5 - Processi e Threads

sabato 26 marzo 2022 11:54

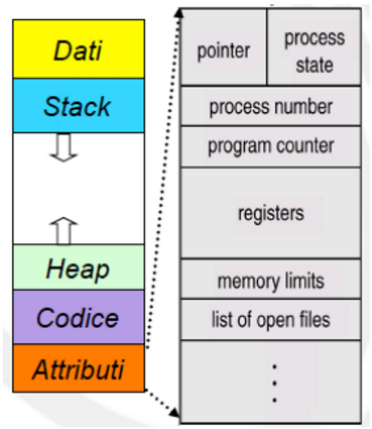
PROCESSO

- Processo: programma in esecuzione
  - -programma: concetto statico (il programma in sè)
  - -processo: concetto dinamico
- Il SO consiste di tanti processi
- Il processo viene eseguito in 2 modi:
  - -**sequenziale:** un'istruzione alla volta
  - -**concorrente:** ogni processo concorre per avere le risorse limitate della macchina (la CPU)
- Se ci sono più processi, le istruzioni di ogni processo non vengono eseguite nello stesso momento
- Ogni processo è descritto dal suo PCB

PROCESS CONTROL BLOCK - PCB

- PCB: una struttura dati con una tabella in memoria principale contenente le seguenti informazioni:

-stato
-Program Counter (PC)
-utilizzo dei registri
-CPU scheduling
-utilizzo della memoria
-accounting
-utilizzo dell'I/O



STATO DI UN PROCESSO

- Durante la sua esecuzione, un processo evolve in diversi stati:
  - -**New:** processo appena avviato, deve essere caricato in memoria
  - -**Ready:** processo in memoria, aspetta di utilizzare la CPU
  - -**Running:** usa la CPU
  - -**Waiting:** il processo torna in memoria se deve aspettare un evento o deve riutilizzare la CPU
  - -**Terminated:** il processo ha finito, muore

SCHEDULING

- Scheduling: gestione dei processi da eseguire nella CPU
  - -se ne occupano gli scheduler

- Long-term scheduler (job scheduler): seleziona quali processi spostare in memoria (decide chi ammettere)
  - -i processi diventano ready in memoria
  - -viene invocato con intervalli di alcuni secondi
- Short-term scheduler (CPU scheduler): seleziona quali processi devono essere eseguiti dalla CPU
  - -i processi passano da ready a running (dispatch)
  - -viene invocato molto frequentemente (parliamo di millisecondi)

- I processi vengono inseriti in delle code:
  - -coda dei processi pronti (ready queue): coda dei processi pronti per l'esecuzione
  - -coda di un dispositivo: cosa dei processi che aspettano che un dispositivo si liberi (ogni dispositivo ha la sua coda)

OPERAZIONI DI DISPATCH (ossia quando il processo passa da ready a running)

1)Cambio di contesto: viene salvato il PCB del processo che esce e viene caricato il PCB del processo che entra
 -è l'operazione di dispatch più costosa
 -ne avvengono migliaia al secondo
2)Passaggio alla modalità utente
3)Salto all'istruzione da eseguire del processo appena arrivato nella CPU

ESEMPIO DI CAMBIO DI CONTESTO

Ipotizziamo di avere i processi P0 e P1. P0 esce dalla CPU per far entrare P0. Viene salvato il PCB di P0 e viene caricato il PCB di P1

CREARE UN PROCESSO

- Un processo può creare un figlio
  -il figlio ottiene risorse dal SO o dal padre
- Due tipi di esecuzione:
  -**sincrona:** il padre attende la terminazione dei figli
  -**asincrona:** il padre continua la propria esecuzione parallelamente a quella dei figli

CREARE UN PROCESSO SU UNIX

- **System call fork:** crea un figlio clone del padre
  -il figlio deve essere più piccolo del padre per evitare di avere due processi completamente uguali
  -il PCB del figlio è diverso
- **System call exec:** cambia il codice del figlio
- **System call wait:** esecuzione sincrona tra padre e figlio

TERMINARE UN PROCESSO

- Un padre può terminare il figlio se:
  -usa troppe risorse
  -il suo compito non è più richiesto
- Se il padre muore, muoiono anche i figli
- **Un figlio NON PUÒ rimanere senza padre**

THREADS

Thread: unità minima di utilizzo della CPU, parte del codice di un processo (esso può essere diviso in più threads)

- Ogni thread ha:
  -thread control block (simile a PCB)
  -user stack
- I thread condividono:
  -spazio di indirizzamento
  -risorse e stato del processo
- Tutti gli SO moderni supportano più thread per un singolo processo (multithreading)
  -processi multi-thread fatti per lavorare su sistemi multi processore

VANTAGGI DEI THREAD

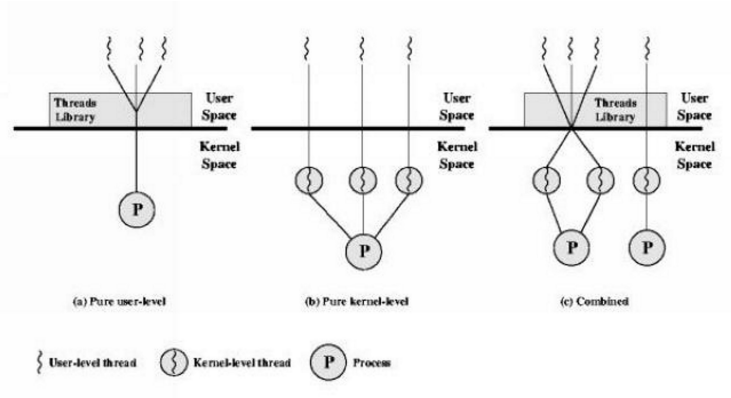
- Più veloci dei processi
  -creazione, terminazione e context switch più veloci
- Condividono risorse
  -thread di uno stesso processo condividono memoria
- Più operazioni allo stesso tempo
  -mentre un thread è bloccato (es. I/O o elaborazione lunga) un altro thread può continuare a lavorare
- Scalabilità
  -un thread in esecuzione su ogni processore

STATI DI UN THREAD

•Come in un processo:
-ready
-running
-waiting
•**Lo stato del processo può non coincidere con lo stato del thread**

IMPLEMENTAZIONE DEI THREAD

•Tre modi:
-Thread a livello utente
-Thread a livello kernel
-Thread in entrambi i livelli



THREAD A LIVELLO UTENTE

•**Vantaggi:**
-gestiti da un unico processo
-non è necessario passare in modalità kernel per utilizzare thread
-portabilità: girano su qualunque SO
•**Svantaggi:**
-se si blocca un thread, si blocca tutto il processo
-niente parallelismo: viene eseguito un solo thread alla volta per processo

THREAD A LIVELLO KERNEL

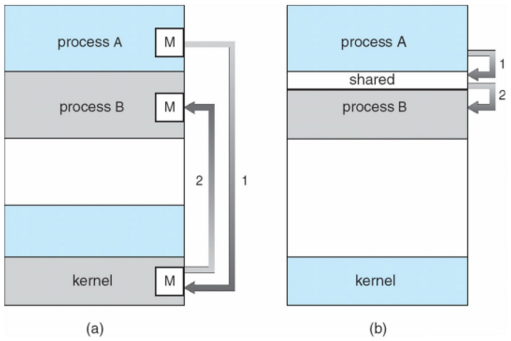
•**Vantaggi:**
-parallelismo: più thread dello stesso processo in esecuzione su CPU diverse
-il blocco di un thread non blocca il processo
•**Svantaggi:**
-poco efficiente: per ogni thread a livello kernel c'è un thread a livello utente

RELAZIONI TRA PROCESSI E IPC

•I processi possono essere di 2 tipi:
-Processi indipendenti: NON comunicano con gli altri processi
-Processi cooperanti: comunicano con gli altri processi

•I processi cooperanti possono comunicare tra loro con 2 tecniche dell'IPC (Inter-Process Communication):

-**Scambio di messaggi**
-**Condivisione della memoria**



Modelli di comunicazione tra processi basati su (a) scambio di messaggi, e (b) condivisione della memoria.

IPC - MESSAGE PASSING

- Lo scambio di messaggi avviene tramite il kernel che fa da mediatore per spedire i messaggi
  - tutti i messaggi del processo A devono passare dal kernel prima di arrivare al processo B
  - viene fatto per ragioni di sicurezza (es. evitare che A spedisca qualcosa che consumi risorse di B)
- 2 operazioni fornite dall'IPC:
  - send(message)
  - receive(message)
- Può avvenire in 2 modi:
  - **diretta**
  - **indiretta**

COMUNICAZIONE DIRETTA

- Può avvenire in 2 modi:
  - **simmetrica:** mittente e destinatario conoscono l'uno il nome dell'altro
  - **asimmetrica:** destinatario non conosce il nome del mittente ma può ricevere messaggi da più processi
- L'unico svantaggio è che se un processo cambia nome, bisogna cambiare il codice

COMUNICAZIONE INDIRETTA

- I processi comunicano tramite una mailbox
  - **send(A, message):** spedisce un messaggio alla mailbox A
  - **receive(A, message):** riceve un messaggio dalla mailbox A
- Più processi possono condividere la stessa mailbox
  - in questo caso bisogna decidere a chi va il messaggio

SINCRONIZZAZIONE

Lo scambio di messaggi può essere:

- **Bloccante:**
  - mittente si blocca finchè il messaggio è ricevuto
  - ricevente bloccato finchè non riceve conferma dalla mailbox o dal mittente
- **Non Bloccante:**
  - mittente continua la sua esecuzione dopo aver mandato il messaggio
  - ricevente non aspetta nessun ACK

IPC - MEMORIA CONDIVISA

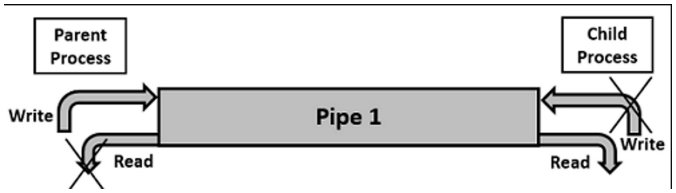
- Processo A condivide la sua memoria heap per fare in modo che entrambi i processi possano scrivere e leggere dati
- Più veloce dello scambio di messaggi ma meno sicuro
- È necessaria sincronizzazione per gli accessi per evitare conflitti
- I due processi comunicano tramite Pipe

PIPE

- Pipe: metodo di comunicazione in cui un processo scrive ad un'estremita e l'altro legge all'altra estremità

Si dividono in due tipologie:

- **Pipe ordinarie:**
  - unidirezionali
  - funzionano solo tra processi padre-figlio



- **Pipe con nome:**
  - bidirezionali
  - funzionano con tutti i processi

GESTIONE DEI PROCESSI NEL SISTEMA OPERATIVO

•Il SO è un programma

Parti del SO possono o non possono essere considerate dei processi, dipende dall'implementazione:

•Kernel separato:

-SO separato dai processi utente in uno spazio di memoria riservato
-concetto di processo applicabile solo ai processi utente
-tipico dei primi SO

•Kernel in modalità utente:

-ogni processo ha un kernel stack nella sua immagine (contiene funzionalità del SO eseguibili come codice protetto)
-dati del SO condivisi tra processi
-le chiamate di sistema richiedono un mode switch (da user mode a kernel mode) anzicchè un context switch (da processo utente a processo di SO, più lento)

•Kernel come processo:

-ogni parte del SO viene eseguita come processo
-solo una parte del SO viene eseguita al di fuori dei processi (scheduler)
-usato nei sistemi multiprocessore (multicore)