Sistemi Operativi

Ionut Zbirciog

2 November 2023

Introduzione allo Scheduling

In un sistema multiprogrammato molteplici processi e thread competono per la CPU, quindi è necessario scegliere a quale processo o thread assegnare la CPU. La parte del sistema operativo che effettua lo scheduling è detto *scheduler* e l'algoritmo che utilizza si chiama algoritmo di scheduling.

Molti problemi di scheduling per processi valgono anche per thread, nel caso in cui i thread sono gestiti dal kernel.

Nei sistemi batch storici, lo scheduling era lineare, ovvero i job venivano eseguiti in modo sequenziale. Con la multiprogrammazione, lo scheduling è diventato complessi a causa della concorrenza tra utenti.

Costi in termini di tempo del scheduling. Lo scambio di processi (*context switch*) è oneroso. - Cambio da modalità utente a modalità kernel. - Salvataggio dello stato del processo. - Esecuzione dell'algoritmo di scheduling. - Aggiornamento della MMU con la nuova mappa della memoria. - Potenziale invalidazione della memoria cache. Tutte queste operazioni possono consumare tempo alla CPU.

Problema di Scheduling dei Processi

Obiettivi:

- Equità - garantire un'equa condivisione della CPU a tutti i processi
- Imposizione della policy - garantire l'attuazione delle policy dichiarate
- Bilanciamento - mantenere tutti i componenti del sistema attivi

I processi alternano fasi di elaborazione CPU-intense con richieste di I/O. In genere un processo utilizza la CPU per un certo periodo senza interrompersi, poi fa una chiamata di sistema per leggere da un file o scrivere un file. Quando la chiamata di sistema è completa, riprende a usare la CPU fino a quando ha bisogno di leggere o di scrivere altri dati, e così via. Quando la CPU copia/preleva dati dalla RAM si parla di elaborazione e non di I/O.

Ci sono due tipologie di processi: 1. Processi *Compute-bound* (CPU-bound): Burst di CPU lunghi, attese di I/O infrequenti, quindi passano molto tempo nella CPU. 2. Processi I/O-bound: Burst di CPU brevi, attese di I/O frequenti. Sono tali a causa della bassa necessità di calcoli, non della durata delle richieste di I/O.

Quando Effettuare lo Scheduling

- **Creazione Nuovo Processo:** Lo scheduler deve decidere quale processo tra figlio e padre scegliere per l'esecuzione. Essendo entrambi in stato di ready, si tratta di una normale decisione di scheduling, che può andare bene in entrambe le direzioni.
- **Uscita di un Processo:** Se un processo esce, occorre scegliere un altro dai processi pronti. Se nessuno è pronto, occorre eseguire un processo inattivo del sistema.
- **Blocco di un Processo:** Se un processo si blocca (I/O, semaforo, etc...), occorre selezionarne un altro.
- **Interrupt di I/O:** Una decisione di scheduling va presa quando si verifica un interrupt di I/O. Se l'interrupt proveniva da un dispositivo di I/O che adesso ha terminato il lavoro, qualche processo bloccato in attesa del dispositivo di I/O potrebbe ora essere pronto a partire. Sta allo scheduler stabilire se eseguire il processo che è appena diventato pronto, quello che era in esecuzione al momento dell'interrupt o qualche altro processo.

Tipologie di Algoritmi Scheduling e Prelazione

- **Non Preemptive (Senza Prelazione):** Sceglie un processo e lo lascia eseguire fino a quando si blocca o rilascia volontariamente la CPU. Anche se è eseguito per ore, non sarà sospeso forzatamente. In effetti durante gli interrupt del clock non vengono prese decisioni di scheduling. Dopo il completamento dell'elaborazione dell'interrupt del clock viene ripristinato il processo che era in esecuzione prima dell'interrupt, a meno che un processo di priorità più alta sia in attesa di un timeout ora soddisfatto.
- **Preemptive (Con Prelazione):** Sceglie un processo e lo lascia girare per un tempo massimo prefissato. Se alla fine dell'intervallo di tempo è ancora in esecuzione, il processo è sospeso e lo scheduler ne sceglie un altro da eseguire (se è disponibile). Uno scheduling con prelazione richiede che vi sia un interrupt del clock alla fine dell'intervallo per restituire il controllo della CPU allo scheduler. Se non è disponibile il clock, lo scheduling senza prelazione rimane l'unica possibilità. La prelazione non è rilevante solo per le applicazioni, ma anche per i kernel dei sistemi operativi, specie se monolitici. Oggi molti di essi hanno la prelazione; se non l'avessero, un driver male implementato o una chiamata di sistema molto lenta potrebbero intasare la CPU. In un kernel con prelazione, invece, lo scheduler può forzare uno scambio di contesto per il driver o la chiamata troppo lenti.

Scheduling nei Sistemi Batch

I sistemi batch sono ancora molto utilizzati in attività aziendali periodiche come elaborazione di paghe, inventari, ecc. Nei sistemi batch non ci sono utenti impazienti al loro terminale in attesa di una risposta rapida a una breve richiesta, quindi sono spesso accettabili algoritmi senza prelazione o algoritmi con prelazione con lunghi periodi per ciascun processo. Questo approccio riduce gli scambi di processo e migliora così le prestazioni. Gli algoritmi batch sono effettivamente abbastanza generali e spesso applicabili anche ad altre situazioni.

Obiettivi

- **Throughput:** massimizzare il numero di job per ora
- **Tempo di turnaround:** ridurre al minimo il tempo dallo start al termine di un job
- **Utilizzo della CPU:** mantenere la CPU sempre impegnata

Algoritmi di Scheduling nei Sistemi Batch

(a) First-Come, First-Served

È un algoritmo di scheduling senza prelazione. I processi vengono assegnati alla CPU nell'ordine in cui arrivano. Viene usata una singola coda di processi in stato di pronto, il primo job esegue immediatamente senza interruzioni. Quando il processo in esecuzione si blocca, viene eseguito il prossimo, quando il processo torna in stato di pronto, viene posto in fondo alla coda.

Vantaggi: È un algoritmo facile da capire e da implementare, basta una singola linked list. Inoltre, è equo in base all'ordine di arrivo dei processi; il processo da eseguire è posizionato in testa alla lista, gli altri processi sono aggiunti in coda in base all'ordine di arrivo.
Svantaggio: Può risultare in tempi di attesa molto lunghi per processi I/O-bound in presenza di un processo CPU-bound.

(b) Shortest Job First

È un algoritmo di scheduling senza prelazione. Si suppone che i tempi di esecuzione siano noti in anticipo, ad esempio dopo una previsione. Lo scheduler preleva il job più breve. Se eseguiti

Figure 1: Shortest Job First

nell'ordine A, B, C, D la media di esecuzione è di 14 min, mentre se eseguiti con SJF, nell'ordine B, C, D, A la media di esecuzione è di 11 min.

Vantaggio: L'algoritmo è ottimale nel minimizzare il tempo di turnaround medio quando tutti i job sono disponibili contemporaneamente.
Svantaggio: Se i job arrivano in momenti diversi, SFJ potrebbe non essere ottimale.

(c) Shortest Remaining Time Next

È una versione con prelazione dell'algoritmo SJF. Con questo algoritmo, lo scheduler sceglie sempre il processo che impiegherà meno tempo per terminare l'esecuzione. Anche in questo caso, il tempo di esecuzione di ciascun processo deve essere noto in anticipo. All'arrivo di un nuovo job, il suo tempo totale è confrontato al tempo restante dei processi attuali. Se il nuovo job richiede meno tempo del processo attuale per terminare, il processo attuale viene sospeso ed è avviato il nuovo job.

Scheduling nei Sistemi Interattivi

In un ambiente con utenti interattivi, l'uso della prelazione è essenziale per evitare che un processo monopolizzi la CPU e neghi il servizio agli altri. Anche se intenzionalmente nessun processo viene eseguito all'infinito, un processo potrebbe escludere tutti gli altri a tempo indeterminato per un errore di programmazione. La prelazione è necessaria per prevenire questi comportamenti. Anche i server rientrano in questa categoria, dato che normalmente servono molti utenti (remoti), e tutti sempre di fretta.

Obiettivi

- **Tempo di risposta:** risposte rapide alle richieste dell'utente
- **Adeguatezza:** soddisfare le aspettative dell'utente in termini di tempi di risposta

Algoritmi di Scheduling nei Sistemi Interattivi

(a) Round-Robin Scheduling

È uno degli algoritmi di scheduling più vecchi, semplici, equi e utilizzati. Ogni processo riceve un intervallo di tempo detto "quanto" per l'esecuzione. Se il processo non ha terminato l'esecuzione, la CPU viene prelazionata per un altro processo. Se un processo si blocca, il passaggio avviene automaticamente. L'algoritmo è facile da implementare, infatti lo scheduler deve mantenere una lista di processi eseguibili, quando il processo esaurisce il suo quanto di tempo viene posto in fondo alla lista. **Problema:** L'unico problema del Round-Robin è la durata del "quanto". Supponendo

Figure 2: Round - Robin

un 1ms per il cambio di contesto e 4ms per il quanto, il 20% del tempo CPU è sprecato in overhead. Un quanto tra 20 e 50 ms è spesso ragionevole per bilanciare efficienza e reattività.

(b) Priority Scheduling

A ciascun processo è assegnata una priorità, e l'esecuzione è consentita al processo eseguibile con la priorità più alta. Per impedire che i processi siano eseguiti indefinitamente, lo scheduler può abbasare la priorità del processo in esecuzione ad ogni interrupt, se scende sotto una soglie, avviene lo scambio con un altro processo. Inoltre c'è la possiblità di assegnare un quanto di tempo per l'esecuzione del processo. Infine, è necessario avere un algoritmo che ne aumenti le priorità altrimenti, potrebbero finire a priorità 0.

Le priorità possono essere assegnare in modo statico, per esempio il costo computazionale in un data center, un utente che paga 50$ ha una priorità diversa rispetto ad un utente che paga 100$. Oppure possono essere assegnare in modo dinamico, basato sull'utilizzo della CPU o sul comportamento I/O-bound.

Spesso conviene ragruppare in processi in classi di priorità e usare lo scheduling a priorità tra le classi e all'interno delle classi usare Round-Robin.

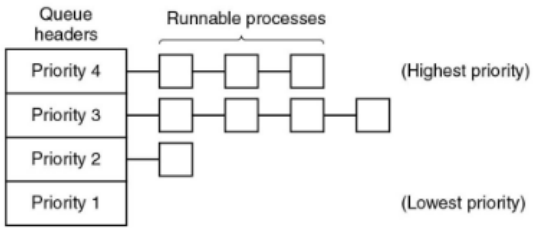

Figure 3: Priority Classes

Nell'immagine un sistema con 4 classi di priorità. Fintanto ci sono processi in prioritò 4, si usa Round-Robin. Se vuota si passa alla 3 poi alla 2 e cosi via. È bene rivedere periodicamente le priorità, altrimenti i processi nelle classi di priorità inferiore rischiano di "morire d'inedia".

(c) Shortest Process Next

Nei sistemi batch, SJF produce sempre il minor tempo medio di risposta, quindi sarebbe comodo poterlo usare anche sui sistemi interattivi. La soluzione è fare stime basate sul comportamento passato (*Aging*). Supponiamo che il tempo stimato per comando per un certo terminale sia T_0. Dopo una nuova esecuzione, T_1, diventa $aT_0 + (1-a)T_1$. Tramite la scelta di a, possiamo decidere se il processo di stima debba dimenticare in breve tempo le ultime esecuzioni o ricordarle a lungo. Con $a = \frac{1}{2}$ otteniamo:

$$T_0, \frac{T_0}{2} + \frac{T_1}{2}, \frac{T_0}{4} + \frac{T_1}{4} + \frac{T_2}{2}, \frac{T_0}{8} + \frac{T_1}{8} + \frac{T_2}{4} + \frac{T_3}{2},$$

dopo 3 esecuzioni il peso di T_0 è 1/8. Questa tecnica è facile da applicare, specialmente con $a = \frac{1}{2}$.

(d) Guaranteed Scheduling

L'idea di base è fare promesse concrete sugli standard di prestazione e rispettarle. Se ci sono n utenti o processi, ciascuno ottiene $\sim \frac{1}{n}$ della potenza della CPU. Il sistema tiene traccia di quanta CPU ha ricevuto ogni processo dal momento della sua creazione (per esempio, 100s). Calcola quanto tempo CPU ogni processo dovrebbe avere: tempo creazione / n. Valuta il rapporto tra il tempo CPU consumato e quello dovuto. Esegue il processo con il rapporto più basso finché non supera il suo concorrente più vicino.

(e) Lottery Scheduling

L'idea di base è quella di dare ai processi un biglietto della lotteria per le diverse risorse del sistema, come il tempo della CPU. Ogni volta che si deve prendere una decisione di scheduling, viene estratto

un biglietto per decidere quale processo ottiene la risorsa. Ai processi più importanti possono essere assegnati dei biglietti extra, per aumentare le loro possibilità di vincita.

Alcune proprietà:

- Reatiività: Risponde velocemente ai nuovi processi grazie alla distribuzione dei biglietti.
- Cooperazione tra processi: il processi possono scambiarsi biglietti tra di loro.
- E' adatto a situazioni dove altri metodi falliscono, per esempio un server video con diverse necessità di frequenze di fotogrammi.

(f) Fair-Share Scheduling

Tradizionalmente, ogni processo è oggetto di scheduling individualmente, senza considerare a chi appartenesse. Di conseguenza, se l'utente 1 ha 9 processi e l'utente 2 ne avvia 1, con il Round-Robin o le priorità uguali, l'utente 1 si prenderà il 90% della CPU e l'utente 2 solo il 10%. Per evitare questa situazione, alcuni sistemi considerano la proprietà di ciascun processo prima di considerarlo. Ogni utente riceve una frazione predefinita di CPU. Lo scheduler si assicura che ogni utente riceva la sua frazione, indipendentemente dal numero di processi posseduti.

Ad esempio, si consideri l'utente 1 con 4 processi A, B, C, D e l'utente 2 con 1 processo E. Una possibile sequenza con il Round-Robin è: A E B E C E D E..., mentre se l'utente 1 ha il doppio del tempo di CPU rispetto all'utente 2, otteniamo A B E C D E A B E...

Scheduling nei Sistemi Real-Time

In sistemi con vincoli real-time, la prelazione è, stranamente, non sempre necessaria, poiché i processi sanno che non possono essere eseguiti per lunghi periodi di tempo e generalmente fanno il loro lavoro e si bloccano rapidamente. Essi eseguono programmi per specifiche applicazioni, a differenza dei sistemi interattivi che possono eseguire programmi arbitrari.

Obiettivi:

- Rispetto delle scadenze - assicurarsi che i dati vengano elaborati nei tempi previsti
- Prevedibilità - assicurarsi che il funzionamento sia constante, specialmente in sistemi multimediali per evitare degradi della qualità

E' usato nei sistemi operativi in applicazioni in cui il tempo di risposta è fondamentale, per esempio in lettori musicali, monitoraggio in terapia intensiva, piloti automatici, controllo robotico in fabbriche, dove ritardi o mancati tempi di risposta possono avere gravi implicazioni.

Sono divisi generalmente in 2 categorie.

Hard Real-Time, nel caso di scadenze assolute da rispettare, **Soft Real-Time**, se una scadenza mancata di tanto in tanto è indesiderabile, ma c'è un certo grado di tollerabilità. In entrambi i casi il comportamento real-time si ottiene suddividendo il programma in un certo numero di processi, ognuno dei quali ha un comportamento prevedibile e noto in anticipo.

Gli eventi cui un sistema real-time può dover rispondere sono di tipo periodico, ovvero avvengono a intervalli regolari, oppure non periodico, ovvero avvengono in modo imprevedibile. Ad esempio,

se ci sono m eventi periodici, l'evento i avviene con un periodo P_i e richiede C_i secondi di tempo della CPU per gestire ogni evento, allora il carico può essere gestito solo se:

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

Per esempio, consideriamo 3 eventi periodici con periodi di 100ms, 200ms, 500ms e tempi richiesti per la CPU di 50ms, 30ms, 100ms. Allora abbiamo $0.5 + 0.15 + 0.2 \leq 1$, quindi il sistema è schedulabile.

Gli algoritmi di scheduling real-time possono essere statici o dinamici. I primi prendono le loro decisioni di scheduling prima che il sistema inizi l'esecuzione, i secondi durante l'esecuzione. Lo scheduling statico funziona solo dove è disponibile in anticipo un'informazione perfetta riguardo al lavoro da svolgere e le scadenze da rispettare. Gli algoritmi di scheduling dinamico non hanno queste restrizioni.

Scheduling di Thread

Un processo può avere molti processi figli sotto il suo controllo. È del tutto possibile che il processo principale abbia un'idea eccellente di quale dei suoi figli sia il più importante e quale meno. Finora, gli algoritmi di scheduling non accettano input dai processi utenti riguardo alle decisioni di scheduling, spesso portando a decisioni sub-ottimali. La soluzione a questo problema è di dividere il meccanismo di scheduling dalla politica di scheduling, ciò significa che l'algoritmo di scheduling è in qualche modo parametrizzato, ma i parametri possono essere forniti dai processi utente. Per esempio, supponiamo un kernel con algoritmo di scheduling a priorità. Il kernel mette a disposizione una chiamata di sistema che permette a un processo di impostare le priorità dei suoi figli. In questo modo, il genitore può influenzare lo scheduling dei suoi figli senza controllarlo direttamente. In conclusione, il meccanismo sta nel kernel, mentre le policy sono definite dal processo utente.

Lo scheduling differisce in base al tipo di thread, se sono thread a livello utente o a livello kernel.

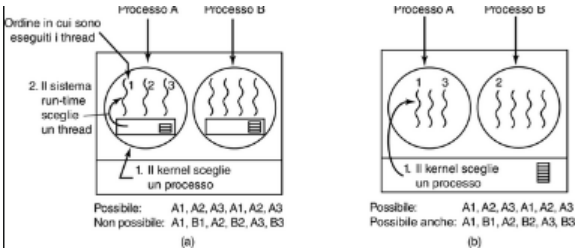

Figure 4: User Threads and Kernel Threads

- **Thread a livello utente:** il kernel ignora l'esistenza dei thread; sceglie un processo per il suo quanto. Lo scheduler interno al processo decide quale thread eseguire senza interruzioni del clock. Questo porta ad avere thread che possono consumare l'intero quanto del processo, influenzando solo il processo interno e non gli altri. Lo scambio dei thread avviene con poche istruzioni, dato che ci pensa il processo al suo interno. Un thread utente, se richiede I/O, sospende l'intero processo.

- **Thread a livello kernel:** il kernel seleziona un thread specifico per l'esecuzione; se un thread eccede il quanto, viene sospeso. Lo scambio del thread comporta uno scambio di contesto, quindi è più lento. Un thread kernel, se richiede I/O, non sospende tutto il processo ma solo il thread stesso.