge su un processore

programma e un entita statica, un processo e dinamico

il processo e l'attualizzazione del programma

assioma del "finite process":ogni processo viene eseguito a una velocita finita, costante ma sconosciuta

Ad ogni istante, un processo può essere totalmente descritto dalle seguenti componenti:

La sua immagine di memoria:

la memoria assegnata al processo (ad es. testo, dati, stack) le strutture dati del S.O. associate al processo (ad es. file aperti) La sua immagine nel processore

la sua immagine nel processore: contenuto dei registri generali e speciali Lo stato di avanzamento

il suo avanzamento del processo: descrive lo stato corrente del processo: ad esempio, se è in esecuzione o in attesa di qualche evento

piu processi possono eseguire lo stesso programma stato dei processi:

- running
- waiting: in attesa di qualche evento
- ready: è pronto ma il processore e impegnato in qulacos'altro

2 introduzione concorrenza

gestione di processi multilpli:

- Multiprogramming: più processi su un solo processore parallelismo apparente
- Multiprocessing: più processi su una macchina con processori multipli parallelismo reale
- Distributed processing: più processi su un insieme di computer distribuiti e indipendenti parallelismo reale

concorrenza: due programmi vengono eseguiti insieme con parallelismo reale o apparente

molte operazioni possono essere create come un insieme di thread concorrenti

interleaving: per ogni processore al massimo un processo alla volta e attivo, multipli processi sono alternati nel tempo

overlapping: piu proceessi sono esguiti contemporeanamente su piu processori, sono alteranti nello spazio

multiprogramming e multiprogramming condividono gli stessi errori derivanti dal fatto che non e possibile predirre gli istanti temporali in cui avvengono le operaziooni e i processi accedono a risorse condivise

se il risultato di una programmazione concorrente dipende dalla temporizzazione dei processi, si dice che quella programmazione contiene una RACE CONDITION

e necessario eliminare la race condition scrivere programmi concorrenti e piu difficile che scrivere programmi sequenziali perche errori possono anche venire da errori nelle iterazioni tra loro

esistono processi concorrenti "ignari l'uno dell'altro", sono indipendenti m concorrono per le stesse risorse, il sistema deve arbitrarle e sincronizzarle. questi non sono creati per lavorare insiem

esisstono processi indirettamente a conoscenza uno dell'altro che non si conoscono per id ma interagiscono indirettamente tramite le risorse condivise, anche loro devono essere ccordinate e sincronizzate

poi ci sono processi direttamente a conoscenza uno dell'altro per id , che comunicano direttamente tramite messaggi, cooperano per certi scopi e informano anche altri processi, il sistema deve fornire meccanismi di comunicazione

la memoria puo essere condivisa o privata

2.1 proprieta della concorrenza

Una proprietà di un programma concorrente è un attributo che rimane vero per ogni possibile storia di esecuzione del programma stesso

esistono 2 tipi di proprieta:

- safety: (nothing bad happens), l'bbiettivo e che se il programma va avanti vada nella direzione giusta
- liveness:(something good eventually happens), il programma non si ferma mai

nei programmi concoorrenti safety significa evitare interferenza tra processsi nelle risorse condivise

liveness significa che la sincronizzazione non deve impedire al programma di andare avanti, cioe non e ammissibile che un processo debba attendere indefinitivamente

2.2 problemi della concorrenza

l'accesso ad una risorsa si dice mutualmente esclusivo se ad ogni istante, al massimo un processo può accedere a quella risorsa

questo risolve il problema della non interferenza ma puo creare quello del deadlock, cio avviene quando dei processi si bloccano a vicenda per un motivo o per l'altro, ad esempio il processo e bloccato perche ha bisogno di usare due processi contemporeanamente ma

la mutua esclusione lo previene, o due processi hanno bisogno di una risorsa occupata dalll'altro.

non interferenza: i processi non devono alterare l'esecuzione degli altri processi in modo non voluto.

starvation invece e quando un processo e fermo perche ha bisogno di una risprsa ma altri processi continuano a saltargli davanti nella coda per la risorsa

azioni atomiche: compiute in modo indivisibile, soddisfano la condizione o tutto o niente, non interferisce con altri processi durante la sua esecuzione

certe azioni vanno eseguite in modo atomico per garantire la non interferenza

le parti di un programma che utilizzano una o piu risorse condivise vengono dette sezioni critiche

3 gestione problemi

3.1 intro

obbiettivi:

- usare solo azioni atomiche nelle sezioni critiche
- evitare blocchi(deadlock,starvation)
- evitare attese inutili, si fa aspettare un processo solo se gli serve una risorsa sezione critica occupata

il sistema non capisce necessariamente da solo cosa e una sezione critica e cosa no quindi devi programmare i processi accordingly (es: se entri in sezione x=x++, e per entrare x deve essere 0) quando entri in una sezione critica devi rispettare 4 proprieta:

- mutua esclusione: max 1 processo attivo che deve accdere a quella risorsa
- assenza di deadlock
- assenza di delay non necessari: Un processo fuori dalla CS non deve ritardare l'ingresso della CS da parte di un altro processo
- eventual entry(assenza di starvation): ogni processo che lo richiede prima o poi accede alla risorsa

se un processo entra in una sezione critica, prima o poi ne uscira, quindi un processo non puo terminare se non fuori da una CS(critical section) si puo provare a gestire i CS tramite:

1. software: cioe si programmano i processsi apposta, rischi di fare errori ed e probabile avere busy waiting
2. hardware: utilizzano istruzioni speciali del linguaggio, progettate apposta, e efficente ma non general purpose
3. ci sono poi approcci basati sul S.O. o sul linguaggio di programmazione(es: semafori o monitor)

3.2 soluzioni software

algoritmo di dekker

algoritmo di peterson(piu semplice)

sono entrambe soluzioni software, funzionano ma sono entrambe baste sul busy waiting, che non andrebbe usato.

3.3 soluzioni hardware

altrimenti si possono usare soluzioni hardware, cioe usare istruzioni speciali per l'hardware

esempio: si possono disabilitare gli interrupt in entrata alla CS e riabilitarli dopo

gli interrupt sono cio che danno il segnale per alternare i processi, senza e fisicamente impossibile non avere mutua esclusione

problemi:

- danneggia il parallelismo
- si lascia ai processi la responsabilita di riattivare gli interrupt che puo generare errori
- ha senso solo su uniprocessori

Le (quasi) sezioni critiche realizzate con istruzioni speciali vengono chiamate spinlock

in generale, i processi hardware:

- possono generalmente essere applicati aqualsiasi numero di processi sia su sistemi mono che multiprocessori
- semplice da verificare
- puo essere utilizzato per sezioni criitiche multiple

  gli svantaggi sono

- ce ancora busy waiting
- non elimina problemi di starvation
- e difficile da programmare

noi invece vorremmo paradigmi poco complessi e facili da implementare.

4 semafori

per avere un paradigma semplice ed efficace, si pensa ai semafori:

due o piu processi si mandano segnaali in modo che un processo rimanga bloccato in un punto specifico fino a quando non riceve un altro segnale

vi sono due operazioni:

v: usata per inviare il segnale

p: usata per attendere il segnale

il semaforo puo essere visto come una variabile intera inizializzato a un numero non negativo

p attende che il semaforo sia positivo e lo decrementa
v aumenta il valore del semaforo
v e p sono entrambe atomiche
struttura dati e coda e ha una politica FIFO, senza s potrebbe avere starvation
non abbaimo eliminato il busy waiting ma lo abbiamo limitato molto