sabato 19 gennaio 2019

Appunti e definizioni SO

Scheduling:

Scheduler: Selezione il processo a cui concedere la risorsa CPU tra i processi presenti nella lista di ready. Esistono due tipologie: l'approccio non-preemptive consente di invocare lo scheduler solo in caso in cui un processo fa una richiesta I/O e passa nello stato di waiting oppure nel caso in cui termina, l'approccio preemtive consente di invocare lo scheduler, oltre ai casi del non-preemptive, anche se il processo in questione non ha ancora effettuato una richiesta I/O e in ready queue c'è un processo che ha priorità rispetto a lui (spesso si usa la versione con il quanto di tempo).

Dispatcher: Modulo che assegna la CPU al processo indicato dallo scheduler. Switcha in modalità kernel e salva il contesto corrente per poi switchare in user mode facendo il Restore del processo scelto. Questo switch ovviamente spreca tempo e inoltre lo switch tra processi rende invalida la cache poiché i dati presenti al momento dello switch non sono del processo scelto.

Metriche di valutazione dello scheduler: UTILIZZO CPU indica frazione di tempo in cui la CPU è utilizzata dai processi, TURNAROUND TIME è il tempo di completamento di un processo, THROUGHPUT è il numero di processi completati nell'unità di tempo, WAITING TIME è il tempo di attesa totale in cui il processo è rimasto nella ready queue, RESPONSE TIME è il tempo che passa tra la richiesta I/O che un processo effettua e la ripresa della CPU. Bisogna massimizzare le prime due e minimizzare le ultime tre.

First Come First Served (FCFS) (Non-preemptive): Viene selezionato il primo processo nella ready queue, alla fine di una richiesta I/O il processo rientra alla fine della ready queue.

Shortest Job First (SJF) (Non-preemptive): Verrà scelto il processo il cui successivo CPU-Burst sarà il più breve. Si mantiene ottimale il tempo medio di attesa e si massimizza il throughput. Il problema è che bisogna prevedere o sapere il dato successivo sul tempo di CPU-Burst di ogni processo nella ready queue per poterli ordinare. Spesso si usano formule statistiche per prevedere il dato, non sempre però la misura è accettabile.

Priority Scheduler (Non-preemptive): Ad ogni processo è assegnato un intero di priorità con la regola che a valori più bassi dell'intero corrispondono valori più alti di priorità. È una sorta di SJF basata non sul valore del CPU-Burst ma sull'intero di priorità. Problema starvation, soluzione aging ossia ad ogni processo viene aumentato l'intero di priorità ogni volta che non vengono scelti.

Preemptive approach: Ogni scheduler sopra descritto può essere usato con un approccio preemptive. Ciò che si aggiunge all'algoritmo è un quanto di tempo scelto a priori, il cui scopo è interrompere un CPU-Burst di un processo nel caso in cui quest'ultimo dovesse avere maggior durata rispetto al quanto. Nel caso si schedulasse un altro processo prima della fine del quanto esso verrebbe resettato.

Round Robin (Preemptive): È praticamente uguale al FCFS con l'aggiunta del quanto di tempo. Allo scadere del quanto il processo è messo in coda alla ready queue.

Multilevel Queue: In realtà esistono diversi tipi di processi, compiono infatti diversi task. Si è optato quindi ad una ready queue divisa in code a livelli di priorità. Ogni coda fa riferimento al proprio algoritmo di scheduling. A loro volta però anche le code devono essere schedulate. Ad ogni processo è assegnata una ed una sola coda di ready.

Multilevel Feedback Queue: Qui invece un processo può muoversi all'interno delle code. Ovviamente bisogna stabilire i parametri per cui un processi può aumentare o diminuire il suo livello di priorità spostandosi tra le varie ready queues.

Real Time CPU Scheduling: I sistemi soft real-time non garantiscono quando un processo verrà schedulato invece i sistemi hard real-time garantiscono che i task vengano serviti alla propria deadline.

Real Time Priority Scheduling: Per gli scheduler di sistemi soft real-time devono supportare l'approccio preemptive basato su priorità, per quelli di sistemi hard reat-time lo scheduler deve essere capace di gestire le deadline dei processi. I processi hanno una nuova caratteristica: la PERIODICITÀ p con cui richiedono l'uso della CPU. Hanno processing time t >= 0 e <= d ossia la DEADLINE a sua volta <= della PERIODICITÀ p.

LITTLE's Formula: Il senso è che in sistemi regolari i processi che arrivano devono essere uguali a quelli che lasciano la ready queue nel tempo => $n = \lambda \times W$. Con n che indica il numero medio di processi presenti nella ready queue, λ indica il numero di processi che nell'unità di tempo in media arriva nella ready queue, W indica il tempo medio di waiting dei processi.

Main Memory:

Starting a program file: In primis viene decodificato il program file header in cui vengono lette informazioni riguardanti i frammenti di memoria e le librerie dinamiche necessarie. In seguito il dynamic linker pensa a sistemare le librerie necessarie nella memoria del processo.

Allocation: Partizione di memoria che da a ogni processo ciò di cui ha bisogno.

Problemi comuni di allocazione: La memoria spesso deve cambiare dinamicamente durante la vita di un processo, ad esempio l'heap e la stack che devono avere un size variabile. Inoltre vanno analizzati anche i problemi di frammentazione, interna nel caso in cui viene assegnata una size fissa di memoria e quest'ultima non viene completamente utilizzata, o esterna nel caso vengono allocati blocchi di size variabili ma può capitare che una richiesta non può essere soddisfatta poiché la memoria libera non è contigua nonostante in totale teoricamente si può soddisfare la richiesta. Infine ci sono problemi di protezione in cui aree di memoria assegnati a processi differenti devono essere accessibili solo dai loro possessori.

Protezione: Per assicurare protezione ogni qual volta la CPU genera un indirizzo viene logicamente verificato tramite una combinazione di base e limite (>= base && < base+limit). Nel caso l'indirizzo non fosse valido allora viene lanciata un'eccezione.

Logical and Physical Addresses: Nasce così il concetto di spazio di indirizzamento logico e fisico. Quello logico è uguale per ogni processo, quello fisico è in comune. Il senso di ciò è rendere facile l'uso da parte dell'User della memoria e anche semplificare il lavoro della CPU, a cui viene nascosta la memoria fisica dal modulo MMU il quale converte gli indirizzi da logici a fisici.

Relocation and Limit Registers: Ecco quindi che ogni volta che la CPU genera un indirizzo lo verifica con il registro limite affinché esso sia valido e tramite il registro relocation riesce a mappare ogni indirizzo logico in uno fisico.

Contiguous Allocation: Il senso generale è che l'MMU svolge il compito di verificare e di mappare indirizzi logici dei processi in aree di memoria fisica differente. Questo è supportato dal fatto che ad ogni processo viene assegnato memoria contigua (insieme di indirizzi successivi racchiusi in frame).

Segmentation: All'inizio i primi processori avevano uno spazio di memoria mappatile attraverso un numero di bit più grande del massimo dei registri interni. Come si potevano mappare tutti gli indirizzi? Il senso è che i bit più significativi venivano conservati in un registro esterno. Successivamente i registri di segmentazione vennero spostati in RAM in una tabella di segmentazione che permisero di aumentarne in numero rendendo molto simile il layout della memoria fisica a quello della memoria logica. Ogni indirizzo logico è formato da una tupla composta da l'indice della tabella dei segmenti e l'offset all'interno di esso. Ogni volta che viene generato un indirizzo per validarlo si accede alla cella i-esima della tabella si legge il valore del registro limite e lo si confronta con l'offset, se risulta minore l'offset allora si procede a sommarci al registro base della cella i-esima sopra citata. Così si ottiene l'indirizzo fisico o nel caso ci sia paginazione l'indice della pagina corrispondente. Inoltre ogni cella della tabella dei segmenti ha un bit di validazione come protezione e vari bit che indicano i privilegi dell'area di memoria.

Paging: Si divide la memoria fisica in frame di size prefissata e si mappa un frame in una singola pagina. La tabella delle proprie pagine è propria di ogni processo che fa riferimento a una tabella delle pagine in main memory ma i valori di ogni cella i-esima di quest'ultiuma corrispondo ad indici di frame fisico. Una volta ottenuto dall'inidrizzo logico l'indice della tabella delle pagine si legge il corrispondente indice di frame a cui si somma lo spiazzamento dell'indirizzo logico per ottenere l'indirizzo fisico corrispondente. I pro sono che non c'è frammentazione esterna mentre i contro che può esserci frammentazione interna e nel caso si conservasse la tabella delle pagine in RAM la mappatura degli indirizzi richiederebbe molto tempo di computazione.

TLB: Funzione di cache per la tabella delle pagine, memoria associativa nella CPU. Numero entries minore nel numero massimo delle pagine. In primis si controlla nel TLB se la pagina i-esima è presente, nel caso si mappa direttamente l'indirizzo fisico, altrimenti si deve accedere alla tabella delle pagine in main memory, leggere il frame corrispondente, accedere per leggere/scrivere il dato e successivamente salvare la nuova pagina letta nel TLB.

EAT (Effective access time): pf(2TTLB + 2TRAM) + (1-pf)(TTLB + TRAM)

Protection Paging: La protezione del paging si effettua come quella dei segmenti, tramite bit che indicano la validità del frame corrispondente e i privilegi dell'aree di memoria indicate.

Shared Pages: Esistono frame che possono essere condivisi da processi essendo puntati da più entries di pagine di processi differenti. C'è da dire che in realtà ogni processo conserva serve una copia di backup dei dati affinché si evitino conflitti sui dati. Questi backup ovviamente sono mappati su frame differenti.

Hierarchical Page Tables: Semplicemente si possono mettere in gerarchia sempre più specifica più tabelle delle pagine, magari per differenziare zone di memoria macroscopiche fino a scendere nel particolare.

Swapping: In alcuni casi un processo può essere conservato in disco e quindi rimosso dalla memoria RAM per motivi di scheduling o spazio. Questo processo spesso è disabilitato di default ma viene attivato nel momento in cui si cerca di allocare memoria per un nuovo processo senza successo a causa della RAM satura. Cosi si switchano due processi: uno stored in disco e un altro caricato in RAM. Questo processo è molto lento rispetto agli altri e si cerca di evitarlo il più possibile. C'è da notare che nel Restore di un processo spesso gli indirizzi logici vengono mappati in differenti indirizzi fisici.

Virtual memory:

Hint: Raramente il codice ha bisogno di tanta memoria, spesso è poca memoria alla volta che serve all'esecuzione del processo. Si cerca quindi di eseguire programmi parzialmente caricati: sarà possibile quindi runnare più processi contemporaneamente e si incrementa l'utilizzo della CPU e il throughput.

Virtual Address Space: All'interno sono contenute tutte le variabili globali, il codice del programma e altre informazioni. Seguendo l'hint iniziale si è deciso che all'interno dello spazio virtuale sono presenti stack e heap che crescono dinamicamente durante l'esecuzione del programma. Al loro aumentare vengono dinamicamente richieste nuove pagine al SO. La stack cresce verso il basso ossia da indirizzi più grandi a quelli più bassi, al contrario l'heap. Lo spazio inutilizzato dello spazio virtuale semplicemente non viene mappato in memoria fisica risparmiando spreco di memoria! NB ogni processo ha a disposizione sempre lo spazio massimo virtuale di indirizzamento.

Demand Paging: Così si è passato a richiedere una pagina solo in cui è strettamente necessario. Ci sono meno operazione di I/O, si usa meno memoria, più utenti contemporaneamente, sistema più responsive. È simile al sistema di paging con servizio di swap. Se la pagina è necessaria allora si referenzia, se la pagina è invalida si abortisce la richiesta, se la pagina non è presente in memoria la si carica. L'estremo sarebbe far partire un processo senza caricare nessuna pagina iniziale, questo porterebbe subito ad un page fault. Spesso però per il principio di località spaziale si preferisce portarsi dietro più pagine a partire da quella richiesta, prevedendo che potrebbe essere a breve richiesta. Per avere questo approccio abbiamo bisogno bit di validità per la tabella delle pagine, una memoria secondaria adibita a swap space, istruzione restart.

EAT of Demand Paging: EAT = (1-p)(T memory access) + (p)(page fault overhead + swap in time).

Demand Paging Optimizations: Si usa uno swap space che è un'area del disco senza filesystem, così da rendere più veloce l'interazione con la memoria tralasciando l'overhead del fs. On startup si carica tutta l'immagine del processo nella swap area, in esecuzione invece vengono effettuate operazioni di swap in e swap out. Inoltre si adopera una soluzione COPY ON WRITE, ossia durante un'esecuzione della syscall fork() si replica nel figlio esclusivamente la tabella delle pagine affinché puntino agli stessi frame di memoria fisica. Quando si fa la fork un bit viene gettato a 1 per ogni pagina, è un bit di trap_on_write il quale segnala all'SO che quella pagina è condivisa in lettura da processo padre e da processo figlio. Nel caso in cui uno dei due dovesse aver la necessità di modificare il contenuto di quell'area, solo al momento della scrittura viene effettuata una copia di quella

pagina, mappandola in un altro frame, aggiornando ovviamente la tabella delle pagine relativa al processo scrittore. Si risparmia un sacco di tempo di esecuzione della syscall non dovendo obbligatoriamente copiare tutta la memoria. Inoltre l'SO mantiene a mo di SLAB una lista dei frame liberi affinché la getBlock venga effettuata in O(1). Per sicurezza un frame liberato viene azzerato dall'os.

Page Replacement: Bisogna capire la politica di sostituzione delle pagine nel caso in cui tutti i frame siano occupati. Ottimale sarebbe poter eliminare la pagina che sarà accesso più tardi possibile ma avremmo bisogno di conoscere il futuro. Pagine che non vengono modificate in scrittura non hanno bisogno di essere ricopiate in disco. Per cui si usa un bit "dirty" per indicare se una pagina è stata modificata.

FIFO Algorithm: Si sceglie come vittima la pagina che è stata caricata meno recentemente. Attenzione soffre di Anomalia di Belady.

Optimal Page Replacement: Si sceglie come vittima quella che non sarà usata per più tempo. Richiede la conoscenza del futuro. Però ci da un Upper Bound per le prestazione degli algoritmi. No Anomalia di Belady.

LRU Page Replacement: Si approssima l'algoritmo ottimale prevedendo il blocco che sarà usato per ultimo. Si basa sulle ripetizioni di accesso sulle pagine. La prossima vittima è quella che non è usata da più tempo tra quelle caricate

Implementazioni LRU: Si associa ad ogni pagina un bit di referenza. All'inizia posto a 0 per tutti. Ogni volta che viene referenziata una pagina il bit in questione viene settato a 1. Il Second-Chance Algorithm o Clock Replacement Algorithm funziona così: Si crea una coda circolare delle pagine, si una un puntatore che scorre ciclicamente la coda in questione. Ogni volta che incontra una pagina con bit di referenza a 1 lo azzera. Appena trova una pagina con bit a 0 usa la pagina come vittima e ferma la sua ricerca. Enhanced Second- Chance Algorithm fa usa di un altro bit che indica se quella pagina è stata modificata. Funziona sempre nello stesso modo solo che le priorità cambiano ossia se il bit di referenza è 0 e quello di modifica anche allora la pagina individuale è la perfetta candidata. Altrimenti se fossero entrambi a 1 si dereferenzia la pagina e si continua la ricerca.

Trashing: Fenomeno per cui il sistema di swapping va in stallo poichè si forma un ping pong di swap in e swap out di pagine che servono ad intervalli regolari. L'unica soluzione è killare qualche processo affinché si faccia spazio nella memoria e si permetta all'os di completare pochi task precedenti non dovendo ogni volta swappare le pagine.

Working Set qModel: la funzione wss(t,p) è l'insieme delle pagine usate dal processo p nell'intervallo di tempo t. D(t) è la sommatoria dei wss di tutti i processi. Nel caso fosse maggiore dei frame disponibili si verifica il fenomeno di trashing.