03 - Processi e Thread

Concetto di processo

Il processo e' un programma di esecuzione. E' l'unita' di esecuzione all'interno del sistema operativo. Solitamente, esecuzione sequenziale (istruzioni vengono eseguite in sequenza, secondo l'ordine specificato nel testo del programma).
Un SO multiprogrammato consente l'esecuzione concorrente di piu processi.

Programma = entita' passiva
Processo = entita' attiva

Il processo e' rappresentato da:

- codice (text) del programma eseguito
- dati: variabili globali
- program counter
- alcuni registri della CPU
- stack: parametri, variabili locali a funzioni/procedure

Inoltre, ad ogni processo possono essere associate delle risorse di SO. Ad esempio:

- file aperti
- connessioni di rete
- altri dispositivi di I/O in uso
- ...

Stati di un processo

Un processo, durante la sua esistenza puo' trovarsi in cari stati:

- **init**: stato transitorio durante il quale il processo viene caricato in memoria e SO inizializza i dati che lo rappresentano (stato iniziale)
- **Ready**: processo e' pronto per acquisire la CPU
- **Running**: processo che sta utilizzando la CPU
- **Waiting**: processo e' sospeso in attesa di un evento
- **Terminated:** stato transitorio relativo alla fase di terminazione e deallocazione del processo dalla memoria

In un sistema monoprocessore e multiprogrammato:

- un solo processo (al massimo) si trova nello stato di running

- piu' processi possono trovarsi negli stati ready e waiting

necessita' di strutture dati per mantenere in memoria le informazioni su processi in attesa di acquisire la CPU (ready) o di eventi (waiting). Per tutto questo e' necessario avere dei descrittori di processo (strutture dati che descrivono lo stato di ciascun processo).

Rappresentazione dei processi

- Ad ogni processo viene associata una struttura dati (descrittore): **Process Control Block (PCB)**
- PCB contiene tutte le informazioni relative al processo:
  - Stato del processo
  - Program Counter
  - Contenuto dei registri di CPU
  - Informazioni di scheduling
  - informazioni per gestore di memoria
  - Informazioni relative all'I/O
  - Informazioni di accouting
  - ...

Il sistema ha per ogni processo un PCB

Scheduling dei processi

E' l'attivita' mediante la quale un SO effettua delle scelte tra i processi, riguardo a:

- caricamento in memoria centrale
- assegnazione della CPU

In generale un sistema operativo compie tre diverse attivita' di scheduling

- scheduling a breve termine (o di CPU)
- scheduling a medio termine (o swapping)
- scheduling a lungo termine

Scheduler a lungo termine

Lo scheduler a lungo termine e' quella componente del SO che seleziona i programmi da eseguire dalla memoria secondaria per caricarli in memoria centrale (creando i corrispondenti processi):

- controlla il grado di multiprogrammazione (numero di processi contemporaneamente presenti nel sistema presenti nel sistema)
- e' una componente importante dei sistemi batch multiprogrammati
- nei sistemi time sharing

Interattivita': spesso e' l'utente che stabilisce direttamente il grado di multiprogrammazione --> scheduler a lungo termine non e' presente

Scheduler a medio termine

Nei sistemi operativi multiprogrammati:

- quantita' di memoria fisica puo' essere minore della somma delle dimensioni degli spazi logici di indirizzi da allocare a ciascun processo
- grado di multiprogrammazione non e', in generale, vincolato dalle esigenze di spazio dei processi

Swapping: trasferimento temporaneo in memoria secondaria di processi (o parti di processi), in modo da consentire l'esecuzione di altri processi

Scheduler a breve termine (o di CPU)

E' quella parte del SO che si occupa della selezione dei processi a cui assegnare la CPU

Nei sistemi time sharing, allo scadere di ogni quanto di tempo, SO:

- decide a quale processo assegnare la CPU (scheduling di CPU)
- effettua il cambio contesto (context switch)

Cambio di contesto

E' la fase in cui l'uso della CPU viene commutato da un processo ad un altro

Quando avviene un cambio di contesto tra un processo P_i ad un processo P_{i+1}:

- Salvataggio dello stato di P_i : SO copia Pc, registri, ... del processo deschedulato P_i nel suo PCB
- Ripristino dello stato di P_{i+1}: SO trasferisce i dati del processo P_{i+1} dal suo PCB nei registri di CPU, che puo' cosi' riprendere l'esecuzione

Il passaggio da un processo al successivo puo' richiedere onerosi trasferimenti da/verso la memoria secondaria, per allocare/deallocare gli spazi di indirizzi di processi.

Scheduler a breve termine

Lo scheduler a breve termine gestisce:

- la coda dei processi pronti: contiene i PCB dei processi che si trovano in stato Ready
- Altre strutture necessarie
    - code di waiting (una per ogni tipo di attesa: dispositivi I/O, timer, ...): ognuna di esse contiene i PCB dei processi waiting in attesa di un evento del tipo associato alla coda

Scheduler

- **Scheduler short-term** scheduler viene invocato con alta frequenza (ms) --> deve essere molto efficiente
- **Scheduler medium-term** viene invocato a minore frequenza (sec-min) --> puo' essere anche piu' lento

Scelte ottimali di scheduling dipendono dalla tipologia di processi:

- **processi I/O-bound** - maggior parte del tempo di operazioni I/O, molti burst brevi di CPU
- **processi CPU-bound** - maggior parte del tempo in uso CPU, pochi burst CPU tipicamente molto lunghi

Code di Scheduling

Coda dei processi pronti (ready queue):

- strategia di gestione della ready queue dipende dalle politiche (Algoritmi) di scheduling adottate dal SO

Scheduling e cambio di contesto

Operazioni di scheduling determinano un costo computazionale aggiuntivo che dipende essenzialmente da:

- frequenza di cambio contesto
- dimensione PCB
- costo dei trasferimenti da/verso la memoria
    - esistono SO che prevedono processi leggeri (thread) che hanno la proprieta' di condividere codice e dati cona altri processi:
        - dimensione PCB ridotta
        - riduzione overhead

Operazione sui processi

Ogni SO multiprogrammato prvede dei meccanismi per la gestione dei processi.

Meccanismi necessari:

- creazione
- terminazione
- interazione tra processi

Sono operazioni privilegiate (esecuzione in modo kernel) --> definizione di system call

Creazione di processi

Un processo (padre) puo' chiedere la creazione di un nuovo processo (figlio). Da qui e' possibile realizzare gerarchie di processi

Relazione padre figlio

Vari aspetti/soluzioni:

- **concorrenza**
  - padre e figlio procedono in parallelo (es. UNIX), oppure
  - il padre si sospende in attesa della terminazione del figlio
- **condivisione delle risorse**
  - le risorse del padre (ad esempio, i file aperti) sono condivise con i figli (es. UNIX), oppure
  - il figlio utilizza le risorse soltanto se esplicitamente richieste da se stesso
- **spazio degli indirizzi**
  - duplicato: lo spazio degli indirizzi del figlio e' una copia di quello del padre (es. fork() in UNIX), oppure
  - differenziato: spazi degli indirizzi di padre e figlio con codice e dati diversi (es. VMS, stesso processo dopo exec() in UNIX)

Terminazione

Ogni processo:

- e' figlio di un altro processo
- puo' essere a sua volta padre di processi

SO deve mantenere le informazioni relative alle relazioni di parentela (nel descrittore: riferimento al padre)

Se un processo termina:

- il padre puo' rilevare il suo stato di terminazione
- i figli adottati da init (o da un processo ancestor, se richiesto esplicitamente tramite dunzione prctl)

Processi leggeri (thread)

Un thread (o processo leggero) e' un unita' di esecuzione che condivide codice e dati con altri thread ad esso associati.

Task = insieme di thread che riferiscono lo stesso codice e gli stessi dati

Thread = {PC, registri, stack, ...}
Task = {thread1, thread2, ..., threadN, text, dati}

Un processo pesante e' equivalente ad un task con un solo thread

Vantaggi dei thread

- **Condivisione memoria**: a differenza dei processi pesanti, un thread puo' condividere variabili con altri (appartenenti allo stesso task)
- **Miglior costo di context switch**: PCB di thread non contiene alcuna informazione relativa a codice e da cambio di contesto fra thread dello stesso task ha un costo notevolmente inferiore al caso dei processi pesanti
- **Minor protezione**: thread appartenenti allo stesso task possono modificare dati gestiti da altri thread

Realizzazione di thread

Alcuni SO offrono anche l'implementazione del concetto di thread (es. MSWinXP, Linux, Solaris)

Realizzazione

- A livello di kernel (MSWinXP, Linux, Solaris, MacOSX):
  - SO gestisce direttamente i cambi di contesto
    - tra thread dello stesso task (trasferimento di registri)
    - tra task
  - SO fornisce strumenti per la sicnronizzazione nell'accesso di thread a variabili comuni
- A livello utente (es. Andrew - Carnegie Mellon, POSIX pthread, vecchie versioni MSWin, Java thread):
  - il passaggio da un thread al successivo (nello steso task) richiede interruzioni al SOO (maggior rapidita')
  - SO vede processi pesanti: minor efficienza
    - es. sospensione di un thread
    - Cambio di contesto tra thread di task diversi
- Soluzioni miste
  - thread realizzati a entrambi i livelli contemporaneamente

Interazione tra processi

I processi, pesanti o leggeri, possono interagire

Classificazione

- processi indipendenti: due processi P1 e P2 sono indipendenti se l'esecuzione di P1 non e' influenzata da P2, e viceversa
- processi interagenti: P1 e P2 sono interagenti se l'esecuzione di P1 e' influenzata dall'esecuzione di P2, e/o viceversa

Processi interagenti

Tipi di interazione:

- **Cooperazione**: l'interazione consiste nello scambio di informazioni al fine di seguire un'attivita' comune
- **Competizione**: i processi interagiscono per sincronizzarsi nell'accesso a risorse comuni
- Interferenza: interazione non desiderata e potenzialmente deleteria tra i processi

Supporto all'interazione:
L'interazione puo' avvenire mediante

- **memoria condivisa** (modello ambiente globale): SO consente ai processi (thread) di condividere variabili, l'interazione avviene tramite l'accesso a variabili condivise
- **scambio di messaggi** (modello ad ambiente locale): i processi non condividono variabili e interagiscono mediante meccanismi di trasmissione/ricezione di messaggi; SO prevede dei meccanismi di supporto dello scambio di messaggi

Aspetti:

- concorrenza --> velocita'
- suddivisione dei compiti tra processi --> modularita'
- condivisione di informazioni
    - assenza di replicazione: ogni processo accede alle stesse istanze di dati
    - necessita' di sincronizzare i processi nell'access a dati condivisi

Processi cooperanti

Esempio: produttore & consumatore
Due processi accedono a un buffer condiviso di dimensione limitata:

- un processo svolge il ruolo di produttore di informazione che verranno prelevate dall'altro consumatore
- buffer rappresenta un deposito di informazioni condiviso

Produttore & consumatore

Necessita' di sincronizzare i processi:

- quando il buffer e' vuoto --> il cosumatore NON puo' prelevare messaggi
- quando il buffer e' pieno --> il produttore NON puo' depositare messaggi