**Politiche di Ordinamento**

**Astrazione di processo**

Ogni processo è associato a un descrittore chiamato Process Control Block che ne specifica le caratteristiche distintive

* Identificatore del processo
* Contesto di esecuzione del processo

• Tutte le informazioni necessarie a ripristinarne lo stato d’esecuzione dopo una sospensione o un prerilascio

* Stato di avanzamento del processo

• Puntatore (d)alla lista del processi in quello stato

* Priorità

• Iniziale, corrente

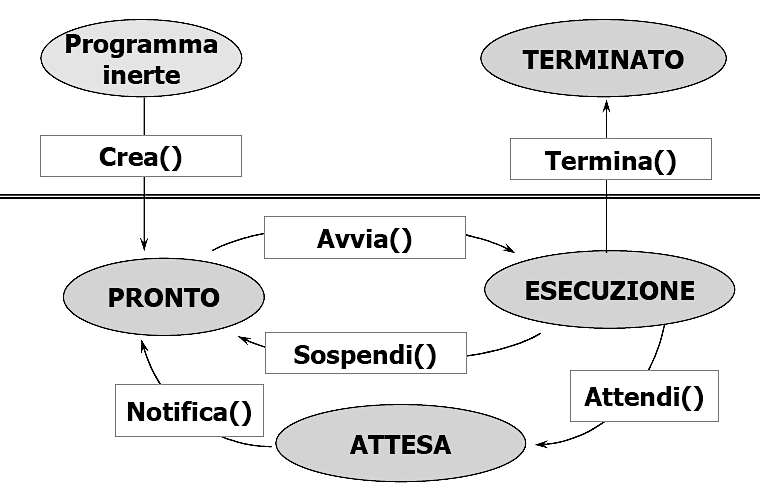
* Diritti di accesso alle risorse e altri eventuali privilegi
* Discendenza familiare

• Puntatore al PCB del processo padre e degli eventuali processi figli

* Puntatore alla lista delle risorse assegnate al processo

• Il PCB relaziona il processo alla sua macchina virtuale

**Ordinamento di processi**

Una decisione di scheduling è necessaria:

* alla creazione di un processo (processo padre o figlio?)
* alla terminazione di un processo (chi lo sostituisce?)
* quando il processo si blocca (in attesa di I/O, o di semaforo, …)

• Importante ad esempio per evitare inversion priority se un processo importante attende che uno meno importante rilasci sezione critica

• Scheduler può non avere info necessarie

* all’occorrenza di un interrupt I/O

• Ad ogni k-esima occorrenza dell’interrupt periodico (50 – 60 Hz)

Diversi metodi per decidere come alternare i processi in esecuzione

* **Scambio cooperativo** (cooperative / non pre-emptive switch)

• Il processo in esecuzione decide da solo quando cedere il controllo ( Windows 3.1 )

* **Scambio a prerilascio** (inconsapevole)

• Il processo in esecuzione viene rimpiazzato

– Da un processo appena arrivato con maggiore importanza (priority-based pre-emptive) » Sistemi a tempo reale

– All’esaurimento del quanto di tempo (time-sharing pre-emptive) » Sistemi interattivi (Unix, Linux, Windows NT)

• Necessita di clock

Il prerilascio si realizza tramite un meccanismo esterno all’esecuzione dei processi

– Un dispositivo (p.es., orologio) solleva una interruzione

– Un gestore software la identifica e, se necessario, la notifica allo scheduler

**Politiche e meccanismi**

Lo **scheduler** è il componente del nucleo che decidel’ordinamento dei processi.

– È progettato prima dei processi che è chiamato a governare

– Sistemi diversi, metriche diverse

• Batch: no preemption

• Interattivi: preemption

• Real time: no preemption ??

Bisogna perciò rendere il suo operato parametrico rispetto a specifici attributi assegnati ai processi

– Per non doverlo cambiare al variare delle applicazioni

• Basta configurare opportunamente gli attributi dei processi

– Lo scheduler attua meccanismi forniti dal nucleo del S/O

Il **dispatcher** è il componente del nucleo che attua le scelte di ordinamento dei processi

– Opera su mandato dello scheduler

– Deve essere molto efficiente perché opera a ogniscambio di contesto (context switch)

• Salva il contesto del processo in uscita, installa quello del processo in entrata e gli affida il controllo della CPU

L’applicazione decide le politiche di ordinamento fissando il valore degli attributi considerati dai meccanismi del nucleo:

– Per determinare l’ordinamento dei processi

– Per influenzare l’attribuzione delle risorse

L’efficienza delle politiche scelte si misura in termini di

– Percentuale di impiego utile della CPU

• Più i processi che il nucleo!

– Il tempo di esecuzione di scheduler e dispatcher è sottratto ai processi

– Numero di processi avviati all’esecuzione per unità di tempo

• Misura di produttività (throughput)

– Durata di permanenza di un processo in stato di pronto

• Tempo di attesa

– Tempo di completamento (turn-around)

– Reattività rispetto alla richiesta di avvio di un processo

• Tempo di risposta

La garanzia di esecuzione dei processi dipende criticamente dalla politica di scambio adottata

– Lo scambio cooperativo non offre alcuna garanzia

– Gli utenti in genere richiedono equità di opportunità (Fairness)

I processi in stato di pronto sono registrati in unastruttura detta lista dei pronti (ready list)

La più semplice gestione della lista è con tecnica a coda(First-Come-First-Served, FCFS)

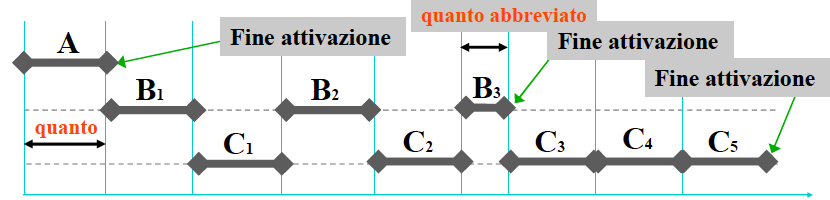
– Il primo processo a entrare in coda sarà anche il primo a essere avviato all’esecuzione

– Molto facile da realizzare e da gestire

Imponendo divisione di tempo (time sharing) sulla politica FCFS si ottiene una tecnica di rotazione detta

round-robin.

Vediamo l’applicazione di un quanto di tempo 2 su tre processi A, B e C con tempo di esecuzione 2, 5, 10 rispettivamente



Le attività di un processo comprendono sequenze di azioni eseguibili dalla CPU intervallate da sequenze di azioni di I/O.

I processi si possono dunque classificare in

– **CPU-bound**

• Comprendenti attività lunga durata sulla CPU

– **I/O-bound**

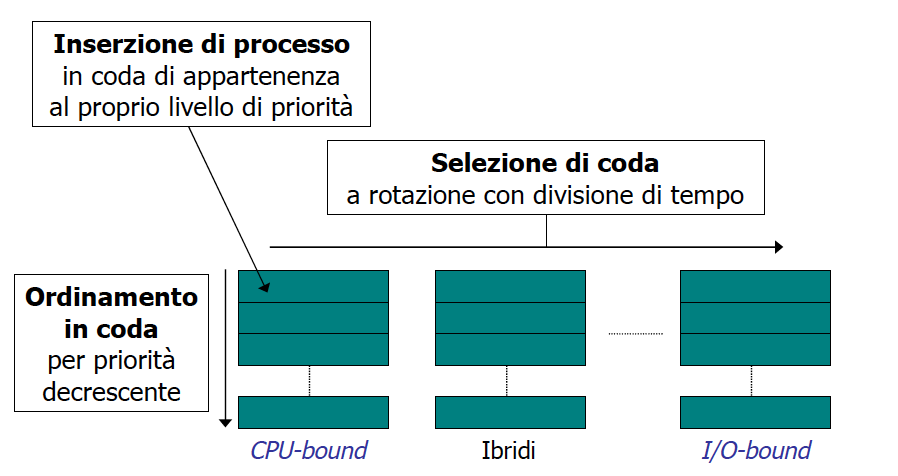
• Comprendenti attività di breve durata sulla CPU intervallate daattività di I/O molto lunghe

La politica FCFS penalizza i processi della classe I/O bound

* Cedendo la CPU durante le attività di I/O sono ritardati al ritorno dai processi che li hanno sostituiti

**Esempio 1: politica di ordinamento a livelli - A rotazione con priorità**

• Scelta di politica – 1 : Assegnare un dato livello di privilegio a ogni singolo processo

****Meccanismo impiegato: attributo rappresentato da una priorità statica o dinamica registrata nel PCB

• Scelta di politica – 2 : Processi distinti sulla base di determinate caratteristiche

* Note a priori, p.es.: CPU-bound, I/O-bound
* Acquisite a tempo d’esecuzione, p.es.: tempo cumulato (di esecuzione o di attesa)

Meccanismo impiegato: rilevazione del valore di un dato campo del PCB

• Scelta di politica – 3 :

* Coda ordinata a priorità per ciascuna classe di processi

Politica a rotazione con priorità

* Selezione di coda round-robin

**Esempio 2: politica di ordinamento a livelli -A priorità con rotazione**

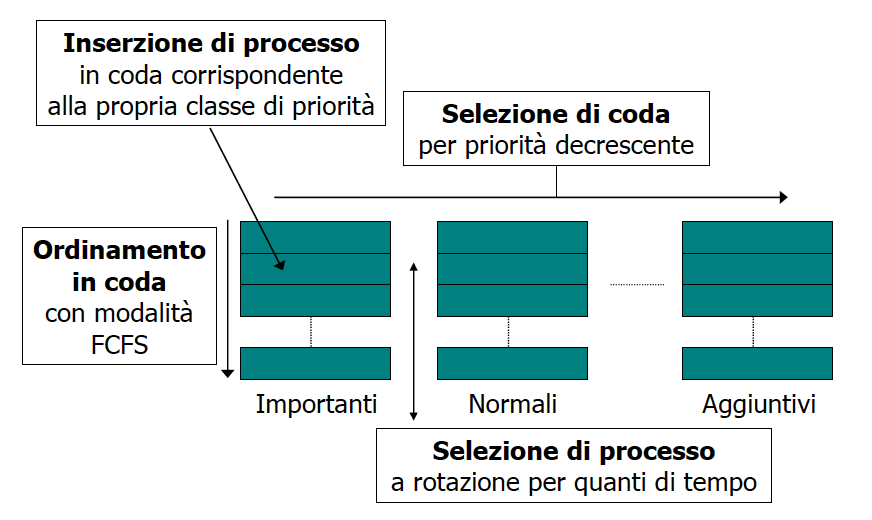
• Scelta di politica – 1 : Assegnare un dato livello di privilegio a ogni singolo processo

Figura Politica a priorità con rotazione

Meccanismo impiegato : attributo rappresentato da una priorità statica o dinamica registrata nel PCB

• Scelta di politica – 2 : processi distinti sulla base di determinate caratteristiche

* Statiche o dinamiche

• Scelta di politica – 3 :

* Coda FCFS per ciascuna classe di processi
* Selezione di coda su base di priorità
* Assegnazione di CPU con modalità round-robin

I meccanismi per realizzare scelte di ordinamento e gestione dei processi risiedono nel nucleo

Le politiche sono determinate fuori dal nucleo:

* Decise nello spazio delle applicazioni
* Decidendo quali valori assegnare ai parametri di configurazione dei processi considerati dai meccanismi di gestione

**Classificazione di sistemi**

Diverse classi di sistemi concorrenti richiedono politichedi ordinamento di processi specifiche

3 classi generali:

* **Sistemi “a lotti”** (batch)

• Ordinamento predeterminato; lavori di lunga durata e limitata

urgenza; prerilascio non necessario

* **Sistemi interattivi**

• Grande varietà di attività; prerilascio essenziale

* **Sistemi in tempo reale**

• Lavori di durata ridotta ma con elevata urgenza; l’ordinamento deve riflettere l’importanza del processo; prerilascio possibile

Caratteristiche desiderabili delle politiche di ordinamento:

–Per tutti i sistemi

• Equità (fairness) – Nella distribuzione delle opportunità di esecuzione

• Coerenza (enforcement) – Nell’applicazione della politica a tutti i processi

• Bilanciamento – Nell’uso di tutte le risorse del sistema

**Obiettivi specifici delle politiche**

Per i sistemi a lotti

• Massimo prodotto per unità di tempo (throughput)

• Massima rapidità di servizio per singolo lavoro (turn-around) – Media statistica

• Massimo utilizzo delle risorse di elaborazione

Per i sistemi interattivi

• Rapidità di risposta per singolo lavoro – Rispetto alla percezione dell’utente

• Soddisfazione delle aspettative generali dell’utente

Per i sistemi in tempo reale

• Rispetto delle scadenze temporali (deadline)

• Predicibilità di comportamento (predictability)

**Politiche di ordinamento**

Per sistemi a lotti

– **FCFS** (First come first served)

• Senza prerilascio, senza priorità

• Ordine di esecuzione = ordine di arrivo

• Massima semplicità, basso utilizzo delle risorse

– **SJF** (Shortest job first)

• Senza prerilascio, richiede conoscenza dei tempi richiesti diesecuzione

• Esegue prima il lavoro (job) più breve

• Non è equo con i lavori non presenti all’inizio

– **SRTN** (Shortest remaining time next)

• Variante di SJF con prerilascio

• Esegue prima il processo più veloce a completare

• Tiene conto di nuovi processi quando essi arrivano

In generale parliamo di lavori quando operiamo senza prerilascio e di processi quando operiamo con prerilascio

Per sistemi interattivi

– **OQ** : Ordinamento a quanti (Round Robin, RR)

• Con prerilascio, senza priorità

• Ogni processo esegue al più per un quanto alla volta

• Lista circolare di processi

– **OQP** : Ordinamento a quanti con priorità

• Quanti diversi per livello di priorità – Come attribuire priorità a processi e come farle eventualmente variare

– **GP** : Con garanzia per processo

• Con prerilascio e con promessa di una data quantità di tempo diesecuzione (p.es. 1/n per n processi concorrenti) – Le necessità di ciascun processo devono essere note (stimate) a priori

• Esegue prima il lavoro maggiormente penalizzato rispetto alla promessa – Verifica periodica o a evento (soddisfacimento della promessa)

– **SG**: Senza garanzia

• Con prerilascio e priorità, opera sul principio della lotteria

– Ogni processo riceve numeri da giocare

– A priorità più alta corrispondono più numeri da giocare

– A ogni scelta per assegnazione di risorsa, essa va al processo possessore del numero estratto

– Le estrazioni avvengono periodicamente (= quanti) e/o a eventi(p.es. attesa di risorse non disponibili)

• Comportamento impredicibile sul breve periodo, ma tende a stabilizzarsi statisticamente nel tempo

– **GU**: Con garanzia per utente

• Come GP ma con garanzia riferita a ciascun utente(possessore di più processi)

Per sistemi in tempo reale

– I sistemi in tempo reale sono sistemi concorrenti nei quali il valore corretto deve essere prodotto entro un tempo fissato

• Oltre tale limite il valore prodotto ha utilità decrescente, nulla o addirittura negativa

– L’ordinamento (scheduling) di processi deve fornire garanzie di completamento adeguate ai processi

• Deve essere analizzabile staticamente (predicibile)

– Il caso peggiore è sempre quando tutti i processi sono pronti insieme per eseguire all’istante iniziale (critical instant)

Per sistemi in tempo reale

– Modello semplice (cyclic executive)

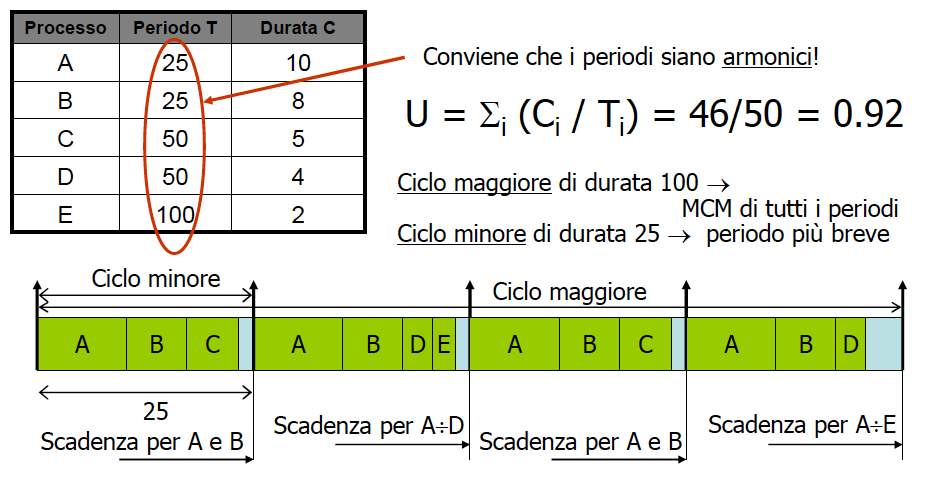
• L’applicazione consiste di un insieme fissato di processi periodici (ripetitivi) ed indipendenti con caratteristiche note

• Ciascun processo è suddiviso in una sequenza ordinata di procedure di durata massima nota

• L’ordinamento è costruito a tavolino come una sequenza di chiamate a procedure di processi fino al loro completamento

• Un ciclo detto maggiore (major cycle) racchiude l’invocazione di tutte le sequenze di tutti i processi

• Il ciclo maggiore è suddiviso in N cicli minori (minor cycle) di durata fissa che racchiude l’invocazione di specifiche sottosequenze

**Esempio 1 - Modello semplice senza suddivisione** 

Per sistemi in tempo reale

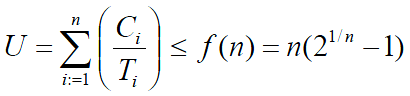
– Ordinamento a priorità fissa

• Preferibilmente con prerilascio (a priorità!) – Processi periodici, indipendenti e noti

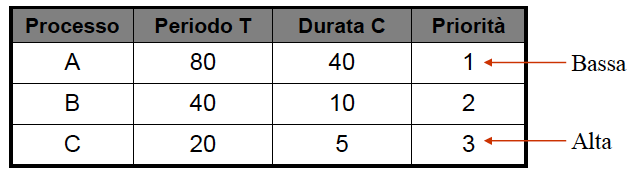
• Assegnazione di priorità secondo il periodo (rate monotonic)

– Per scadenza uguale a periodo (D = T), priorità maggiore per periodo più breve

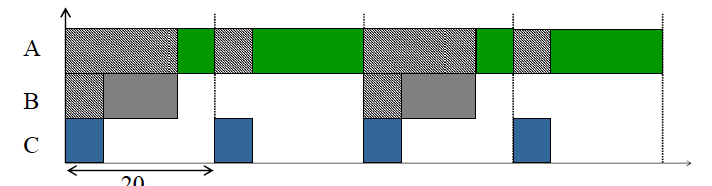
– Test di ammissibilità sufficiente ma non necessario per n processi indipendenti (Liu & Layland, 1973)



**Esempio 2 - Caso semplice ordinamento a priorità**



Il test di ammissibilità fallisce U = 1 > f (3) = 0,78 ma il sistema è ammissibile!



Per sistemi in tempo reale

– Ordinamento a priorità fissa con prerilascio e scadenza inferiore a periodo (D < T)

• Assegnazione di priorità secondo la scadenza

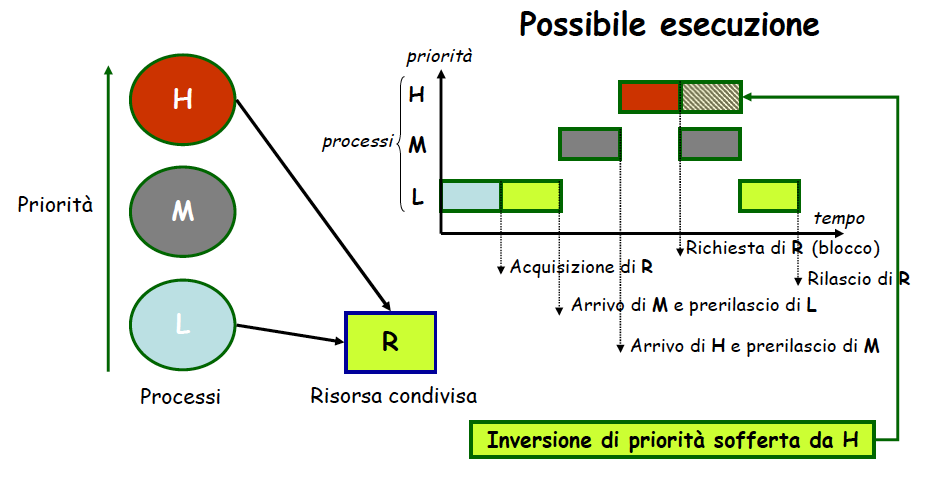
• Rischio di inversione di priorità

– Processi a priorità maggiore bloccati dall’esecuzione di processi a priorità minore

– Effetto causato dall’accesso esclusivo a risorse condivise

– Può condurre a blocco circolare (deadlock)

**Inversione di priorità**



**Un caso reale: Mars Pathfinder**

•Inversione di priorità

* Priorità media attiva mentre quella alta è bloccata dalla bassa

–Risultato: frequenti reset di sistema

• Sistema Operativo VxWorks

– Ordinamento con prerilascio

– Bus informazioni condiviso - Mutex

– Gestione Bus - Alta priorità

– Raccolta dati meteo - Bassa priorità

– Trasmissione - Media priorità

**Inversione di priorità, a parole**

• Esempio IP1

Consideriamo tre processi L, M, H in ordine di priorità crescente

Assumiamo che condividano la risorsa R (Mutex)

Inversione di priorità :

• L si aggiudica R

• H diviene attivo e vuole R

• H deve attendere che L rilasci R – Il tempo d’uso di R ha durata prevedibile

• M diviene attivo e blocca L (diverse priorità)

• H deve attendere che M finisca … – … oltre che L esca dalla sezione critica …

• Cosa accade se nel frattempo si attivano altri processi a prioritàintermedia tra M e H?

Soluzione: Innalzamento delle priorità

– Versione base (Basic Priority Inheritance)

– La BPI non impedisce il deadlock

1. L’innalzamento avviene solo quando un processo a priorità maggiore si blocca all’ingresso di una risorsa attualmente inpossesso di un processo a priorità inferiore

2. Il processo che possiede la risorsa (e che ha avuto l’innalzamento di priorità) può così terminare senza altre interruzioni – L’arrivo di un altro processo di priorità ancora superiore causa prerilascio e riporta la situazione al punto 1

– BPI richiede il controllo di accesso e quindi causa catene di blocchi a tempo d’esecuzione

Studiare l’uso di BPI sull’esempio IP1

Soluzione: Innalzamento della priorità

– Versione avanzata (Immediate ceiling priority)

• Ogni processo j ha una priorità statica di base PBj

• Ogni risorsa condivisa i ha una priorità (ceiling) PCi pari alla massima priorità dei processi che possono richiedere di usarla

• Ogni processo j ha una priorità dinamica Pj = max.PBj, PCi. per ogni risorsa condivisa i in suo possesso

• Un processo può acquisire una risorsa solo se la sua priorità dinamica corrente è maggiore del ceiling di tutte le risorse attualmente in possesso di altri processi

– Un processo a priorità maggiore può essere bloccato una sola volta durante l’intera sua esecuzione (solo per la durata della sezione critica del processo a priorità più bassa)

La tecnica IPC evita il deadlock

• Esempio IP2

* Consideriamo tre processi L, M, H con priorità crescente
* Assumiamo che tutti condividano le risorse R1 e R2 (entrambe Mutex)

• Il priority ceiling di R1 e R2 è superiore alla priorità di H

* L acquisisce R1 e ne assume il ceiling poi si accinge a richiedere R2
* H diventa pronto a questo istante e vorrebbe prerilasciare L

• Ma non può perché la priorità di H non è superiore al ceiling di R1 e R2

* Quindi H resta pronto ma non riesce a prerilasciare L
* L acquisisce anche R2 e poi prosegue fino a rilasciare R1 e R2

• La priorità di L ritorna al valore originale

* H ha ora priorità maggiore di L di ogni altro eventuale M

• H può acquisire R2 proseguire e completare

• La tecnica IPC impedisce il formarsi di catene di blocchi …

* I processi {H}i subiscono al più 1 blocco da parte di 1 processo L in possesso dirisorsa R condivisa con {H}

• Blocco = ritardo nel primo prerilascio

Per sistemi in tempo reale

– Calcolo del tempo di risposta Ri del processo i

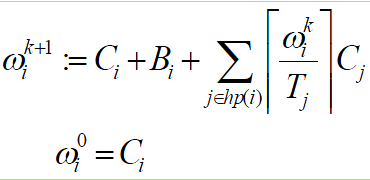
• Tempo di blocco del processo i

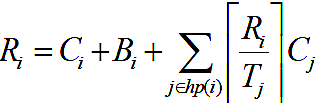
– Bi = maxk{Ck} per ogni risorsa k usata da processi a priorità più bassa di i

• Interferenza subita dal processo i da parte di tutti i processi j a priorità maggiore

– Ii = Ej [Ri / Tj]Cj

• Ri = Ci + Bi + Ii





**Tempo di risposta R**

