Thread

Due o più thread possono essere eseguiti contemporaneamente se abbiamo una CPU multithread ma se uno di essi cerca di accedere o modificare una variabile comune ad entrambi avviene un conflitto, le possibili soluzioni possono essere:

- Vietare variabili globali (radicale mai usato)
- Usare solo variabili globali private (copie private) che hanno un maggior spazio di memoria

Thread in Linux

Linux alloca lo stesso tipo di descrittore di processo per processi e thread (tasks come per esempio: task_struct che è un descrittore)usa la chiaamta di sistema basata su UNIX (fork) per generare task figli e per abilitare il thread, Linux fornisce una versione modificata (clone) che accetta argomenti che specificano quali risorse condividere con il task figlio

Thread in Windows

I processi contengono i programmi, dei riferimenti ad oggetti (handle) e i thread con cui condividono risorse:

- Thread: unità corrente di esecuzione assegnata ad un processore, che esegue un pezzo di codice del processo nel contesto del processo, utilizzando le risorse del processo
- Il contesto di esecuzione, che contiene:
  - Runtime stack
  - Stato dei registri della macchina
  - Molti attributi
  - Unità reale di esecuzione ad un processo

Ed è composto da due componenti: PEB (Process Enviroment Block) e TEB (Thread Enviroment Block)

Windows può raggruppare i processi in **Job** per limitare e gestire l'uso delle risorse per tutti i thread del processo del Job contengono i processi stessi

I thread possono creare e **fiber** (viene eseguito nel contesto del thread che lo crea, invece che lo scheduler)allocando strutture dati e uno stack a livello utente, i thread possono essere convertiti in fiber, le fiber possono essere

create indipendentemente dai thread e il vantaggio di usarli è che per il cambio di contesto avviene solo a livello utente

La relazione thread->fiber è di molti-a-molti, ma di regola un thread è associabile a un insieme di fiber e non sempre vengono usate

Windows fornisce ad ogni processo un pool di thread che si compongono di un numero di thread worker, che sono thread di livello kernel che eseguono funzioni previste dal thread utente

Thread Pool: è una funzionalità di Win32 e consiste nell'avere una coda di task da eseguire e quindi vengono creati dei thread di default che vengono messi dentro a questa pool e appena sono liberi prendono un task da eseguire dalla coda, questo evita la creazione/distruzione continua di thread, tali thread possono bloccarsi in attesa di eventi e in quella fase non possono essere riassegnati ad un altro task da eseguire, per esempio: in applicazioni di tipo cliente-servente

Scheduling

Lo scheduling è un algoritmo che si occupa di amministrare l'esecuzione dei processi nella CPU, possiamo distinguerli di due tipi a seconda della loro funzione:

- Un processo CPU-bound (utilizzano più la CPU) come, per esempio, gli algoritmi con calcoli grandi
- Un processo I/O bound (usano di più l'I/O) come, per esempio, videogiochi, editor testo, etc...

In pratica vengono classificati attraverso i Bursts della CPU, cioè gruppi di esecuzione della CPU alternati a periodi di attesta per operazioni di I/O

La Politica di scheduling del processore (scheduler) decide quale processo viene eseguito ad un certo istante e diversi schedulers possono avere diversi obbiettivi:

- Massimizzare il throughput: cioè il numero di processi eseguiti per unità di tempo, tempo di attesa
- Minimizzare la latenza: situazione anomala
- Prevenire la starvation (attesa infinita)
- Completare i processi entro una scadenza temporale

- Massimizzare l'uso del processore

Tutti questi obbiettivi sono implementati da un algoritmo

I criteri di scheduling si suddividono in:

- Processi processor-bound: usa tutto il tempo di CPU disponibile
- Processi I/O-bound: genera richieste di I/O velocemente e lasci il processore
- Processi batch: richiedono lavoro da eseguire senza l'interazione dell'utente
- Processi interattivi: richiede frequenti input dall'utente

Gli obbiettivi dello scheduling sono:

1. Sistemi Batch
2. Sistemi Interattivi
3. Sistemi Real-time

Ci sono più livelli di scheduling, che possono essere suddivisi in tre categorie:

1. Scheduling di alto livello: determina quale job può competere per le risorse e controlla il numero di processi nel sistema ad un dato tempo e decido il livello ideale di multiprogrammazione
2. Scheduling di alto livello intermedio: determina quali processi possono competere per l'uso del processore e risponde a fluttuazioni del carico del sistema
3. Scheduling di basso livello: assegna le priorità e i processori ai processi, fisicamente

In pratica lo scheduling ad alto livello sceglie il livello di multiprogrammazione, quello intermedio lo tratta e quello basso lo assegna

I processi Interattivi sono soggetti a prelazione, in pratica possono essere rimossi o interrotti dall'attuale processore, questo può portare ad avere un miglioramento del tempo di risposta ed è importante per ambienti interattivi, però pur essendo soggetti a prelazione rimangono in memoria

Quelli invece che non sono soggetti a prelazione vengono eseguiti fino al completamento o fino a quanto utilizzano il processore, sono processi non importanti e possono bloccare indefinitamente altri più importanti

Priorità

Esistono due tipi di priorità:

1. Priorità statica (costante): la priorità assegnata non cambia, i pro sono che è facile da implementare, hanno un basso overhead e non è reattiva a variazioni dell'ambiente
2. Priorità dinamica (cambia nel temo): è reattiva a cambiamenti, reagisce ai cambiamenti di stato del sistema, favorisce una certa interattività e richiede maggior overhead della statica è giustificato dalla maggior capacità di reagire

Gli obbiettivi dello scheduling dipendono dal tipo di sistema e cercano di:

- Massimizzare il throughput molto rilevante in sistemi batch
- Massimizzare il numero dei processi interattivi che ricevono un tempo di risposta accettabile
- Minimizzare il tempo di risposta (turnaround) anche in sistemi batch
- Massimizzare l'uso delle risorse (utilizzazione)
- Evitare l'attesa infinita
- Forzare priorità: passaggio immediato da minima a massima
- Minimizzare l'overhead
- Garantire la predicibilità: non posso prevedere con la priorità dinamica

Per esempio nei sistemi:

- Batch (prestazioni): è un sistema che cerca di massimizzare il numero di processi serviti per unità di tempo (throughput), di minimizzare il tempo totale di residenza nel sistema (tempo di turnaround) e di massimizzare l'uso del processore (utilizzo della CPU)
- Interattivi: cercano di minimizzare il tempo di risposta e di adeguarsi alle richieste e aspettative degli utenti
- Real-time: l'obbiettivo è di rispettare le scadenze e di mantenere la qualità del servizio (prevedibilità)

Gli algoritmi di scheduling decidono quando e quanto a lungo porre in esecuzione ogni processo, dalla creazione di un processo figlio decidono chi deve eseguire, alla terminazione di un processo quale altro processo eseguire e se un processo si blocca quale altro processo eseguire (relazioni), fino alla gestione di interrupt

In pratica fa scelte su:

- Prelazione
- Priorità
- Tempo di esecuzione
- Tempo fino al completamento
- Equità

Esistono più tipo di algoritmi di scheduling, tipo:

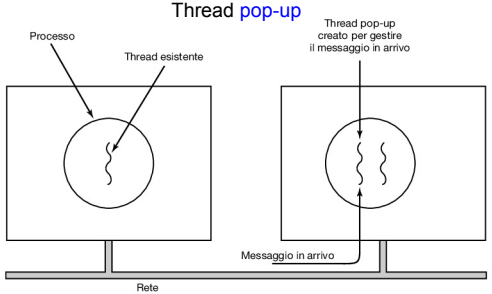
- Scheduling FIFO (First-In-First-Out): è lo schema più semplice che vedremo, in pratica i processi sono trattati in base al tempo di arrivo, senza la prelazione, viene utilizzato raramente come algoritmo principale di scheduling
- Scheduling SJF (Shortest-Job-First): lo scheduler selezione il processo con il minimo tempo per terminare stimato, questo porta ad una larga varianza (quando si discostano mediamente, se è grande avrà valori molto lontani fra di loro mediamente) dal tempo di attesa, anche qui la prelazione è assente e questo può portare tempi di risposta lenti a richieste interattive, si basa sulla stima del tempo per completare l'esecuzione che però potrebbe essere inaccurata o falsata, ma la si può correggere ma i tempi devono essere disponibili, questo tipo di sistema non può essere adatto per i moderni Sistemi Interattivi ma è più veloce del FIFO
- Scheduling SRT (Shortest-Remaining-time-First): è la versione con prelazione del SJF, i processi più corti in arrivo effettuano prelazione sui processi in esecuzione, la varianza del tempo d risposta è molto grande siccome i processi lunghi aspettano ancora di più che non con il SPF, teoricamente è un ottimo algoritmo di scheduling grazie al tempo medio di attesa ma non sempre ottimale in pratica, siccome i processi in arrivo corti possono effettuare prelazione sui processi quasi completati quindi c'è un overhead di cambio di contesto che può diventare significativo
- Scheduling RR (Round-Robin): è basato su quello FIFO, i processi son eseguiti solo per un periodo di tempo limitato detto intervallo o quanto di tempo, è presente la prelazione ed è facile da implementare, richiedere al sistema di mantenere parecchi processi in memoria per minimizzare l'overhead, spesso utilizzato come parte di algoritmi più

complessi e per sistemi interattivi, bisogna definire la dimensione del quanto, cioè, l'overhead di cambi di contesto con efficienza ridotta della CPU per limitare l'attesa in coda rispetto alla lunghezza media del Brust della CPU

La dimensione del **quanto** determina il tempo di risposta alle richieste interattive e quando è:

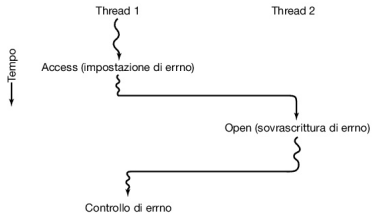
- Molto **grande** i processi vengono eseguiti per lungo tempo e rischia di degenerare nella FIFO
- Molto piccola il sistema passa più tempo nel cambio di contesto che nell'esecuzione di processi
- Media quindi è abbastanza lunga per i processi interattivi per fare le richieste di I/O e i processi batch ancora ottengono maggior parte del tempo del processore

Thread pop-up



Creazione di thread all'arrivo di un messaggio
prima dell'arrivo dopo l'arrivo

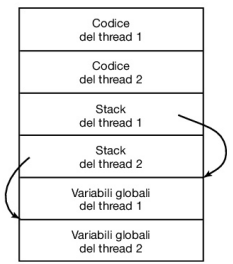
Thread - conflitti



Conflitti di thread per l'uso di una variabile globale '*errno*'

Possibili soluzioni: vietare variabili globali,
usare solo variabili globali private (copie private)
(maggior spazio di memoria)

Thread - conflitti



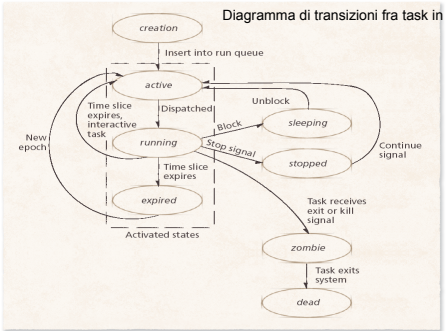
Uso di variabili globali private

Thread in Linux

- Linux alloca lo stesso tipo di descrittore di processo per processi e thread (tasks)
- `Task_struct` descrittore
- Linux usa la chiamata di sistema basata su UNIX `fork` per generare task figli
- Per abilitare il thread, Linux fornisce una versione modificata denominata `clone`
  - `clone` accetta argomenti che specificano quali risorse condividere con il task figlio

Thread in Linux

Diagramma di transizioni fra task in Linux



Threads Windows

- I processi contengono i programmi, gli `handle` (riferimenti ad oggetti) e i Thread con cui condividono risorse
  - `Thread`: unità corrente di esecuzione assegnata ad un processore
  - Esegue un pezzo di codice del processo nel contesto del processo, utilizzando le risorse del processo
  - Il contesto di esecuzione contiene
    - Runtime stack
    - Stato dei registri della macchina
    - Molti attributi
    - unità reale di esecuzione inviato ad un processore
- PEB Process Environment Block
- TEB Thread Environment Block

Threads Windows

- Windows può raggruppare i processi in *job*
  per limitare e gestire l'uso di risorse per tutti i thread del processo del job
  I *job* contengono processi
- threads possono creare e *fiber* allocando strutture dati e uno stack a livello utente
  - *Fiber* viene eseguito nel contesto del thread che lo crea, invece che lo scheduler
  - I thread possono essere convertiti in *fiber,* le *fiber* possono essere create indipendentemente dai thread
  - Vantaggio per cambio di contesto solo a livello utente
- La relazione *thread fiber* è molti a molti, ma di regola un thread è associabile a un insieme di *fiber*
  - Non sempre usate
- Windows fornisce ad ogni processo un pool di thread che si compongono di un numero di thread worker, che sono thread di livello kernel che eseguono funzioni previste dal thread utente

Threads Windows

- Thread pool funzionalità di Win32
  - Coda di task da eseguire
  - I thread del pool appena liberi prendono un task da eseguire dalla coda

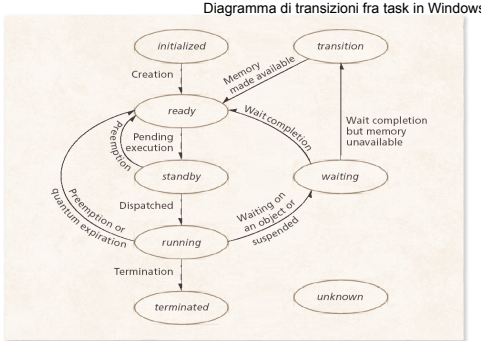
  Evita la creazione/distruzione continua di thread

  I thread possono bloccarsi in attesa di eventi e in quella fase non possono essere riassegnati ad una altro task da eseguire

  Esempio: in applicazioni di tipo cliente-servente

Threads Windows

Diagramma di transizioni fra task in Windows



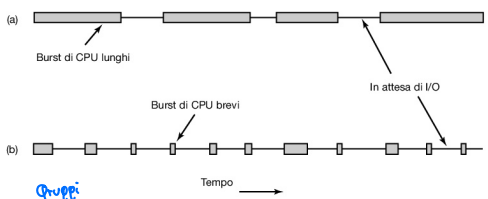
2 – Processi e Thread

Sommario

Obbiettivi

- Tipi di scheduling
- Scopi dello scheduling del processore
- Scheduling con e senza prelazione
- Uso della priorità nello scheduling
- Criteri di scheduling
- Algoritmi
- Scheduling con scadenza e real-time

Introduzione allo Scheduling



Gruppi

- Bursts of CPU alternati a periodi di attesa per operazioni di I/O
  (a) un processo CPU-bound → algoritmi con calcoli grandi, più CPU
  (b) un processo I/O bound → word processor, editor testo, più I/O

Introduzione allo Scheduling

- Politica di scheduling del processore } Scheduler
  - Decide quale processo viene eseguito ad un certo istante
  - Diversi schedulers possono avere diversi obbiettivi
    - Massimizzare il throughput → num. processi eseg. per unità di tempo
    - Minimizzare la latenza
    - Prevenire la starvation (attesa infinita)
    - Completare i processi entro una scadenza temporale
    - Massimizzare l'uso del processore

Tempo di attesa
Situazioni anormali
implementano il dei un algoritmo

Criteri di Scheduling

- Processi *processor-bound*
  - Usa tutto il tempo di CPU disponibile
- Processi *I/O-bound*
  - Genera richieste di I/O velocemente e lascia il processore
- Processi batch
  - Richiedono lavoro da eseguire senza l'interazione dell'utente
- Processi interattivi
  - Richiede frequenti input dell'utente

Tipi di Sistemi – obbiettivi dello Scheduling

- Sistemi Batch
- Sistemi Interattivi
- Sistemi Real-time

Livelli di Scheduling

Sceglie il livello multi

- Scheduling di alto livello
  - Determina quale job può competere per le risorse
  - Controlla il numero di processi nel sistema ad un dato tempo
  - Livello di multiprogrammazione → decido, livello ideale

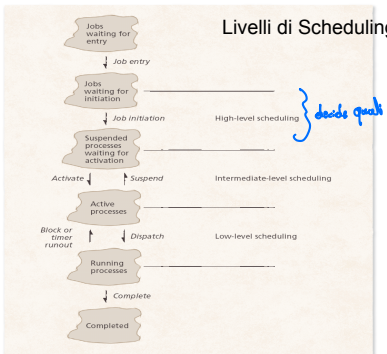
Tratta

- Scheduling di livello intermedio
  - Determina quali processi possono competere per l'uso del processore
  - Risponde a fluttuazioni del carico del sistema

Assegna

- Scheduling di basso livello
  - Assegna le priorità
  - Assegna i processori ai processi } fisicamente

Livelli di Scheduling



decide quali proces:

Scheduling con e senza prelazione

- Processi soggetti a prelazione
  - Possono essere rimossi dall'attuale processore
  - Si può avere un miglioramento del tempo di risposta
  - Importante per ambienti interattivi
  - I processi soggetti a prelazione rimangono in memoria
- Processi non soggetti a prelazione
  - Eseguiti fino al completamente o fino a quanto utilizzano il processore
  - Processi non importanti possono bloccarne indefinitamente altri più importanti

Priorità

- Priorità statica → costante
  - La priorità assegnata ad un processo non cambia
  - Facile da implementare
  - Basso overhead
  - Non reattiva a variazioni dell'ambiente
- Priorità dinamica → cambia nel Tempo
  - Reattiva a cambiamenti → reagisce ai cambiamenti di stato del sistema
  - Favorisce una certa interattività
  - Richiede maggior overhead della statica
    - Giustificato dalla maggior capacità di reagire

Obbiettivi dello Scheduling

- Diversi obbiettivi dipendono dal tipo sistema
  - Massimizzare il throughput molto rilevante in sistemi batch
  - Massimizzare il numero dei processi interattivi che ricevono un tempo di risposta accettabile
  - Minimizzare il tempo di risposta (turnaround) anche in sistemi batch
  - Massimizzare l'uso delle risorse (utilizzazione)
  - Evitare l'attesa infinita
  - Forzare priorità → passaggio immediato da minima a massima
  - Minimizzare l'overhead
  - Garantire la predicibilità → non posso garantire con la priorità dinamica

Obbiettivi dello Scheduling

- Diversi obbiettivi comuni a molti scheduler per sistemi generali
  - Equità *(Fairness)*
    ogni processo riceve la CPU in modo equo
  - Predicibilità
    la politica dichiarata deve essere attuata
  - Bilanciamento
    impegnare tutte le parti del sistema

92

Obbiettivi dello Scheduling

Diversi obbiettivi per

- Sistemi Batch → prestazioni
  - Throughput
    massimizzare il numero di processi serviti per unità di tempo
  - Tempo di turnaround
    minimizzare il tempo totale di residenza nel sistema
  - Utilizzo della CPU
    massimizzare l'uso del processore
- Sistemi Interattivi
  - Tempo di risposta minimizzare
  - Adeguatezza alle richieste e aspettative degli utenti
- Sistemi Real-time
  - Scadenze rispettarle
  - Prevedibilità mantenere la qualità del servizio

93

Algoritmi di Scheduling

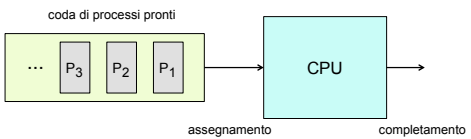
Decidono quando e quanto a lungo porre in esecuzione ogni processo
- alla creazione di un processo figlio chi eseguire
- alla terminazione di un processo quale altro processo eseguire
- se un processo si blocca quale altro processo eseguire (relazioni)
- alla gestione di interrupt

- Fa scelte su
  - Prelazione
  - Priorità
  - Tempo di esecuzione
  - Tempo fino al completamento
  - Equità

94

Scheduling First-In-First-Out (FIFO)

- Scheduling FIFO
  - Lo schema più semplice
  - I processi sono trattati in base al tempo di arrivo
  - Senza prelazione
  - Utilizzato raramente come algoritmo principale di scheduling



95

Shortest-Job-First (SJF) Scheduling

- Scheduler seleziona il processo con il minimo tempo per terminare stimato
  - Tempo media di attesa minore di FIFO
    - Riduce il numero di processi in attesa
  - Potenzialmente larga varianza del tempo di attesa → quando si discostano molto, si è grande valore molto lontani fra di loro mediamente
  - Senza prelazione
    - Può portare a tempi di risposta lenti a richieste interattive
  - Si basa sulla **stima** del tempo per completare l'esecuzione
    - Potrebbe essere inaccurata o falsata
    - Correzioni possibili
    - Tutti i tempi devono essere disponibili
  - Non sempre adatto per moderni sistemi interattivi

Shortest-Remaining-Time- First (SRT) Scheduling

- SRT scheduling
  - Versione con **prelazione** di SJF
  - I processi più corti in arrivo effettuano prelazione sui processi in esecuzione
  - Varianza del tempo di risposta molto grande: i processi lunghi aspettano ancora di più che non con SPF
  - Teoricamente ottimo per il tempo media di attesa
  - Non sempre ottimale in pratica
    - I processi in arrivo corti possono effettuare prelazione su processi quasi completati
    - Overhead di cambio di contesto che può diventare significativo

Scheduling in sistemi batch

Esecuzione di quattro lavori nell'ordine di arrivo

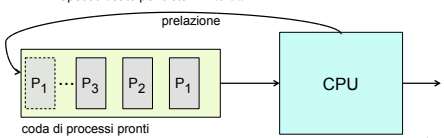
Esecuzione di quattro lavori In ordine di tempo di esecuzione crescente

Un esempio di scheduling shortest job first SJF

A. Tanenbaum – Modern Operating Systems

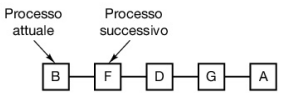
Scheduling Round-Robin (RR)

- Scheduling Round-Robin
  - Basato su FIFO
  - I processi son eseguiti solo per un periodo di tempo limitato detto intervallo o **quanto** di tempo
  - Con prelazione
  - Facile da implementare
  - Richiede al sistema di mantenere parecchi processi in memoria per minimizzare l'overhead
  - Spesso utilizzato come parte di algoritmi più complessi
  - Spesso usato per sistemi interattivi

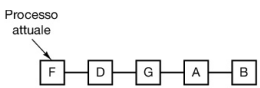


Scheduling Round-Robin

Scheduling Round Robin in sistemi interattivi



lista di processi eseguibili



lista di processi eseguibili dopo che il processo B ha usato il suo quanto

A. Tanenbaum – Modern Operating Systems