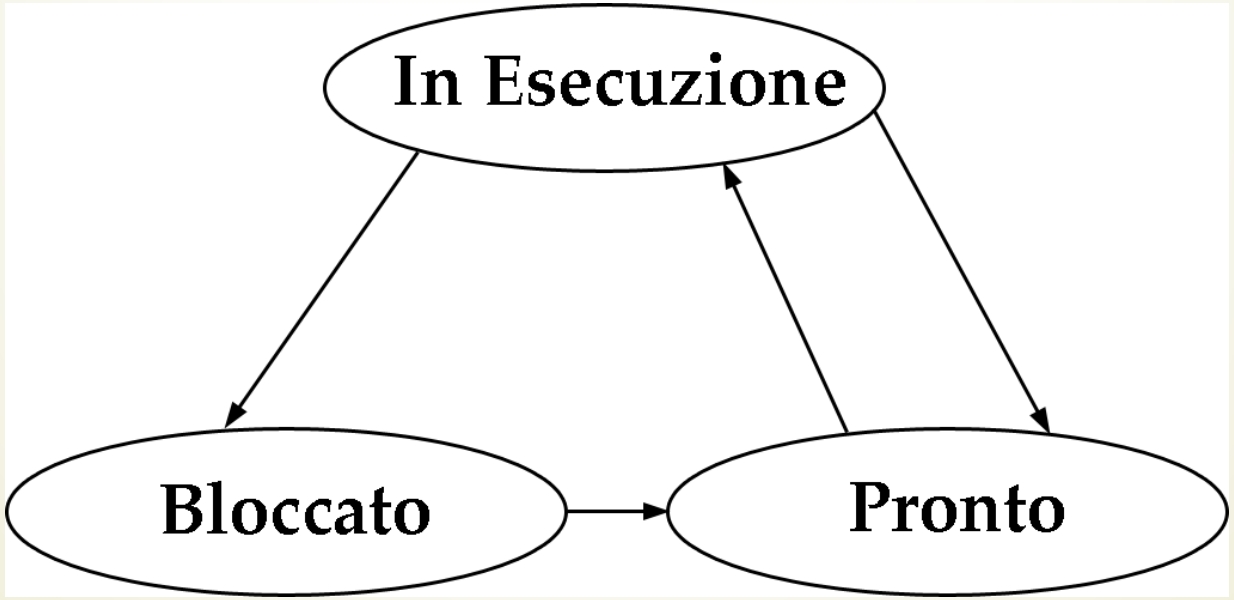
ESERCITAZIONE 6 - Soluzioni

Processi e thread

Processi e Thread

Calcolatori e processi (1)

1) In figura è mostrato lo stato di tre processi. In teoria, con 3 stati potrebbero verificarsi 6 transizioni, 2 per uscire da ogni stato. Tuttavia, sono mostrate solo 4 transizioni. Esistono circostanze in cui possono verificarsi una o entrambe le transizioni mancanti?



Calcolatori e processi (1)
Soluzioni (1)

- In un calcolatore basilare, che può risolvere un unico task alla volta, potrebbe avvenire la transizione *Blocked→Running*, in quanto essendo il processo attualmente in considerazione l'unico risolvibile dal core, non ci sarebbe la necessità di porlo preventivamente nello stato *Ready*.
- La transizione *Ready → Blocked*, invece, non dovrebbe mai verificarsi. Infatti, anche nei casi limite, un processo in stato *Ready* è, per l'appunto, pronto, e non ci sono situazioni naturali in cui si possa invocare una chiamata bloccante su un altro processo.

Calcolatori e processi (2)

2) Confrontare la lettura di un file usando un file server a thread singolo e uno multi-thread. Occorrono 15 ms per ottenere una richiesta di lavoro, smistarla ed elaborala, nel caso in cui essa faccia riferimento a dati presenti in un blocco di cache. Nel caso in cui sia necessario effettuare un'operazione su disco (che avviene in un terzo dei casi) sono richiesti ulteriori 75 ms, durante i quali il processo è sospeso. Quante richieste al secondo può gestire il server se è a thread singolo? E se è multi-thread?

Calcolatori e processi (2) Soluzioni (1)

- Thread singolo

15 ms → 15 ms → 90 ms

Ripetere fino al raggiungimento di 1s.

120 ms: 3 richieste

1000/120 = 8,3

8,3 * 3 = 25 richieste al secondo

Calcolatori e processi (2)
Soluzioni (2)

- Thread multipli

15 ms → 15 ms → 90 ms

In ogni richiesta di 90 ms possono essere risolte 5 richieste da 15 ms. Si mantenga, tuttavia, la proporizone di 1 a 2 per le richieste a disco.

L'idea è: 90 ms

15 ms

15 ms

Quindi, ogni 90 ms si risolvono 3 richieste. Il calcolo ora è semplice:

1000/90 = 11

11*3 = 33 richieste al secondo.

Calcolatori e processi (3)

3) In un computer con 1 GB di memoria, il sistema operativo occupa 512 MB e i processi occupano mediamente 64 MB, se l'attesa media dell'I/O è del 60%, qual è l'utilizzo della CPU? Aggiungendo 256 MB di RAM, quale sarà il nuovo utilizzo della CPU?

Calcolatori e processi (3) Soluzioni (1)

- Il tempo di attesa del 60% è riferito all'attesa di completamento di operazioni di I/O.
- Se abbiamo *n* processi in memoria e il tempo di attesa medio è *p*, allora la probabilità che tutti i processi siano in attesa (e quindi la CPU sia intattiva) è p^n
- L'utilizzo della CPU è, dunque, 1- p^n
- Il numero *n* di processi è ottenibile dividendo la dimensione di RAM libera per lo spazio occupato naturalmente da un solo processo.

Calcolatori e processi (3) Soluzioni (2)

- $p = 60\%$
- n = (1GB – 512 MB) / 64 MB = 512/64 = 8
- Utilizzo CPU = $1 - p^n = 1 - 0.6^8 = 0.983$

Calcolatori e processi (3) Soluzioni (3)

- p = 60%
- Nuova RAM: 256MB
- n = (1GB – 512 MB+256) / 64 MB = 768/64 = 12
- Utilizzo CPU = 1- p^n = 1 - 0.6^{12} = 0.997

Calcolatori e processi (4)

4) In un sistema a thread a livello utente, c'è uno stack per thread oppure ce ne è una per processo? E in un sistema a thread a livello kernel? Spiegare.

Calcolatori e processi (4) Soluzioni (1)

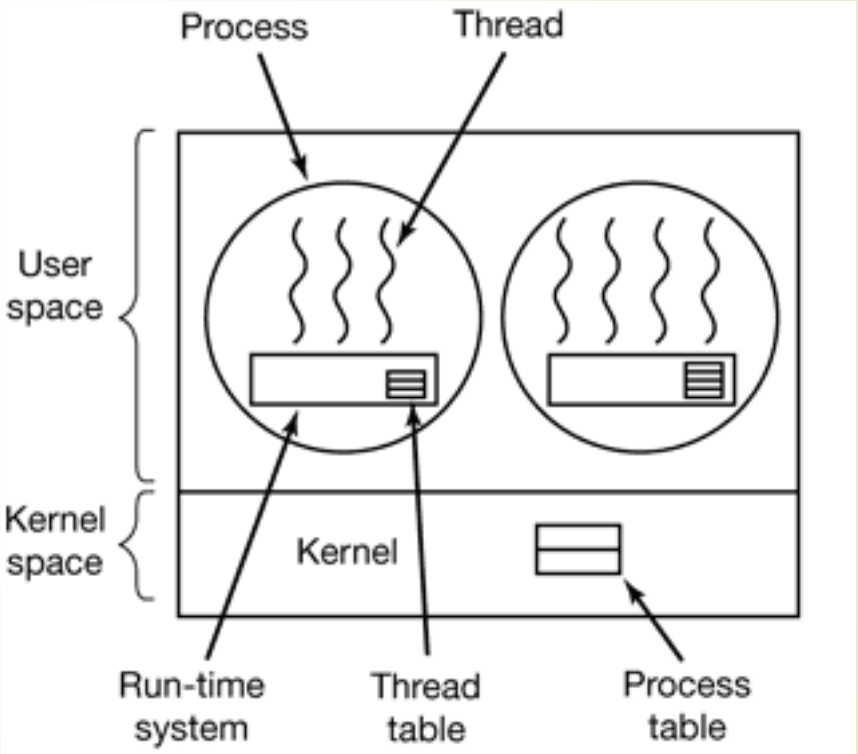
Thread

- unità di base di utilizzo della CPU
- Contiene:
  - Program counter
  - Insieme registri
  - Spazio stack
- Condivide:
  - Spazio indirizzamento (no protezione)
  - Dati globali
  - File aperti

Calcolatori e processi (4) Soluzioni (2)

Thread livello utente

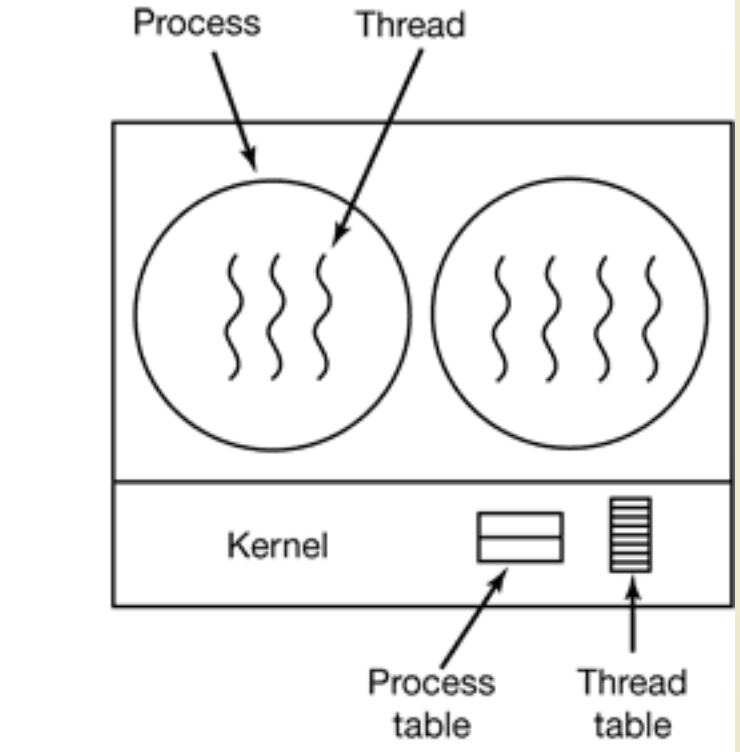
- Sistema ha un solo processo
- Implementabili in sistemi che non supportano thread
- Ogni processo ha tabella thread
- Cambio contesto thread più veloce rispetto processi
- Thread hanno algoritmo schedulazione interno
- Problema: chiamate bloccanti
- Thread possono switchare solo se rilascio CPU volontario



Calcolatori e processi (4) Soluzioni (3)

Thread livello kernel

- Sistema conosce e gestisce i thread
- Tabella thread oltre che processi
- Chiamate bloccanti sono chiamate di sistema
- Costo elevato creazione superato tramite "riciclo" thread terminati.



Calcolatori e processi (4)
Soluzioni (4)

- Soluzione:
  - Thread livello kernel: una stack per thread. Ognuno di essi è riconosciuto dal sistema operativo. Conseguentemente, più di una stack per processo.
  - Thread livello utente: una stack per thread e una sola per processo. A livello di kernel, infatti, i thread sono completamente trasparenti.

Calcolatori e processi (5)

4) Supponendo di utilizzare l'algoritmo round – robin per lo scheduling in un sistema interattivo, se la coda è fatta dai processi A-B-C-D-E, nel caso in cui il quanto sia di 100 ms e il un tempo di cambio di contesto di 1 ms, quanto tempo sarà necessario prima che E venga eseguito? Qual è il rapporto tra cambio di contesto e tempo di esecuzione?

Come cambiano I tempi analizzati nel caso in cui il quanto sia, invece, di 4 ms?

Quale delle due soluzioni sembra più favorevole?

Calcolatori e processi (5) Soluzioni (1)

- Round- robin: a ogni processo viene assegnato un quanto di tempo in cui può andare in esecuzione, i processi vengono assegnati alla CPU tramite una coda circolare.
- Se ha una richiesta bloccante o è terminato prima dello scadere del quanto, la CPU viene immediatamente assegnata al processo successivo.
- Al termine di un quanto di tempo, viene cambiato l'assegnamento alla CPU, e deve intercorrere un certo tempo, il tempo di **cambio di contesto** prima che il nuovo processo possa iniziare la propria esecuzione.

Calcolatori e processi (5) Soluzioni (2)

- E: devono essere eseguiti per il loro quanto A, B, C e D e i ripettivi cambi di contesto. Quindi E deve aspettare, per la sua prima parte di esecuzione:

100 + 1 + 100 +1 + 100 + 1 + 100 + 1 =404 ms, cioè quasi mezzo secondo

- Il rapporto è molto semplice: 1ms/100ms = 0.01, cioè l'1%
- Secondo caso:

4 + 1 + 4 + 1 + 4 + 1 + 4 + 1 =20 ms

- Il rapporto è: 1ms/4ms = 0.25, cioè il 25%

Calcolatori e processi (5) Soluzioni (3)

- Confronto: pro prima soluzione:
  - Tempo cambio di contensto molto più piccolo di tempo esecuzione,
  - Processi con lunghe CPU burst favoriti
- Confronto: contro prima soluzione
  - Tempo attesa processi potenzialmente lungo (400 ms è un tempo considerevole)
  - Se frequenti I/O burst, simile a seconda soluzione
- Confronto: pro seconda soluzione:
  - Processi hanno tempo di attesa basso, importante specialmente se forniti di iterfaccia utente
- Confronto: contro seconda soluzione:
  - Cambio contesto è rilevante: confrontabile con tempo esecuzione
  - Sfavorisce processi con lunghe CPU burst