Progetto scheduler simulator (link pagina GitHub)

Cassissa Federico
Giliberti Jeff Michael

Esame di
Calcolatori Elettronici
Consegna del 14/09/18

Descrizione del simulatore

Il progetto consente di confrontare le performance della CPU in base al diritto di prelazione e l'algoritmo di scheduling scelto. Il programma simula il parallelismo di due processori e la concorrenza tra processi.

I processori sono implementati con due thread sincroni a corrispondenza 1:1 con i thread a livello Kernel con oppure senza preemption.

La memorizzazione dei tasks avviene tramite liste doppiamente concatenate: ogni coda di stato è identificata da una lista ed ogni task è assegnato esclusivamente ad una sola lista che infatti rappresenta il suo stato. Gli inserimenti nelle code di READY e BLOCKED sono gestite nel seguente modo:

- READY: i tasks sono ordinati in ordine decrescente di priorità associata che è indicata dal "service_time", l'ordinamento si mantiene inserendo i tasks nel posto corretto. Entrambi i core prelevano dalla stessa lista READY scorrendo dalla testa, controllando preventivamente che l'"arrival_time" sia compatibile con il proprio clock;
- BLOCKED: la lista contiene in testa i tasks bloccati dal core0 in ordine crescente di attesa; in coda contiene i tasks bloccati dal core1 in ordine decrescente. Il tempo di attesa è memorizzato come "cicli di clock attuali + durata operazione bloccante". In questo modo ogni core verificherà i tasks bloccati partendo dall'inizio oppure dal fondo.

Sono inoltre presenti le liste NEW, in cui vengono inseriti i tasks letti dal documento in input, RUNNING, in cui sono posti i processi in esecuzione, e EXIT, in cui sono posti i tasks che hanno terminato la loro esecuzione. La lista RUNNING è organizzata in base ai core similmente alla lista BLOCKED, mentre le altre due sono ordinate semplicemente per arrivo dei tasks, indipendentemente dal core.

Dopo aver elaborato, in "main.c", i comandi ricevuti in avvio, avviene la creazione e gestione dei processi figli. In essi sono creati e gestiti i thread. Infine, all'interno della funzione eseguita dai thread avviene la vera e propria simulazione dei core attraverso lo scheduling e l'esecuzione stessa dei tasks. Il passaggio di stato dei tasks da una coda all'altra è implementato e gestito in "listmaker.c".

Comparativa algoritmi di scheduling

Scheduler not preemptive

La schedulazione dei tasks avviene secondo la politica “Shortest Process Next” (SPN) che permette di minimizzare il tempo di attesa medio trascorso nella coda di ready, questo perché evita l’attesa di lunghe operazioni eseguite dai tasks più lunghi, da parte di quelli più brevi. L’algoritmo favorisce la massimizzazione del throughtput, infatti tra i processi in ready è scelto quello con minore “service time” totale, indicato dal numero di cicli di clock che impiegherà la CPU per portarlo a termine.
Comporta un aumento del tempo di risposta medio ossia il numero di cicli di clock che intercorrono da quando il processo è inserito nella coda di ready a quando sarà portato nello stato di running assegnandogli la CPU. In alcuni casi questo determina una lunga attesa per i processi più lunghi che dovranno attendere l‘elaborazione dei processi più brevi.

Scheduler preemptive

La schedulazione dei tasks avviene secondo la politica “Shortest Remaining Time” (SRT) che permette di massimizzare il throughput perché la CPU viene assegnata al task che ha il minor numero di istruzioni ancora da eseguire.
Anche in questo algoritmo si favorisce la minimizzazione del tempo di attesa medio a sfavore del tempo di risposta perché i processi più lunghi potrebbero dover attendere molto prima di essere schedulati per la prima volta.

Esempio di esecuzione di alcuni tasks del test case 03_tasks.csv

ID task	lunghezza	N° istruzioni	Arrival_time	Exit_time	
				preemp	not_preemp
0	562	316	2	5025	78757
1	252	141	16	414	586
2	664	399	41	117293	138225
3	799	497	51	188302	173608
4	349	234	81	552	2336
9	1324	778	180	520423	514424
50	1097	714	856	359008	348688
100	539	310	1720	80252	69259
1000	1625	950	16409	736316	802303
1821	70	48	29886	30071	30107

Possiamo notare che i processi più brevi sono completati in minor tempo dallo scheduler preemptive come accade per il processo con id 4; al contrario i processi più lunghi come il 3, il 9 ed il 50 sono terminati prima dallo scheduler not preemptive. Ciò non si verifica per il processo con id 1000, questo può essere dovuto al fatto che il processo preemptive aggiorna la priorità del processo con il numero effettivo di istruzioni rimanenti.

In conclusione, si può affermare che lo scheduling con preemption consente di diminuire ulteriormente il tempo d'attesa medio al costo di aver un maggior numero di cambi di contesto.

Giustificazione delle scelte implementative

- Scelta delle strutture dati: le liste doppiamente linkate hanno rappresentato un buon compromesso tra difficoltà implementativa, efficienza ed elasticità. In particolare l'elasticità, avendo cinque diverse code da gestire. Riguardo alla complessità computazionale, nonostante l'ottimizzazione negli inserimenti non la migliori, consente tuttavia un deciso miglioramento nel tempo d'esecuzione.

- Scelta delle politiche di scheduling: la scelta è ricaduta su algoritmi di scheduling con priorità. Nel nostro caso era possibile conoscere in anticipo il numero di istruzioni di ogni task: questa specificità di conseguenza ci ha fatto scegliere gli algoritmi SPN e SRT; nella reale elaborazione di una CPU non è possibile conoscere esattamente le durate dei singoli task.

Descrizione delle problematiche incontrate

- Collaborazione tra i membri del team: per condividere e lavorare in modo semplice abbiamo deciso di sviluppare il progetto su GitHub e lavorare in "live sharing" nell'ambiente di Visual Studio Code tramite un'estensione apposita; infine l'esecuzione del programma su ambiente Linux è avvenuta tramite la WSL (Windows Subsystem for Linux).
- Ideazione e scrittura degli algoritmi: prima di scrivere il codice si è reso necessario ragionare, scrivere su carta e testare i passaggi che dovevano essere eseguiti dall'algoritmo ed individuare tutti i casi possibili che si potevano presentare.
- Debugging e testing: infine i malfunzionamenti sono stati individuati con l'utilizzo del debugger, valgrind e l'analisi dei dati di output.