Alcune definizioni

algoritmo: sequenza di passi che consentono di risolvere un problema

programma: descrizione di un algoritmo tramite un linguaggio che ne rende possibile l'esecuzione da parte di un processore

evento: esecuzione di una delle istruzioni del processore

processo: sequenza di eventi prodotti da un processore nell'esecuzione di un programma.

si noti che...

le definizioni date sono relative ad algoritmi, programmi e processi sequenziali

gli eventi di un processo sequenziale sono ad ordinamento totale

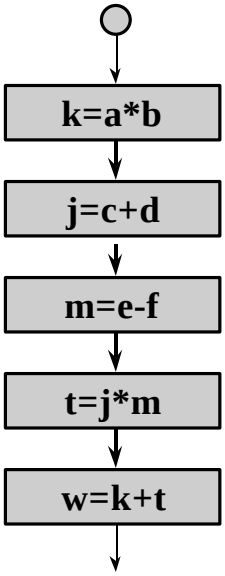
è ben raro che la soluzione di un problema richieda necessariamente l'esecuzione di un'unica sequenza di passi, ossia di un algoritmo sequenziale in quanto in generale esistono più soluzioni possibili in cui le azioni da compiere sono parzialmente ordinate tra loro

un esempio

Valutare l'espressione:

(a * b) + (c+d) * (e - f)

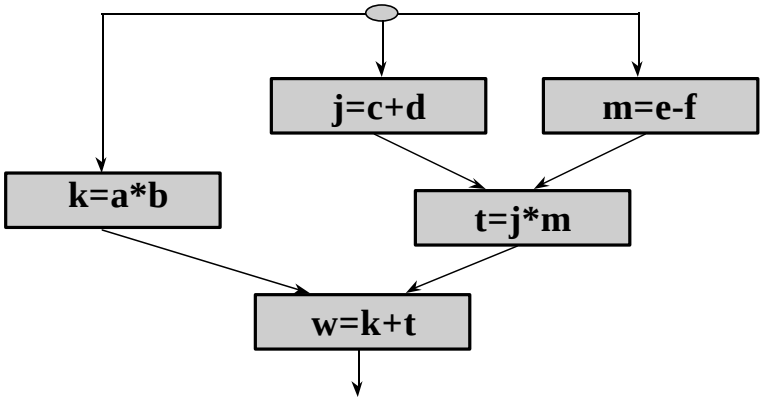
Vincoli di precedenza tra le azioni imposti in un possibile algoritmo sequenziale



un esempio

Valutare l'espressione: (a * b) + (c+d) * (e - f)

Vincoli di precedenza tra le azioni imposti dal problema



ne consegue che

in generale un problema può essere risolto con un algoritmo concorrente ossia con un algoritmo che definisce più sottosequenze di azioni da compiere che non sono tra loro totalmente ordinate e che quindi possono essere in parte eseguite in parallelo

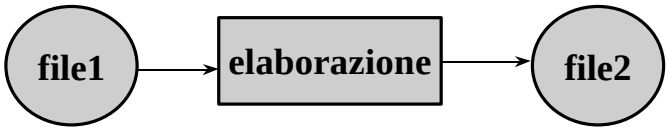
la soluzione di un problema mediante un algoritmo concorrente richiede la disponibilità di un linguaggio concorrente ossia di un linguaggio che consenta di descrivere le singole sottosequenze di azioni da eseguire e i relativi vincoli di dipendenza

ne consegue che

l'esecuzione di un programma scritto in un linguaggio concorrente è il risultato di più processi sequenziali ciascuno dei quali è il risultato dell'esecuzione di una delle sottosequenze di azioni definite dall'algoritmo concorrente individuato e descritto nel programma

i processi sequenziali possono essere prodotti in concorrenza da un unico processore o in parallelo da più processori fino ad un numero pari al loro numero.

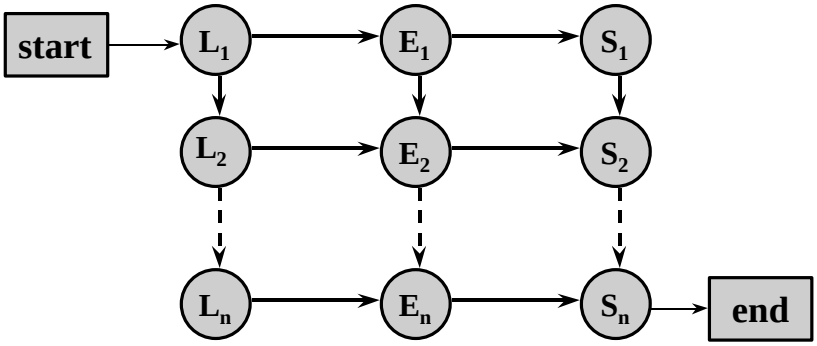
un nuovo esempio




```
var buffer: T; i:1..n
begin
  for i:= 1 to n do
    begin
      lettura (buffer)
      elaborazione (buffer)
      scrittura (buffer)
    end
end
```

un nuovo esempio

ma in realtà dette L_i, E_i e S_i le operazioni da compiere sul record i-simo, per la natura del problema i vincoli di dipendenza sono:



Le interazioni tra processi

Atteso che l'esecuzione di un programma è il frutto di uno o più processi sequenziali, dinamicamente in un sistema sono presenti più processi

.

Le possibili interazioni sono per:

- **competizione**
- **cooperazione**
- **interferenza**

La competizione

nasce dalla presenza in un sistema di risorse comuni ossia utilizzabili da più processi

può essere definita come una interazione prevedibile e non desiderata ma necessaria

richiede che processi che intendano utilizzare risorse comuni si sincronizzino implicitamente

il S.O. si fa carico della sincronizzazione dei processi che intendono utilizzare le risorse comuni che gestisce direttamente

il S.O. rende disponibili i meccanismi per la sincronizzazione implicita dei processi che intendono utilizzare risorse comuni gestite a livello di programma applicativo

La cooperazione

nasce dalla necessità di sincronizzare la dinamica dei processi cooperanti ossia di processi che devono soddisfare dei vincoli di dipendenza

può essere definita come una interazione prevedibile e desiderata

i processi cooperanti possono essere:

- **processi di utente relativi all'esecuzione di un programma concorrente**
- **processi del S.O.**
- **processi di utente e processi del S.O.**

La cooperazione

richiede che processi che intendano cooperare si sincronizzino esplicitamente

i processi cooperanti si scambiano:

- **messaggi di pura sincronizzazione**
- **messaggi di sincronizzazione e dati (comunicazione)**

il S.O. si fa carico della cooperazione tra i processi di utente e i processi del S.O.

il S.O. rende disponibili i meccanismi per la cooperazione tra i processi di utente

L'interferenza

nasce da errori di programmazione concorrente

è dovuta a:

a) **competizione tra processi per uso non autorizzato di risorse comuni**

b) **erronea soluzione di problemi di competizione e di cooperazione**

si manifesta spesso in modo non deterministico in quanto dovuta alla differente velocità di esecuzione dei processi

il S.O. può prevenire le interferenze di tipo a) con meccanismi di controllo degli accessi

il S.O. non è in grado di prevenire le interferenze di tipo b)

Creazione e terminazione di processi

i processi sono creati dinamicamente da altri processi mediante meccanismi resi disponibili dal S.O.

la creazione di un processo avviene a cura di un processo del S.O. (l'interprete di JCL) all'atto del lancio di un programma e a cura di un processo utente durante l'esecuzione di un programma concorrente

la terminazione di un processo avviene mediante meccanismi del S.O. e può essere volontaria o forzata

alla creazione corrisponde l'attribuzione di risorse al processo (e.g. spazio di memoria) mentre alla terminazione è associato il loro recupero

Stati di un processo

Stato puntuale di un processo

Un processo, in quanto sequenza di eventi prodotti dall'esecuzione di un programma, è caratterizzato dal possedere uno stato che è definito dal valore del contatore di programma e dalla n-pla dei valori memorizzati nei registri che contengono le sue istruzioni e i suoi dati.

Concorrono quindi a definire lo stato di un processo i valori delle sue variabili globali, dello stack dei registri del processore accessibili da programma

Stati di un processo

Stato globale di un processo

Definisce la condizione in cui si trova un processo in relazione alla sua possibilità di evolvere, ossia di eseguire istruzioni su di un processore. Può essere uno dei seguenti:

running: le sue istruzioni sono eseguite da un processore

ready: il processo è in attesa che le sue istruzioni vengano eseguite da un processore

waiting: il processo è in attesa di un evento dovendosi sincronizzare, implicitamente o esplicitamente, con un altro processo

Stati di un processo

nello stato running lo stato puntuale di un processo cambia mentre è bloccato negli stati ready e waiting

lo stato ready è anche detto di attesa passiva di un processore e manca se si fa l'ipotesi che il sistema possiede tanti processori quanti sono i processi

oltre che negli stati precedenti, un processo può essere negli stati di:

new: in cui attende che il S.O. gli assegni le risorse necessarie per poterne iniziare l'esecuzione

terminated: in cui attende che il S.O. recuperi tutte le risorse che possiede al termine dell'esecuzione delle sue istruzioni

Transizioni tra gli stati

new ⟶ ready: il S.O. ha assegnato al processo la necessaria memoria centrale e la ha inizializzata, nulla osta all'esecuzione su di un processore della sua prima istruzione

ready ⟶ running: lo scheduler della CPU ha assegnato un processore al processo.

running ⟶ ready: il processo ha perso forzatamente l'uso del processore (e.g. per eccesso di utilizzo continuativo)

running ⟶waiting: il processo ha eseguito una primitiva del S.O.

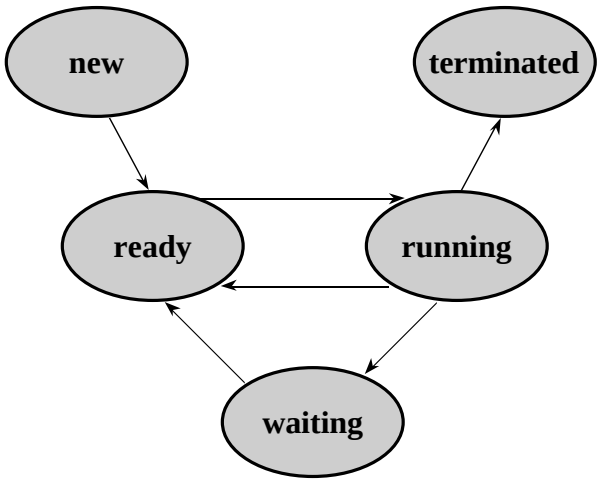
- per sincronizzarsi implicitamente o esplicitamente con altri processi e deve attendere
- per l'uso di risorse gestite dal S.O. e la risorsa non è disponibile o è stata concessa e trattandosi di risorsa di I/O l'operazione è stata lanciata e deve attenderne la terminazione

Transizioni tra gli stati

waiting ⟶ ready: la risorsa gestita dal sistema operativo è disponibile e viene concessa o l'operazione di I/O in precedenza lanciata è terminata

running ⟶ terminated: il processo ha eseguito la primitiva del S.O. di terminazione e attende che le risorse di cui ancora dispone siano recuperate dal S.O.

Stati di un processo e transizioni tra gli stati



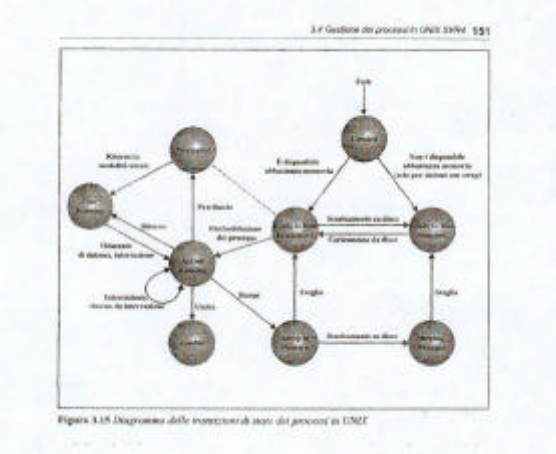


Figura 3.15 *Diagramma delle transizioni di stato dei processi in UNIX*