SAPIENZA
UNIVERSITÀ DI ROMA

Scheduler dei processi di un sistema operativo, confronto ed analisi di varie politiche

Facoltà di Ingegneria dell'informazione, informatica e statistica

Corso di Laurea in Ingegneria informatica ed automatica

Candidato

Simone Trenta
Matricola 1724141

Relatore

Prof. Giorgio Grisetti

Anno Accademico 2020/2021

Tesi discussa il 22 September 2015
di fronte a una commissione esaminatrice composta da:

Prof. ... (presidente)
Prof. ...
Prof. ...
Prof. ...
Prof. ...
Prof. ...
Prof. ...

Scheduler dei processi di un sistema operativo, confronto ed analisi di varie politiche
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: trenta.1724141@studenti.uniroma1.it

Sommario

Tramite un progetto di simulazione di uno scheduler di un sistema operativo, si analizzano le varie politiche di scheduling fra le più comuni, ossia first came first served (FCFS), round robin (RR), shortest job first (SJF) e shortest remaining job first (SRJF).

Indice

Capitolo 1

Introduzione

In un sistema operativo la gestione delle richieste di risorse da parte dei processi è demandata allo scheduler. Questa funzione di un sistema operativo è pressoché essenziale, senza la quale le risorse disponibili non potrebbero essere riservate e utilizzate in modo efficiente.

Nel Capitolo 2

Nel Capitolo 3

Nel Capitolo 4

Capitolo 2

Lo scheduler in un sistema operativo

2.1 La sua utilità

Un odierno sistema operativo è composto da molteplici parti, una di esse è lo scheduler dei processi. Questi ultimi sono parte costituente del sistema operativo, sono la parte esecutiva di esso. Per meglio capire, i processi richiedono delle risorse del calcolatore, che siano risorse unicamente della CPU oppure di risorse di I/O. Qualunque tipo di risorsa richiedano, spesso non è immediatamente disponibile, essendo limitate in un calcolatore. Per cui è necessaria la presenza di un elemento che permetta al dispositivo di gestire le proprie risorse, tale elemento è per l'appunto lo scheduler. Vedendo il problema da parte dei processi, problema che è la possibilità di usare una risorsa o meno, si crea la necessità di dare delle priorità per poterne usufruire. Andando quindi a dare origine a tali priorità si producono di conseguenza delle politiche per poterle mettere in atto.

2.2 Politiche di scheduling

Le politiche di scheduling sono degli algoritmi, che permettono quindi di riservare le risorse del calcolatore in un certo modo. Ognuna di esse ha, come ogni cosa, dei vantaggi e degli svantaggi. Tale prospettiva non permette di rendere efficiente una unica politica per ogni tipo di calcolatore, ma sarà necessario dotare ogni sistema operativo con l'algoritmo più adatto. Alcuni parametri che vengono persi in considerazione per valutare l'efficienza di un algoritmo sono:

- Utilizzo della CPU
- Throughput del sistema
- Tempo di attesa di un processo
- Turnaround di un processo
- tempo di risposta di un processo

Fra questi non tutti potranno essere ottimizzati allo stesso tempo, in particolare nei sistemi batch si massimizza throughput e minimizza il turnaround, mentre nei sistemi interattivi si minimizza il tempo medio di risposta ed il tempo di attesa. I parametri considerati nel progetto implementato sono il tempo di attesa e il tempo

di completamento di un processo. Quest'ultimo da considerarsi come il tempo totale che ogni processo ha impiegato per terminare, per cui partendo sempre dallo stesso valore per il tempo massimo iniziale, se il tempo di completamento sarà elevato vorrà dire che il processo ha impiegato molto per concludersi, rimanendo talvolta in attesa.

2.2.1 First came first served

L'algoritmo *first came first served*, forse, è il più semplice da capire, e da implementare. Come dice il nome stesso esso non ha una particolare complessità, semplicemente quando un processo arriva, se è disponibile la risorsa, quindi un core della cpu, esso viene messo in running. Altrimenti viene messo in una coda di waiting, dalla quale vengono presi i processi da mettere in running, prima di considerare i nuovi processi in arrivo. In questa politica può risultare che un processo debba attendere la terminazione di un altro prima di poter entrare in running. Per cui può risultare che dovrà attendere molto in stato di waiting, nascendo quindi il problema di attesa lunghe se le durate dei processi sono elevate.

2.2.2 Round robin

L'algoritmo *round robin* è una successiva perfezione del precedente fcfs (first came first served). Anche in questo caso è presente una coda dei processi che sono arrivati. Se è disponibile la risorsa richiesta dal processo in arrivo, gli viene riservata e tale processo può sfruttarla, altrimenti viene inserito nella coda di attesa. Un processo in running, però, non ha la risorsa riservata fino a quando lo necessita, ma in questo caso è presente un clock, che inizia a scorre appena il processo entra in running. Terminato il clock, la risorsa viene liberata dal processo. Esso viene inserito nella coda dei processi in attesa, e da essa viene preso il primo processo per destinargli la risorsa appena liberatesi. Il problema di questo algoritmo è scegliere un valore ottimale per il clock, se troppo elevato si tenderà ad avere un approccio simile a fcfs, se non uguale, se troppo breve i processi non riusciranno a sfruttare la risorsa richiesta che dovranno liberarla.

2.2.3 Shortest job first

L'algoritmo *shortest job first* ordina i processi da mettere in running, ovviamente se c'è disponibilità di risorsa, in base a quanto tempo essi la richiedono, dal più breve al più lungo. Anche in questo algoritmo un processo entrato in running termina completamente il suo job prima di terminare. Se la risorsa non è disponibile, il processo viene inserito in una coda, il cui ordinamento è basato sulla durata del job. Inoltre tutti i processi da mettere in running vengono presi da questa coda a cui si aggiungo i processi che arrivano in un determinato istante. In questo algoritmo risulterà che i processi con la richiesta minore di risorsa saranno schedulati e termineranno per primi. Il tempo medio di attesa si riduce, però il problema principale di questa politica nella realtà è riuscire a predire, con sufficiente accuratezza, quanto un processo occuperà una risorsa. Nel caso del simulatore sviluppato ciò non è necessario, poiché ogni processo che arriva dichiara per quanto tempo occuperà la risorsa. Al contrario, realisticamente, si utilizza la seguente formula per predire l'utilizzo successivo della cpu:

$$\hat{b}_{t+1} = \alpha b_t + (1 - \alpha)\hat{b}_t \tag{2.1}$$

In cui si ha:

$$\hat{b}_{t+1} \tag{2.2}$$

è la predizione al tempo t+1;

$$\alpha \tag{2.3}$$

è il coefficiente di decadimento;

$$b_t \tag{2.4}$$

è la misura avvenuta al tempo t;

$$\hat{b}_t \tag{2.5}$$

è la predizione al tempo t.

2.2.4 Shortest remaining job first

L'algoritmo *shortest remaining job first* è simile al precedente sjf. Però in questo caso ad ogni ciclo di clock della cpu, i processi in running vengono confrontati con tutti quelli in waiting e quelli appena arrivati. I processi, sempre in base alla disponibilità della risorsa, vengono messi in running, o ci rimangono, in ordine in base alla durata del job, dal più piccolo al più grande. Anche in questo caso il problema principale è riuscire a predire la durata della richiesta della risorsa, e si utilizza la formula (2.1).

Capitolo 3

MOONS

3.1 The Very Large Telescope

Property of the European Southern Observatory...

3.2 The Multi-Object Optical and Near-infrared Spectrograph

The *Multi-Object Optical and Near-infrared Spectrograph* is a future generation MOS instrument for the VLT.

Capitolo 4

Conclusions

The grasping power of the mirror..

Bibliografia

[1] Blanche P.A., Gailly P., et al., "*Volume phase holographic gratings: large size and high diffraction efficiency*", Optical Engineering, Vol. 43, No.11, November 2004

[2] Cirasuolo M., et al., *"MOONS Science Report"*, MOONS Document Number: VLT-TRE-MON-14620-0001, Issue: 1.0, 31st January 2013

[3] European Southern Observatory, http://www.eso.org