SCHEDULING DEI THREAD > 5.4

Se supportati, l'SO fa lo scheduling dei soli Kernel-level thread. Per eseguire gli User-level bisogna associarli, solitamente con LWP.

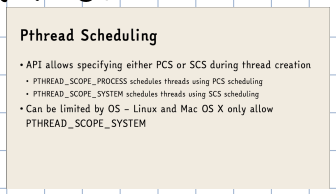
AMBITO DELLA CONTESA 5.4.1

Si intende la contesa per aggiudicarsi la CPU:

- **Process Contention Scope (PCS)**
  Presente nei modelli many-to-many e many-to-one
  L'esecuzione è pianificata su un LWP libero, per cui la contesa si verifica tra thread dello stesso processo.
- **System Contention Scope (SCS)**
  Presente nel modello one-to-one (Linux/windows).
  La contesa avviene su tutti i Kernel-level thread.

SCHEDULING CON PTHREADS 5.4.2

Si può specificare nella creazione



SCHEDULING PER SISTEMI MULTI PROCESSORE > 5.5

Si possono distinguere in:

- omogenei: ogni processore ha le stesse specifiche
- non omogenei: hanno specifiche energetiche, frequenza diverse.

Le strategie dello scheduling possono essere:

AMP: ASYMMETRIC MULTI PROCESSING

Le attività del S.O. sono affidate ad un unico processore!

- vantaggi: - facile da realizzare - poca IPC
- svantaggi: - il "master process" diventa collo di bottiglia.

SMP: SYMMETRIC MULTI PROCESSING

Ogni processore è in grado di auto-gestirsi e di selezionare il task da eseguire dalla ready queue.
Da cui si può avere che la ready queue è

- comune: Bisogna impedire che processori distinti

scelgono lo stesso task.

- privata: richiede un corretto bilanciamento dei task.

LOAD BALANCING > S.S.3

Tecnica usata negli SMP con coda privata. Prevede di bilanciare il carico tra i processori.

Possono essere presenti in contemporanea:

- **Push migration**
  Vengono fatti dei controlli periodici sul bilanciamento da un processore. Se rileva sbilanciamenti sposta dei task verso le code dei processori scarichi.
- **Pull migration**
  Ogni processore scarico prende dei task in attesa dai processori carichi.

PRELAZIONE DEL PROCESSORE > S.S.4

È una tecnica usata spesso nei sistemi NUMA in cui si associa ad un processo il suo processore.

Può essere:

- **soft affinity**: L'S.O. tenta ma non garantisce l'affinity.
- **hard affinity**: l'S.O. garantisce l'affinity.

NB: è naturale il conflitto col load balancing.

PROCESSORI MULTI-CORE > S.S.2

Nei sistemi moderni, ogni processore ha più unità di calcolo (core) al suo interno. Per l'SO appaiono come proc separati. In questi sistemi si possono definire 2+ thread hardware per core, in modo di commutare le esecuzioni

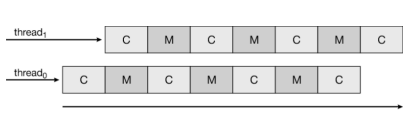
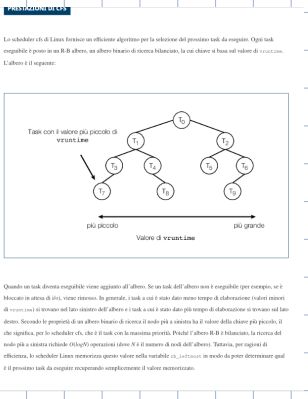


Figura 5.13 Sistema multicore e multithread.

SCHEDULING SU LINUX > 5.7.1

- 0_2.5: Linux era pensato per 1 core solo
- 2.6: viene creato il CFS Completely Fair Scheduler ed è usato con la classe di scheduling real time



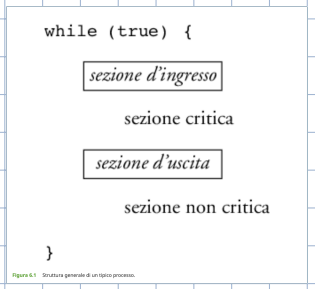
Lo scheduler cfs di Linux fornisce un efficiente algoritmo per la selezione del prossimo task da eseguire. Ogni task eseguibile è posto in un R-B albero, un albero binario di ricerca bilanciato, la cui chiave si basa sul valore di `vruntime`. L'albero è il seguente:

Quando un task diventa eseguibile viene aggiunto all'albero. Se un task dell'albero non è eseguibile (per esempio, se è bloccato in attesa di i/o), viene rimosso. In generale, i task a cui è stato dato meno tempo di elaborazione (valori minori di `vruntime`) si trovano nel lato sinistro dell'albero e i task a cui è stato dato più tempo di elaborazione si trovano nel lato destro. Secondo le proprietà di un albero binario di ricerca il nodo più a sinistra ha il valore della chiave più piccolo, il che significa, per lo scheduler cfs, che è il task con la massima priorità. Poiché l'albero R-B è bilanciato, la ricerca del nodo più a sinistra richiede $O(\log N)$ operazioni (dove N è il numero di nodi dell'albero). Tuttavia, per ragioni di efficienza, lo scheduler Linux memorizza questo valore nella variabile `rb_leftmost` in modo da poter determinare quali è il prossimo task da eseguire recuperando semplicemente il valore memorizzato.

CRITICAL SECTION PROBLEM > 6.2

È un problema in cui, dati N processi, ognuno di essi ha una sezione critica in cui vengono modificati dati comuni. Si definiscono 3 sezioni:

- sezione d'ingresso
  In cui ogni processo richiede il permesso per entrare ai processi dentro.
- sezione critica
- sezione d'uscita: In cui il processo esce.



```
while (true) {
    sezione d'ingresso
        sezione critica
    sezione d'uscita
        sezione non critica
}
```

Figura 6.1 Struttura generale di un tipico processo.

Soluzioni al problema devono presentare le seguenti caratteristiche:

- mutua esclusione:
  Se un processo è in esecuzione nella sezione critica, nessun altro può entrare in esecuzione.
- progresso
  Se la sezione critica non è occupata e un processo vuole entrare può farlo.
- attesa limitata
  Ogni processo non può stare in attesa per sempre.