Sistemi Operativi

Modulo 2: Concorrenza

Sezione 1

1. Introduzione alla concorrenza

Introduzione

- Un sistema operativo consiste in un gran numero di *attività* che vengono eseguite più o meno *contemporaneamente* dal processore e dai dispositivi presenti in un elaboratore.
- Senza un modello adeguato, la coesistenza delle diverse attività sarebbe difficile da descrivere e realizzare.
- Il modello che è stato realizzato a questo scopo prende il nome di *modello concorrente* ed è basato sul concetto astratto di *processo*

Processi e programmi

- **Definizione: *processo***
  - E' un'attività controllata da un programma che si svolge su un processore
- **Un processo non è un programma!**
  - Un programma è un entità *statica*, un processo è *dinamico*
  - Un programma:
    - specifica un'insieme di istruzioni e la loro sequenza di esecuzione
    - non specifica la distribuzione nel tempo dell'esecuzione
  - Un processo:
    - rappresenta il modo in cui un programma viene eseguito nel tempo
- **Assioma di *finite progress***
  - Ogni processo viene eseguito ad una velocità finita ma sconosciuta

Stato di un processo

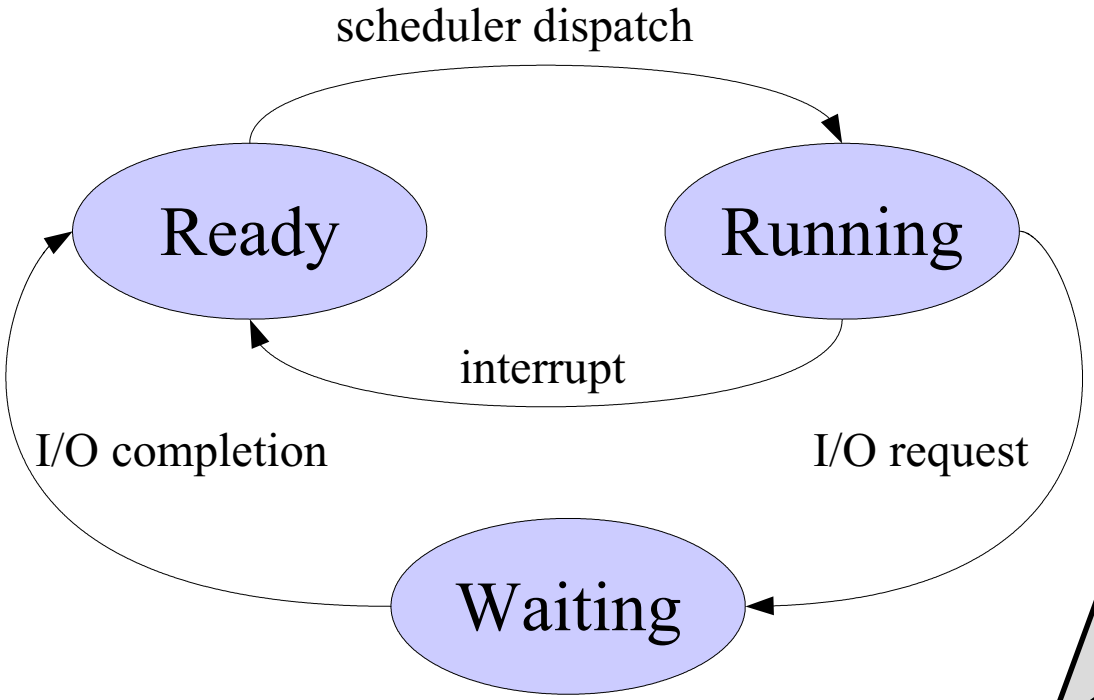
- **Ad ogni istante, un processo può essere totalmente descritto dalle seguenti componenti:**
  - *La sua immagine di memoria*
    - la memoria assegnata al processo (ad es. testo, dati, stack)
    - le strutture dati del S.O. associate al processo (ad es. file aperti)
  - *La sua immagine nel processore*
    - contenuto dei registri generali e speciali
  - *Lo stato di avanzamento*
    - descrive lo stato corrente del processo: ad esempio, se è in esecuzione o in attesa di qualche evento

Processi e programmi (ancora)

- **Più processi possono eseguire lo stesso programma**
  - In un sistema multiutente, più utenti possono leggere la posta contemporaneamente
  - Un singolo utente può eseguire più istanze dello stesso editor
- **In ogni caso, ogni istanza viene considerata un processo separato**
  - Possono condividere lo stesso codice ...
  - ... ma i dati su cui operano, l'immagine del processore e lo stato di avanzamento sono separati

Stati dei processi (versione semplice)

- **Stati dei processi:**
  - *Running*: il processo è in esecuzione
  - *Waiting*: il processo è in attesa di qualche evento esterno (ad es. completamento operazione di I/O); non può essere eseguito
  - *Ready*: il processo può essere eseguito, ma attualmente il processore è impegnato in altre attività



Nota: modello semplificato, nel seguito vedremo un modello più realistico

Cos'è la concorrenza?

- **Tema centrale nella progettazione dei S.O. riguarda la gestione di processi *multipli***
  - *Multiprogramming*
    - più processi su un solo processore
    - parallelismo *apparente*
  - *Multiprocessing*
    - più processi su una macchina con processori multipli
    - parallelismo *reale*
  - *Distributed processing*
    - più processi su un insieme di computer distribuiti e indipendenti
    - parallelismo *reale*

Cos'è la concorrenza?

- **Esecuzione concorrente:**
  - Due programmi si dicono in esecuzione concorrente se vengono eseguiti in parallelo (con parallelismo reale o apparente)
- **Concorrenza:**
  - E' l'insieme di notazioni per descrivere l'esecuzione concorrente di due o più programmi
  - E' l'insieme di tecniche per risolvere i problemi associati all'esecuzione concorrente, quali *comunicazione* e *sincronizzazione*

Dove possiamo trovare la concorrenza?

- **Applicazioni multiple**
  - la multiprogrammazione è stata inventata affinchè più processi indipendenti condividano il processore
- **Applicazioni strutturate su processi**
  - estensione del principio di progettazione modulare; alcune applicazioni possono essere progettate come un insieme di processi o thread concorrenti
- **Struttura del sistema operativo**
  - molte funzioni del sistema operativo possono essere implementate come un insieme di processi o thread

Multiprocessing e multiprogramming: differenze?

Prima di iniziare lo studio della concorrenza, dobbiamo capire se esistono differenze fondamentali nella programmazione quando i processi multipli sono eseguiti da processori diversi rispetto a quando sono eseguiti dallo stesso processore

Multiprocessing e multiprogramming: differenze?

- **In un singolo processore:**
  - processi multipli sono "*alternati nel tempo*" per dare l'impressione di avere un multiprocessore
  - ad ogni istante, al massimo un processo è in esecuzione
  - si parla di *interleaving*
- **In un sistema multiprocessore:**
  - più processi vengono eseguiti *simultaneamente* su processori diversi
  - i processi sono "*alternati nello spazio*"
  - si parla di *overlapping*

Multiprocessing e multiprogramming: differenze?

- **A prima vista:**
  - si potrebbe pensare che queste differenze comportino problemi distinti
  - in un caso l'esecuzione è simultanea
  - nell'altro caso la simultaneità è solo simulata
- **In realtà:**
  - presentano gli stessi problemi
  - che si possono riassumere nel seguente:

non è possibile predire la velocità relativa dei processi

Un esempio semplice

- **Si consideri il codice seguente:**

In C:

```
void modifica(int valore) {
  totale = totale + valore
}
```

In Assembly:

```
.text
modifica:
  lw $t0, totale
  add $t0, $t0, $a0
  sw $t0, totale
  jr $ra
```

- **Supponiamo che:**
  - Esista un processo P_1 che esegue `modifica(+10)`
  - Esista un processo P_2 che esegue `modifica(-10)`
  - P_1 e P_2 siano in esecuzione concorrente
  - `totale` sia una variabile condivisa tra i due processi, con valore iniziale 100
- **Alla fine, `totale` dovrebbe essere uguale a 100. Giusto?**

Scenario 1: multiprogramming (corretto)

P1	`lw $t0, totale`	totale=100, $t0=100, $a0=10
P1	`add $t0, $t0, $a0`	totale=100, $t0=110, $a0=10
P1	`sw $t0, totale`	totale=110, $t0=110, $a0=10
S.O.	`interrupt`	
S.O.	`salvataggio registri P1`	
S.O.	`ripristino registri P2`	totale=110, $t0=? , $a0=-10
P2	`lw $t0, totale`	totale=110, $t0=110, $a0=-10
P2	`add $t0, $t0, $a0`	totale=110, $t0=100, $a0=-10
P2	`sw $t0, totale`	totale=100, $t0=100, $a0=-10

Scenario 2: multiprogramming (errato)

P1	`lw $t0, totale`	totale=100, $t0=100, $a0=10
S.O.	`interrupt`	
S.O.	`salvataggio registri P1`	
S.O.	`ripristino registri P2`	totale=100, $t0=? , $a0=-10
P2	`lw $t0, totale`	totale=100, $t0=100, $a0=-10
P2	`add $t0, $t0, $a0`	totale=100, $t0= 90, $a0=-10
P2	`sw $t0, totale`	totale= 90, $t0= 90, $a0=-10
S.O.	`interrupt`	
S.O.	`salvataggio registri P2`	
S.O.	`ripristino registri P1`	totale= 90, $t0=100, $a0=10
P1	`add $t0, $t0, $a0`	totale= 90, $t0=110, $a0=10
P1	`sw $t0, totale`	totale=110, $t0=110, $a0=10

Scenario 3: multiprocessing (errato)

- I due processi vengono eseguiti simultaneamente da due processori distinti

Processo P1:

```
lw $t0, totale

add $t0, $t0, $a0
sw $t0, totale
```

Processo P2:

```
lw $t0, totale
add $t0, $t0, $a0

sw $t0, totale
```

- **Nota:**
  - i due processi hanno insiemi di registri distinti
  - l'accesso alla memoria su **`totale`** non può essere simultaneo

Alcune considerazioni

- **Non vi è sostanziale differenza tra i problemi relativi a multiprogramming e multiprocessing**
  - ai fini del ragionamento sui programmi concorrenti si ipotizza che sia presente un "processore ideale" per ogni processo
- **I problemi derivano dal fatto che:**
  - non è possibile predire gli istanti temporali in cui vengono eseguite le istruzioni
  - i due processi accedono ad una o più risorse condivise

Race condition

- **Definizione**
  - Si dice che un sistema di processi multipli presenta una *race condition* qualora il risultato finale dell'esecuzione dipenda dalla temporizzazione con cui vengono eseguiti i processi
- **Per scrivere un programma concorrente:**
  - è necessario eliminare le race condition

Considerazioni finali

- **In pratica:**
  - scrivere programmi concorrenti è più difficile che scrivere programmi sequenziali
  - la correttezza non è solamente determinata dall'esattezza dei passi svolti da ogni singola componente del programma, ma anche dalle interazioni (volute o no) tra essi
- **Nota:**
  - Fare debug di applicazioni che presentano race condition non è per niente piacevole...
  - Il programma può funzionare nel 99.999% dei casi, e bloccarsi inesorabilmente quando lo discutete con il docente all'esame...
  - (... un corollario alla legge di Murphy...)

Notazioni per descrivere processi concorrenti - 1

- **Notazione esplicita**

```
process nome {
   ... statement(s) ...
}
```

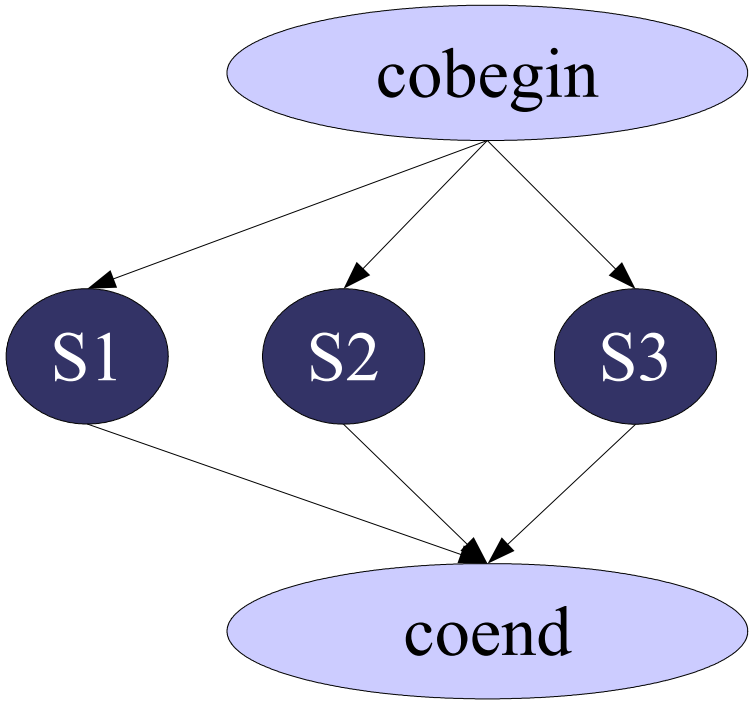
- **Esempio**

```
process P1 {
   totale = totale + valore;
}
process P2 {
   totale = totale - valore;
}
```

Notazioni per descrivere processi concorrenti - 2

- **Notazione cobegin/coend**

```
cobegin
   ... S1 ...
//
   ... S2 ...
//
   ... S3 ...
//
   ...
coend
```



- **Ogni statement viene eseguito in concorrenza**
- **Le istruzioni che seguono il coend verranno eseguite solo quando tutti gli statement sono terminati**

2. Interazioni tra processi

Interazioni tra processi

- **E' possibile classificare le modalità di interazione tra processi in base a quanto sono "*consapevoli*" uno dell'altro.**
- **Processi totalmente "ignari" uno dell'altro:**
  - processi indipendenti non progettati per lavorare insieme
  - sebbene siano indipendenti, vivono in un ambiente comune
- **Come interagiscono?**
  - *competono* per le stesse risorse
  - devono *sincronizzarsi* nella loro utilizzazione
- **Il sistema operativo:**
  - deve arbitrare questa *competizione*, fornendo meccanismi di *sincronizzazione*

Interazioni tra processi

- **Processi "indirettamente" a conoscenza uno dell'altro**
  - processi che condividono risorse, come ad esempio un buffer, al fine di scambiarsi informazioni
  - non si conoscono in base ai loro id, ma interagiscono indirettamente tramite le risorse condivise
- **Come interagiscono?**
  - *cooperano* per qualche scopo
  - devono *sincronizzarsi* nella utilizzazione delle risorse
- **Il sistema operativo:**
  - deve facilitare la *cooperazione*, fornendo meccanismi di *sincronizzazione*

Interazioni tra processi

- **Processi "direttamente" a conoscenza uno dell'altro**
  - processi che comunicano uno con l'altro sulla base dei loro id
  - la comunicazione è diretta, spesso basata sullo scambio di messaggi
- **Come interagiscono**
  - *cooperano* per qualche scopo
  - *comunicano* informazioni agli altri processi
- **Il sistema operativo:**
  - deve facilitare la *cooperazione*, fornendo meccanismi di *comunicazione*

Proprietà

- **Definizione**
  - Una *proprietà* di un programma concorrente è un attributo che rimane vero per ogni possibile storia di esecuzione del programma stesso
- **Due tipi di proprietà:**
  - *Safety* ("nothing bad happens")
    - mostrano che il programma (se avanza) va "nella direzione voluta", cioè non esegue azioni scorrette
  - *Liveness* ("something good eventually happens")
    - il programma avanza, non si ferma... insomma è "*vitale*"

Proprietà - Esempio

- ***Consensus*, dalla teoria dei sistemi distribuiti**
  - Si consideri un sistema con **N** processi:
    - All'inizio, ogni processo *propone* un valore
    - Alla fine, tutti i processi si devono accordare su uno dei valori proposti (*decidono* quel valore)
- **Proprietà di safety**
  - Se un processo decide, deve decidere uno dei valori proposti
  - Se due processi decidono, devono decidere lo stesso valore
- **Proprietà di liveness**
  - Prima o poi ogni processo corretto (i.e. non in crash) prenderà una decisione

Proprietà - programmi sequenziali

- **Nei programmi sequenziali:**
  - le proprietà di *safety* esprimono la correttezza dello stato finale (il risultato è quello voluto)
  - la principale proprietà di *liveness* è la terminazione
- **Quali dovrebbero essere le proprietà comuni a tutti i programmi concorrenti?**

Proprietà - programmi concorrenti

- **Proprietà di *safety***
  - i processi non devono "*interferire*" fra di loro nell'accesso alle risorse condivise
  - questo vale ovviamente per i processi che condividono risorse (non per processi che cooperano tramite comunicazione)
- **I meccanismi di sincronizzazione servono a garantire la proprietà di safety**
  - devono essere usati propriamente dal programmatore, altrimenti il programma potrà contenere delle race condition

Proprietà - programmi concorrenti

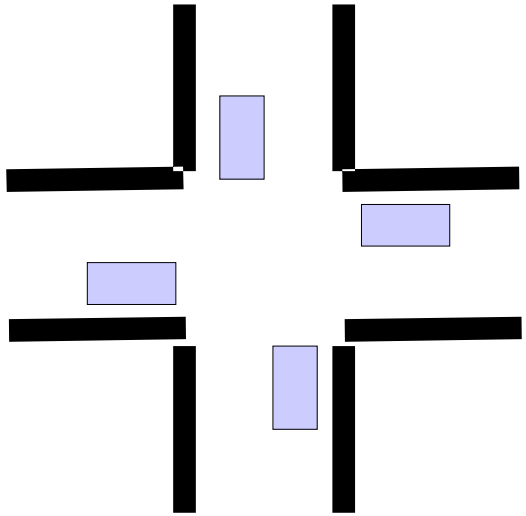
- **Proprietà di *liveness***
  - i meccanismi di sincronizzazione utilizzati non devono prevenire l'avanzamento del programma
    - non è possibile che *tutti* i processi si "*blocchino*", in attesa di eventi che non possono verificarsi perché generabili solo da altri processi bloccati
    - non è possibile che *un* processo debba "*attendere indefinitamente*" prima di poter accedere ad una risorsa condivisa
- **Nota:**
  - queste sono solo descrizioni informali; nei prossimi lucidi saremo più precisi

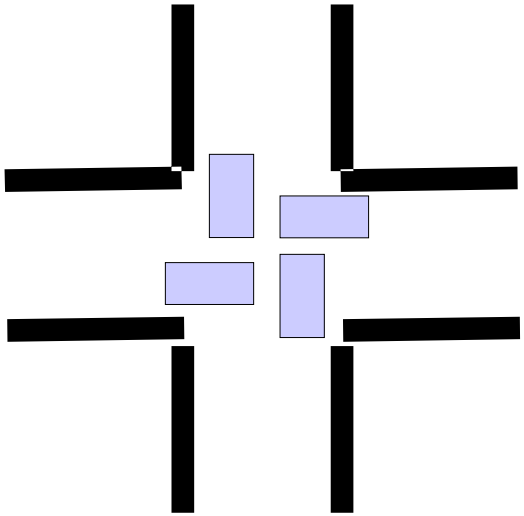
Mutua esclusione

- **Definizione**
  - l'accesso ad una risorsa si dice *mutualmente esclusivo* se ad ogni istante, al massimo un processo può accedere a quella risorsa
- **Esempi da considerare:**
  - due processi che vogliono accedere contemporaneamente a una stampante
  - due processi che cooperano scambiandosi informazioni tramite un buffer condiviso

Deadlock (stallo)

- **Considerazioni:**
  - la mutua esclusione permette di risolvere il problema della non interferenza
  - ma può causare il blocco permanente dei processi
- **Esempio: incrocio stradale**





Deadlock (stallo)

- **Esempio:**
  - siano $\mathbf{R_1}$ e $\mathbf{R_2}$ due risorse
  - siano $\mathbf{P_1}$ e $\mathbf{P_2}$ due processi che devono accedere a $\mathbf{R_1}$ e $\mathbf{R_2}$ contemporaneamente, prima di poter terminare il programma
  - supponiamo che il S.O. assegni $\mathbf{R_1}$ a $\mathbf{P_1}$, e $\mathbf{R_2}$ a $\mathbf{P_2}$
  - i due processi sono bloccati in attesa circolare
- **Si dice che P_1 e P_2 sono in deadlock**
  - è una condizione da evitare
  - è definitiva
  - nei sistemi reali, se ne può uscire solo con metodi "distruttivi", ovvero uccidendo i processi, riavviando la macchina, etc.

Starvation (inedia)

- **Considerazioni:**
  - il deadlock è un problema che coinvolge tutti i processi che utilizzano un certo insieme di risorse
  - esiste anche la possibilità che un processo non possa accedere ad un risorsa perché "sempre occupata"
- **Esempio**
  - se siete in coda ad uno sportello e continuano ad arrivare "furbi" che passano davanti, non riuscirete mai a parlare con l'impiegato/a

Starvation (inedia)

- **Esempio**
  - sia **R** una risorsa
  - siano $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ tre processi che accedono periodicamente a **R**
  - supponiamo che $\mathbf{P_1}$ e $\mathbf{P_2}$ si alternino nell'uso della risorsa
  - $\mathbf{P_3}$ non può accedere alla risorsa, perché utilizzata in modo esclusivo da $\mathbf{P_1}$ e $\mathbf{P_2}$
- **Si dice che P3 è in *starvation***
  - a differenza del deadlock, non è una condizione definitiva
  - è possibile uscirne, basta adottare un'opportuna politica di assegnamento
  - è comunque una situazione da evitare

Riassunto

Tipo	Relazione	Meccanismo	Problemi di controllo
processi "ignari" uno dell'altro	competizione	sincronizzazione	mutua esclusione deadlock starvation
processi con conoscenza indiretta l'uno dell'altro	cooperazione (sharing)	sincronizzazione	mutua esclusione deadlock starvation
processi con conoscenza diretta l'uno dell'altro	cooperazione (comunicazione)	comunicazione	deadlock starvation

Riassunto

- **Nei prossimi lucidi:**
  - vedremo quali tecniche possono essere utilizzate per garantire mutua esclusione e assenza di deadlock e starvation
  - prima però vediamo di capire esattamente quando due o più processi possono interferire