I Processi

Il Concetto di Processo

Il processo è un programma in esecuzione

- È l'**unità di esecuzione** all'interno del SO
- Solitamente, l'esecuzione di un processo è **sequenziale** (le istruzioni vengono eseguite in sequenza, secondo l'ordine specificato nel testo del programma)
- Un SO multiprogrammato consente l'esecuzione **concorrente** di più processi

D'ora in poi faremo implicitamente riferimento al caso di SO multiprogrammati

Il Concetto di Processo

Programma = entità passiva
Processo = entità attiva

Il processo è rappresentato da:

- **codice** (*text*) del programma eseguito
- **Dati:** variabili globali
- **Program Counter**
- Alcuni **registri** di CPU
- **Stack**: parametri, variabili locali a funzioni/procedure

Il Concetto di Processo

Processo = {PC, registri, stack, text, dati}

Inoltre, a un processo possono essere associate delle risorse del sistema operativo. Ad esempio:

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

Stati di un processo

Un processo, durante la sua esistenza può trovarsi in vari **stati**:

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

Stati di un processo

Transizioni di stato:

Stati di un processo

Transizioni di stato:

Stati di un Processo

In un sistema **monoprocessore/multiprogrammato**:

- un solo processo (al massimo) si trova nello stato *running*
- più processi possono trovarsi negli stati *ready* e *waiting*

  necessità di strutture dati per mantenere in memoria le informazioni su processi in attesa
  - di acquisire la CPU (ready)
  - di eventi (waiting)

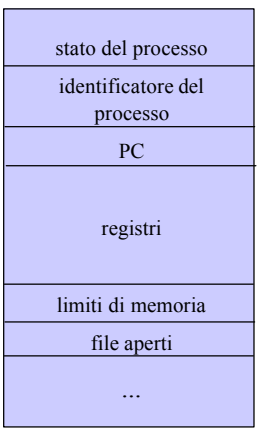
  ⇒ **Descrittore di processo**

Rappresentazione dei Processi

Come vengono rappresentati i processi nel sistema operativo?

- Ad ogni processo viene associata una struttura dati (descrittore): **Process Control Block** (**PCB**)
- Il **PCB** contiene tutte le informazioni relative al processo:
  - Stato del Processo
  - Contenuto dei registri di CPU (PC, SP, IR, accumulatori, etc.)
  - Informazioni di scheduling (priorità, puntatori alle code, etc.)
  - Informazioni per gestore di memoria (registri base, limite, etc.)
  - Informazioni relative all'I/O (risorse allocate, file aperti, etc.)
  - Informazioni di accounting (tempo di CPU utilizzato, etc.)
  - ...

Process Control Block



Il sistema operativo gestisce i PCB di tutti i processi, organizzandoli in opportune strutture dati (ad es., *code* di processi)

Scheduling dei processi

È l'attività mediante la quale il sistema operativo effettua delle scelte tra i processi, riguardo a:

- caricamento in memoria centrale
- assegnazione della CPU

In generale, il sistema operativo compie **tre diverse attività 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 è 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)
- è una componente importante dei sistemi **batch multiprogrammati**
- nei sistemi **time sharing**
  - **Interattività**: spesso è l'utente che stabilisce direttamente il grado di multiprogrammazione => lo scheduler a lungo termine non è presente

Scheduler a medio termine (swapper)

Nei sistemi operativi multiprogrammati:

- la quantità di memoria fisica può essere minore della somma delle dimensioni degli spazi logici di indirizzi da allocare a ciascun processo
- Il grado di multiprogrammazione non è, in generale, vincolato dalle esigenze di spazio dei processi

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

Scheduler a breve termine (o di CPU)

È 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, il SO:

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

Cambio di Contesto

È 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} (ovvero, P_i cede l'uso della CPU a P_{i+1}):

- **Salvataggio dello *stato* di P_i**: il SO copia PC, registri, etc. del processo ***deschedulato*** P_i nel suo PCB
- **Ripristino dello *stato* di P_{i+1}**: il SO trasferisce i dati del processo P_{i+1} dal suo PCB nei registri di CPU, che può così riprendere l'esecuzione

➔ Il passaggio da un processo al successivo può richiedere onerosi trasferimenti da/verso la memoria secondaria, per allocare/deallocare gli spazi di indirizzi dei processi (vedi gestione della memoria)

Scheduler a breve termine (o di CPU)

Lo scheduler a breve termine gestisce :

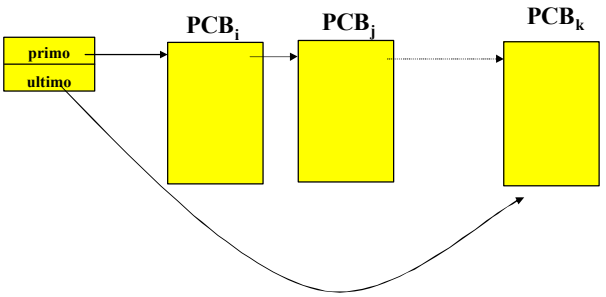
- la **coda dei processi pronti:** contiene i PCB dei processi che si trovano in stato *Ready*

Altre strutture dati necessarie:

- le **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

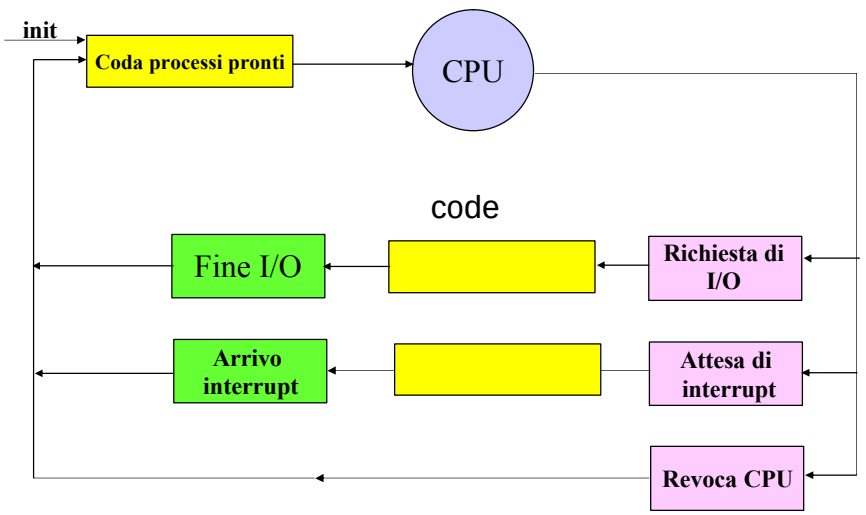
Code di Scheduling

Coda dei processi pronti (*ready queue*):



la strategia di gestione della ***ready queue*** dipende dalle **politiche** (algoritmi) di scheduling adottate dal SO

Diagramma di accodamento



Scheduling e Cambio di Contesto

Il cambio di contesto determina un costo computazionale aggiuntivo che dipende essenzialmente da:

- **frequenza** di commutazione della CPU ($1/\Delta t$)
- dimensione del **PCB**
- costo dei trasferimenti da/verso la memoria
  - ➔esistono sistemi operativi che prevedono **processi *leggeri*** (***thread***) che hanno la proprietà di condividere codice e dati con altri processi:
    - dimensione del PCB ridotta
    - riduzione dell'overhead

Operazioni sui Processi

Ogni SO multiprogrammato prevede dei meccanismi per la gestione dei processi

Meccanismi necessari:

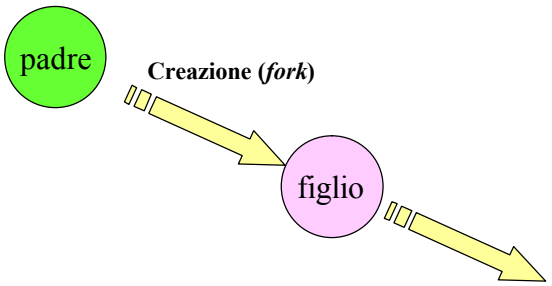
- creazione
- terminazione
- interazione tra processi

Sono operazioni privilegiate (esecuzione in modo kernel)

➔ definizione di **system call**

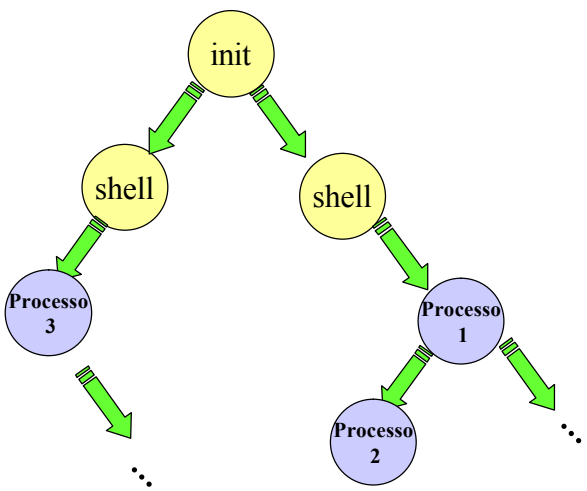
Creazione di Processi

- Un processo (***padre***)può richiedere la creazione di un nuovo processo (***figlio***)



- È possibile realizzare **gerarchie** di processi

Gerarchie di Processi (es. UNIX)



Relazione Padre-Figlio

Parametri:

- **concorrenza**
  - padre e figlio procedono in ***parallelo*** (es. UNIX), oppure
  - il padre ***si sospende*** in attesa della terminazione del figlio
- **condivisione di risorse**
  - le risorse del padre (ad esempio, i file aperti) sono ***condivise*** con i figli (es. UNIX), oppure
  - il figlio utilizza risorse soltanto se esplicitamente richieste da se stesso
- **spazio degli indirizzi**
  - ***duplicato***: lo spazio degli indirizzi del figlio è 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, exec() in UNIX)

Terminazione

Ogni processo:

- è **figlio** di un altro processo
- può essere a sua volta **padre** di processi

Il sistema operativo deve mantenere le informazioni relative alle relazioni di *parentela*

➔ nel descrittore: **riferimento al padre**

Se un processo termina: il padre può rilevare il suo stato di terminazione

Processi *Pesanti*

Da quanto visto finora:

Processo = {PC, registri, text, dati, stack, ...}

In generale, quindi:

- ***processi*** diversi eseguono codici distinti
- ***processi*** diversi accedono a dati diversi

➔ i ***processi*** **non condividono memoria**

Processi con queste caratteristiche vengono anche chiamati:

PROCESSI PESANTI

- Il costo del cambio di contesto per processi pesanti può essere elevato (dimensione del descrittore, accessi in memoria)

Thread (Processi Leggeri)

Un **thread** (o processo leggero) è un'unità 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

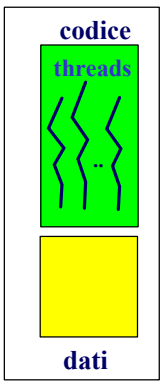
➔ **codice e dati non sono caratteristiche del singolo thread, ma del task al quale appartengono**

Quindi:

Thread = {PC, registri, stack, ...}

Task = {thread1, thread2, ..., threadN, text, dati}

Processi Leggeri (Thread)



task

- Il **processo pesante** equivale a un task con un solo thread

Vantaggi dei Thread

- **Condivisione di memoria**: a differenza dei processi (*pesanti*), un thread può condividere variabili con altri (appartenenti allo stesso task)
- **Minor costo di context switch**: il PCB di un thread non contiene alcuna informazione relativa a codice e dati
  - ➔ il cambio di contesto 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 sistemi operativi realizzano il concetto di thread (es. NT, **Linux**, Solaris)

Realizzazione

A livello kernel (NT, OS2, Linux):

- il SO gestisce direttamente i cambi di contesto
  - tra thread dello stesso task (trasferimento di registri)
  - tra task
- il SO fornisce strumenti per la sincronizzazione nell'accesso di thread a variabili comuni

Realizzazione di Thread

Realizzazione

- **A livello utente (es. Andrew - Carnegie Mellon):**
  - il passaggio da un thread al successivo (nello stesso task) non richiede interruzioni al SO (**maggior rapidità**)
  - il SO ***vede* processi pesanti**: **minore efficienza**
    - Es. sospensione di un thread
    - Cambio di contesto tra thread di task diversi
- **Soluzioni Miste (es. Solaris 2):**
  - i thread possono essere realizzati a entrambi i livelli

Interazione tra processi

I processi *possono*, in generale, interagire

Classificazione

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

Processi interagenti

Tipi di interazione:

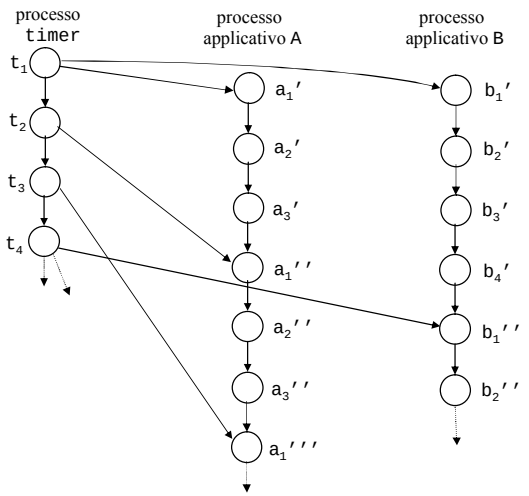
- **Cooperazione: interazione prevedibile e desiderata, insita nella logica del programma concorrente.**
- **Competizione: interazione prevedibile ma "non desiderata" tra processi che interagiscono per sincronizzarsi nell'accesso a risorse comuni**
- **Interferenza: interazione non prevista e non desiderata, potenzialmente deleteria tra processi**

Processi Cooperanti

- Esempio: temporizzazione di processi di controllo
  - sistema per il controllo di un impianto industriale: monitorare periodicamente alcune grandezze fisiche (temperature, pressioni, portate ecc.)
  - insieme di processi sequenziali concorrenti ciascuno dei quali è dedicato a monitorare una delle grandezze fisiche d'interesse.
  - ogni processo legge il valore della grandezza, verifica se tale valore rientra nel range di valori ammessi e, in caso contrario, intervenire per mezzo di appositi attuatori per riportare il valore nel range desiderato.

- Tutti i processi applicativi procedono concorrentemente.
- Ogni processo è periodico -> necessità di sincronizzazione
- **Processo Timer:** invia periodicamente segnali ai processi applicativi da attivare

Processi Cooperanti



Competizione

- Esempio: accesso a una stampante condivisa
  - due processi A e B che condividono l'uso della risorsa stampante.
  - Indichiamo con:

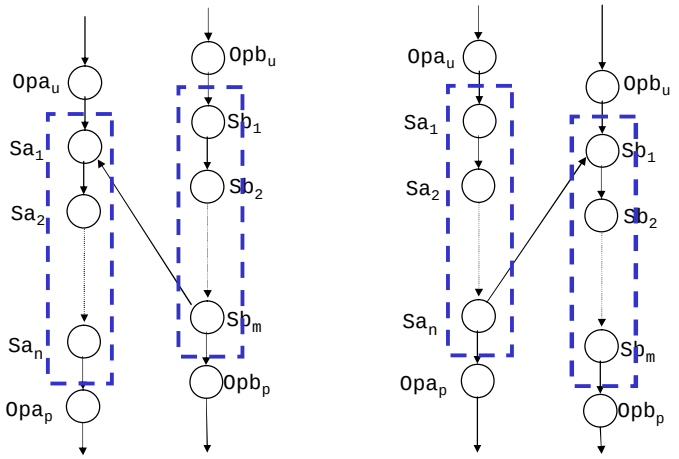
    Sa1, Sa2, ..., San

    Sb1, Sb2, ..., Sbm

    le azioni corrispondenti all'esecuzione da parte dei processi A e B delle istruzioni componenti la funzione di stampa.

  Per garantire una corretta competizione sulla stampante è necessario assicurare la **mutua esclusione** fra le esecuzioni della funzione stampa da parte dei due processi.

Mutua esclusione



Cooperazione e competizione

In entrambi gli esempi i processi interagenti sincronizzano le loro velocità.

Differenza:

- la cooperazione prevede uno schema di sincronizzazione fisso
- nella competizione sono consentite due diverse forme di sincronizzazione, entrambe corrette.
  - A seconda delle velocita` di esecuzione dei due processi verra` attuata una forma o l'altra.

Processi interagenti

Supporto all'interazione

L'interazione può avvenire mediante

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