Capitolo 6

STALLO

I sistemi multiprogrammati sono caratterizzati da un certo numero di processi sequenziali in esecuzione concorrente[1] i quali, per la loro stessa natura, richiedono l'uso di appropriate risorse di calcolo come *cicli di CPU, spazio di memoria, blocchi su disco, accesso ad unità periferiche*, ecc... Ciascun processo vede il sistema come una collezione di risorse, in numero finito, le quali sono raggruppate in classi, ciascuna delle quali consiste di un numero finito di unità identiche[2] cosicchè se il processo *richiede* un'istanza di un certo tipo di risorsa, l'allocazione di una qualunque istanza di quel tipo soddisferà la richiesta.

Quando un processo chiede di acquisire una risorsa e questa non è attualmente disponibile, allora il processo entra in *stato d'attesa*. Può accadere, in alcuni casi, che il processo in attesa non cambi più stato, in ogni intervallo di tempo successivo, perchè la risorsa richiesta è già stata assegnata a qualche altro processo in attesa. Situazioni di questo genere sono note col termine di *stallo*[3].

6.1 INTRODUZIONE

Poichè uno dei compiti principali che devono essere assolti da un sistema operativo è quello di far condividere fra i vari processi le risorse del sistema di calcolo che gestisce, il fatto che tali risorse possano essere *richieste e mantenute* dai processi, porta alla conclusione che il verificarsi di uno *stallo* diventa un problema potenziale. Il progetto di un sistema operativo non può, dunque, prescindere da questo problema sebbene un evento di questo genere si osserva assai raramente e, quindi, potrebbe essere inteso come un falso problema. Anche se molte situazioni di stallo possono essere, in effetti, prevenute utilizzando procedure di schedulazione "ad hoc", l'analisi delle proprietà dello stallo conduce alla definizione di algoritmi più efficienti per il *riconoscimento* e la *prevenzione* dello stallo.

A tale scopo lo stallo viene caratterizzato da opportune proprietà che possono essere riconosciute in un semplice modello a grafo del sistema di elaborazione. In esso apparirà chiaro come lo *stallo* si presenta come una situazione in cui uno o più processi sono *bloccati* a causa di richieste che non possono mai essere soddisfatte. In questo senso i processi in stallo continueranno a rimanere bloccati fino a quando non viene intrapresa un'azione speciale da qualche forza esterna come il sistema operativo o l'operatore.

Le situazioni di stallo possono essere descritte più precisamente con la seguente definizione.

[1]che può essere anche contemporanea, se sono presenti più CPU
[2]dette *istanze*
[3]o, *deadlock*

Si dice che un insieme di processi è in uno stato di **stallo** *quando ogni processo dell'insieme sta aspettando il verificarsi di un evento che può essere causato soltanto da un altro processo di quell'insieme.*

Gli eventi che interessano in questo caso sono legati, essenzialmente, all'acquisizione e al rilascio di risorse. Si consideri, come esempio, la semplice situazione che può verificarsi in un sistema di elaborazione in cui sono presenti due processi P_1 e P_2 e due risorse R_1 e R_2. Si supponga che le risorse non possano essere rilasciate[4]. Si supponga, inoltre, che R_1 sia stata assegnata a P_1 mentre R_2 risulta assegnata a P_2. Se, ora, P_1 chiede R_2 mentre P_2 chiede R_1, ne risulterà che i processi P_1 e P_2 sono in una situazione di stallo.

6.2 STRATEGIE PER LO STALLO

Poichè il verificarsi dello stallo è un evento indesiderabile, l'analisi e lo studio di questa particolare situazione porta all'approfondimento della logica di interazione dei processi nei sistemi di calcolo. Nello stallo i processi non terminano mai la loro esecuzione e, quindi, le risorse ad essi assegnate non vengono mai rilasciate determinando, in alcuni casi, l'impossibilità per altri processi ad iniziare l'esecuzione. Perciò, ogni sistema di elaborazione, dove più processi possono *condividere risorse* o *scambiare messaggi*, deve realizzare qualche strategia, esplicita o implicita, per una gestione adeguata delle situazioni di stallo. Queste strategie possono essere di tre tipi.

- Prevenzione - Il sistema è realizzato in modo tale che lo stallo sia impossibile, cioè il sistema è *sicuro* rispetto allo stallo. Un sistema che non è sicuro può, in alcuni casi, essere reso sicuro proibendo quelle operazioni che possono portarlo in stallo.

- Riconoscimento - In questa caso una situazione di stallo, sebbene possa verificarsi, é riconoscibile nel momento in cui sta per accadere; però, il sistema è in grado di *ristabilire le condizioni di normale funzionamento* facendo terminare i processi coinvolti nello stallo. Questa strategia consente una migliore utilizzazione delle risorse rispetto al caso precedente, dove lo stallo è assolutamente impedito, e può essere impiegata quando lo stallo non è molto frequente e il ristabilimento delle normali condizioni di funzionamento non è troppo oneroso. Questa strategia, seppure poco impiegata, offre tutti i vantaggi del *fallimento soffice*[5] rispetto a quella descritta nel seguito.

- Eliminazione - La situazione di stallo può verificarsi sempre e non è riconosciuta automaticamente dal sistema. Diventa responsabilità dell'operatore decidere se un tale evento si è verificato e quindi compiere i passi necessari per rimuoverlo.

Come è noto, ogni processo che intende acquisire una risorsa deve, innanzittutto, *richiedere* la risorsa; successivamente la può utilizzare, dopodichè la *rilascerá* rendendola disponibile ad un altro processo. I processi, naturalmente, possono richiedere tutte le risorse di cui abbisognano per portare a termine il loro compito, con l'unico limite che *il numero di risorse richieste non deve mai superare il numero totale di risorse disponibili del sistema.*

In condizioni normali di funzionamento, un processo può utilizzare le risorse facendo riferimento solamente al seguente protocollo.

- Richiesta.- Se la richiesta non può essere immediatamente soddisfatta[6] allora il processo richiedente deve attendere fino a quando la risorsa non gli viene allocata.

[4] dai processi in attesa di una richiesta
[5] soft fail
[6] perchè, ad esempio, la risorsa è già stata assegnata ad un altro processo

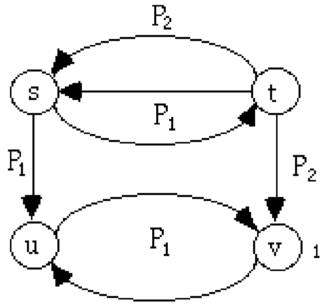

Figure 6.1: esempio di sistema

- Uso.- Il processo pu'o operare sulla risorsa.
- Rilascio.- Il processo rilascia la risorsa.

L'interazione fra processi in un sistema di calcolo può avvenire *esplicitamente*, come nel caso dello scambio messaggi, oppure *implicitamente*, come nella competizione per l'acquisizione di variabili comuni[7]. Il concetto di **risorsa**, dunque, viene utilizzato nel senso più generale del termine e con esso si intende *qualunque oggetto che può provocare l'arresto di un processo.* Ai fini della discussione che segue, è comodo classificare le risorse in *riusabili*, se vengono utilizzate per modellare la *competizione* su oggetti[8] e **consumabili**, quando vengono usate per modellare lo scambio messaggi o di segnali.

6.3 SISTEMI DI ELABORAZIONE

L'approfondimento del problema dello stallo richiede un'analisi più dettagliata di che cosa si debba intendere con i termini *sistema* e *processo.* La nozione di processo è ben nota nell'ambito dei sistemi di elaborazione; è una definizione semplice e precisa che, però, coinvolge dettagli di funzionamento che sono inessenziali per gli scopi dell'analisi che segue. Pertanto, considereremo la seguente definizione.

Si definisce **sistema** la coppia $\Sigma \times \Pi$, dove Σ è un insieme, anche non finito, di *stati* { s, t, u, v, ... } e Π un insieme finito di *processi* $\{ P_1, P_2, P_3, ... \}$. Ciascun processo P_i è definito come una relazione su Σ^2 , e quindi è un suo sottoinsieme $P_i \subset \Sigma^2$.

Come si vede, la definizione di processo proposta, sebbene più generale in quanto considera i processi non deterministici, è anche più semplice perchè non richiede di sapere che cosa farà esattamente il processo in seguito ad una transizione di stato. Ad esempio, si può sapere che il processo P_2 è attualmente in grado di richiedere la risorsa R_1 oppure rilasciare la risorsa R_2, ma non si può stabilire con esattezza quale delle due operazioni il processo effettuerà veramente.

Si consideri, ad esempio, la figura seguente in cui viene illustrato un sistema di 4 stati $\Sigma \equiv \{ s, t, u, v \}$ in cui sono presenti i processi $\{ P_1, P_2 \}$.

Nel grafo i nodi sono etichettati con nomi di stato mentre gli archi hanno come etichette i nomi di processo e rappresentano, quindi, le relative relazioni. In questo caso $P_1(t, u)$ è interpretato come: *quando il sistema si trova nello stato t il processo* P_1 *può cambiare stato portando il sistema dallo stato t ad u.*

[7]che possono rappresentare sia *dispositivi fisici* che *informazioni* per l'accesso ad aree comuni di dati
[8]tipicamente, dispositivi e/o variabili condivise

Si è visto nella definizione che, quando il processo P_i porta lo stato del sistema da $s \in \Sigma$ a $t \in \Sigma$, il cambiamento di stato che indicheremo con

$$P_i(\ s,\ t\ )$$

può essere interpretato come un'**operazione** eseguita dal processo P_i. In generale, comunque, il sistema può evolvere in una sequenza di stati $u_0,\ u_1,\ u_2, ..., u_{n-1},\ u_n$ mediante le operazioni $P_{i_1}(\ u_0,\ u_1\ )$, $P_{i_2}(\ u_1,\ u_2\ )$, ..., $P_{i_n}(\ u_{n_1},\ u_n\ )$. Considerando la chiusura transitiva dell'unione di tutte le relazione che rappresentano i processi del sistema, e che scriveremo

$$P^* \ \equiv\ \{P_1 \cup P_2 \cup P_3 \cup ... \cup P_n\}^*,$$

se il sistema consente la sequenza di operazioni sopra citate, indicheremo con

$$P^*(\ s,\ t\ )\ \equiv\ (\exists i_1)(\exists i_2)...(\exists i_n)(\exists u_1)(\exists u_2)...(\exists u_{n-1})P_{i_1}(\ s,\ u_1\ )\&P_{i_2}(\ u_1,\ u_2\ )\&...\&Pi_n(\ u_{n-},\ t\ )$$

una sequenza arbitraria, ma finita, di una o più operazioni che i processi possono eseguire sul sistema, portandolo dallo stato $sin\Sigma$ al nuovo stato $tin\Sigma$.

Quando il processo P_i non può eseguire alcuna operazione, a partire, dallo stato corrente del sistema, si dice che P_i è *bloccato*; invece, se P_i giungerà comunque in uno stato nel quale non potrà eseguire più alcuna operazione, si dice che P_i è in *stallo*. Formalmente, si hanno le seguenti definizioni.

Si dice che il processo P_i è **bloccato** nello stato $s \in \Sigma$ se non esiste alcuno stato $t \in \Sigma$ tale che $P_i(\ s,\ t\ )$. Si dice che il processo P_i 'e in **stallo** nello stato $s \in \Sigma$ se per tutti gli stati $v \in \Sigma$ per cui $P^*(\ s,\ v\ )$, il processo P_i risulta bloccato in $v \in \Sigma$.

Le precedenti definizioni si possono mettere in forma compatta se si introducono i simboli $\mathrm{B}_i(\ s\ )$ e $\Delta_i(\ s\ )$ a significare che il processo P_i è *bloccato*, rispettivamente, *in stallo* nello stato $s \in \Sigma$. Si ha, allora,

$$\mathrm{B}_i(\ s\ )\ \equiv\ \neg(\exists u)P_i(\ s,\ u\ ) \tag{1}$$

$$\Delta_i(\ s\ )\ \equiv\ (\forall v)[\ P^*(\ s,\ v\ ) \supset \mathrm{B}_i(\ v\ )] \tag{2}$$

Si osservi, a questo proposito, come la definizione (2) possa mettersi in una forma il cui significato rende conto intuitivamente di che cosa si debba intendere per stallo. Si ha, infatti,

$$\begin{aligned}
\Delta_i(\ s\ ) &\equiv (\forall v)[\ P^*(\ s,\ v\ ) \supset \neg(\exists t)P_i(\ v,\ t\ )] \\
&\equiv (\forall v)[\ (\exists t)P_i(\ v,\ t\ )] \supset \neg P^*(\ s,\ v\ )] \\
&\equiv (\forall v)(\forall t)[\ P_i(\ v,\ t\ )] \supset \neg P^*(\ s,\ v\ )] \\
&\equiv (\forall v)(\forall t)\neg[\ P_i(\ v,\ t\ )]\&P^*(\ s,\ v\ )] \\
&\equiv \neg(\exists t)(\exists v)[\ P^*(\ s,\ v\ )]\&P_i(\ v,\ t\ )] \\
&\equiv \neg(\exists t)P^*(\ s,\ t\ )
\end{aligned}$$

dove l'ultima equivalenza si deduce facilmente dalla definizione stessa di P^*. Questa relazione, in effetti, può essere introdotta equivalentemente, con la posizione

$$P^*(\ s,\ t\ )\ \equiv\ (\exists i)(\exists u)P^*(\ s,\ u\ )\&P_i(\ u,\ t\ ) \tag{3}$$

cosicchè la (2) può interpretarsi immediatamente: *il processo P_i è in* stallo *nello stato s se il sistema non è in alcun modo in condizione di compiere qualche sequenza di operazioni che sia in grado di sbloccare P_i*. Dall'osservazione precedente segue immediatamente che un processo *non è in stallo nello stato $s \in \Sigma$ se e solo se esiste qualche stato $t \in \Sigma$ per cui risulta $(\exists u)P^*(\ s,\ u\ )\&P_i(\ u,\ t\ )$*. Se uno o più processi sono in stallo nello stato $s \in \Sigma$, allora si dice che s è uno **stato di stallo**. Se tutti i processi sono in stallo in $s \in \Sigma$, allora si dice che s è uno **stato di stallo totale**. Anche in questo caso è possibile formalizzare le definizioni introducendo i simboli $\Delta(\ s\ )$ e $\Theta(\ s\ )$ a significare che $s \in \Sigma$ è uno stato di stallo *semplice*, rispettivamente, *totale*. Si ha, allora,

$$\Delta(\ s\ )\ \equiv\ (\exists i)\Delta_i(\ s\ ) \tag{4}$$

$$\Theta(\ s\ )\ \equiv\ (\forall i)\Delta_i(\ s\ ) \tag{5}$$

Come appare chiaramente dalle precedenti, uno stato di stallo *totale* è anche uno stato di stallo *normale* ma, in generale, non è vero il viceversa.

6.4 RISORSE DI UN SISTEMA

Finora si è parlato solamente di stati del sistema che sono critici in quanto stati di stallo; è necessario, pertanto, caratterizzare quali sono i normali stati di funzionamento del sistema. A questo proposito, si dice che uno stato è sicuro se, qualunque numero di operazioni vengano eseguito non può mai accadere che questo diventi uno stato di stallo. Si ha, allora, la seguente definizione.

Si dice che uno stato $s \in \Sigma$ è uno stato **sicuro** del sistema se qualunque stato $t \in \Sigma$ si consideri, per il quale sia verificata la relazione $P^*(\ s,\ t\ )$, t non è uno stato di stallo per il sistema.

Anche questa definizione può essere agevolmente formalizzata, introducendo il simbolo $\Psi(\ s\ )$ a significare che $s \in \Sigma$ è uno stato sicuro. Si ha, allora,

$$\Psi(\ s\ )\ \equiv\ (\forall t)[\ P_*(\ s,\ t\ ) \supset \neg\Delta(\ s\ )\ ] \tag{6}$$

Si dice che un sistema é sicuro se contiene uno o più stati sicuri. Il lemma che segue deriva immediatamente dalle defnzioni di stato di stallo e di stato sicuro.

Lemma 1.-*Si supponga che $s \in \Sigma$ e $t \in \Sigma$ siano due stati tali che si abbia $P^*(\ s,\ t\ )$. Se s è uno stato di stallo, allora anche t è uno stato di stallo. Se s è uno stato sicuro, allora anche t è uno stato sicuro.*

Cosi̇̀, lo stallo e la sicurezza sono condizioni *permanenti*. Le defnzioni introdotte finora, comunque, sono molto generali e possono essere applicate in ambiti assai diversi rispetto al contesto dei sistemi operativi. In pratica, qualunque sistema in cui si possono identificare degli stati e delle operazioni su di essi, il cui effetto è una transizione di stato, può essere trattato allo stesso modo. Lo stallo, in questo caso, diventa semplicemente l'impossibilità per il sistema, o di una sua parte, di evolvere.

Consideriamo, invece, l'esempio precedente che caratterizzava l'interazione di due processi P_1 e P_2 che condividono due unità identiche della stessa risorsa. Si supponga che ciascun processo possa richiedere una singola unità di risorsa per volta, e che un processo che ha più unità ne possa rilasciare solamente una alla volta. Ogni processo di questo sistema può trovarsi in uno dei seguenti stati:

- Il processo non ha alcuna risorsa e non ne richiede.
- Il processo non ha alcuna risorsa e ne richiede una unità.
- Il processo ha una unità di risorsa e non ne richiede altre.
- Il processo ha una unità di risorsa e ne richiede un'altra.
- Il processo ha due unità di risorsa e non ne richiede altre.

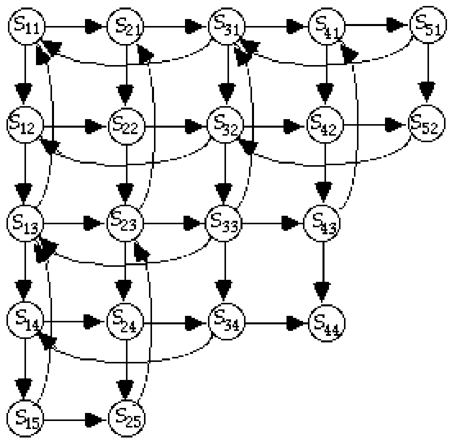

Figure 6.2: sistema con due processi e due risorse

Si è assunto implicitamente che ciascun processo non chiede altre unità, quando ne ha già due, perchè il numero totale di risorse disponibili è esattamente due. Per ciascun processo P_i, sono definite le seguenti transizioni di stato:

- $P_i(\ 1,\ 2\ )$, $P_i(\ 3,\ 4\ )$ in cui il processo P_i chiede un'istanza di risorsa;
- $P_i(\ 2,\ 3\ )$, $P_i(\ 4,\ 5\ )$ in cui al processo P_i viene allocata un'unità di risorsa;
- $P_i(\ 3,\ 1\ )$, $P_i(\ 5,\ 3\ )$ in cui il processo P_i rilascia un'istanza di risorsa.

Se si rappresenta lo stato del sistema con S_{ij} dove i è lo stato del processo P_1 mentre j è l'indice associato allo stato del processo P_2, si ottiene la rappresentazione grafica, sopra riportata, degli stati del sistema, con le relative transizioni, corrispondenti ad operazioni dei processi coinvolti. Come è facile verificare il numero totale degli stati è pari a $5^2 - 5 \equiv 20$, in quanto non sono possibili tutti quegli stati in cui i processi cercano di acquisire risorse in quantità superiore al massimo numero disponibile. Non è difficile rendersi conto che S_{44} è uno stato di stallo in quanto si verifica la situazione per cui ogni processo, avendo già acquisito una risorsa ciascuno, ne chiede un'altra.

Si è già accennato in precedenza che è conveniente dividere le risorse in due classi distinte: *riusabili* e *consumabili*. Entrambe consistono di un numero di unità identiche dello stesso tipo. Quando un processo richiede un'unità di una certa risorsa può accadere che venga bloccato finchè non si rendano disponibili un numero sufficiente di unità di quella risorsa in modo da poterne soddisfare la richiesta; solo in questo caso il processo può acquisire le unità richieste. I processi che non sono in attesa di allocazione di risorse possono, eventualmente, rilasciare unità di altre risorse, incrementando il numero di unità disponibili di quelle risorse.

- Le risorse riusabili hanno le seguenti proprietà. Esiste una fissata quantità di unità[9] complessive per ciascun tipo di risorsa. Ogni istanza è *disponibile*[10] oppure è stata *acquisita*[11] da un particolare processo. Ciascuna istanza può essere assegnata, al più, ad un processo per volta. I

[9] o, istanze
[10] o, non assegnata
[11] o, assegnata

processi possono rilasciare qualunque numero di istanze di risorse già acquisite ma non ancora rilasciate, purchè non siano bloccati in attesa di acquisire ulteriori unità di risorse richieste. Le istanze acquisite dai processi non possono venirgli tolte; quando un processo acquisisce una risorsa, questa non è ulteriormente disponibile finchè il processo non la rilascia.

- Le risorse consumabili hanno le seguenti proprietà. Non c'è un numero fissato di unità complessive per ciascun tipo di risorsa. Ogni unità é sempre disponibile; quando un processo ne acquisisce un'istanza, questa cessa immediatamente di esistere. Solo i processi *produttori* possono rilasciare unità di quella risorsa. In ogni istante, il produttore può rilasciare un numero arbitrario di istanze della risorsa che produce, purchè non sia bloccato in attesa di acquisire altre istanze di risorse di cui aveva fatto richiesta. Ogni unità rilasciata diventa immediatamente disponibile.

La differenza fondamentale fra le risorse *riusabili* e quelle *consumabili* consiste nel fatto che le risorse riusabili non possono essere nè create nè distrutte, ma solamente trasferite da un processo all'altro attraverso un *pool* di unità disponibili. Al contrario, le istanze di risorse consumabili vengono create[12] e, quindi, distrutte[13].

6.5 DEFINIZIONE DI GRAFO

Per sviluppare adeguatamente il modello di sistema di elaborazione, di cui si è accennato nell'introduzione, è necessario considerare alcune definizioni fondamentali della teoria dei grafi.

Si dice **grafo orientato etichettato** la terna $N \times R \times A$ dove N è un insieme non vuoto di nodi, R una relazione su N, ossia un sottoinsieme di $N \times N$, indicato con $R \subset N \times N$, e A un insieme di archi tale che se risulta $R(\ a,\ b\ )$ allora c'è un arco che ha a come primo estremo e b come secondo. In questo caso si dice che l'arco è uscente da a ed entrante in b.

A questo punto, a seconda delle ulteriori condizioni che vengono poste sulla struttura del grafo, si hanno le definizioni indicate di seguito.

- Grafo Bipartito.- Un grafo si dice *bipartito* se l'insieme dei suoi nodi si puó partizionare in due sottoinsiemi disgiunti N_1 e N_2, cioè $N_1 \cup N_2 \equiv N, \quad N_1 \cap N_2 \equiv \emptyset$, tale che ciascun arco ha un nodo in N_1 e l'altro in N_2.

- Nodo Terminale.- Un nodo si dice *terminale* quando non esiste alcun arco uscente da esso.

- Nodo Isolato.- Un nodo si dice *isolato* quando non esistono nè archi uscenti, nè archi entranti in quel nodo. Se il grafo contiene un arco avente il nodo a come primo estremo e il nodo b come secondo estremo, a si dice anche *nodo padre* mentre b prende il nome di it nodo figlio.

- Cammino.- Si dice *cammino* di estremi a e b, una sequenza di nodi $\{u_0, u_1, u_2, ..., u_{n-1}, u_n\}$ tale che $(\ u_i,\ u_{i+1}\ )$ sono archi del grafo ed, inoltre, $u_0 \equiv a$ e $u_n \equiv b$.

- Ciclo.- Si dice *ciclo* un cammino per il quale il secondo estremo coincide con il primo.

- Insieme Raggiungibile.- Si dice *insieme raggiungibile* di un nodo $a \in N$, l'insieme di tutti i nodi $b \in N$ tali che esiste un cammino che ha b come secondo estremo ed a come primo.

- Annodamento.- Si dice *annodamento* un insieme non vuoto X di nodi del grafo tale che l'insieme dei nodi raggiungibili da ciascun nodo di X è *esattamente* l'insieme X.

---

[12] o, prodotte

[13] o, consumate

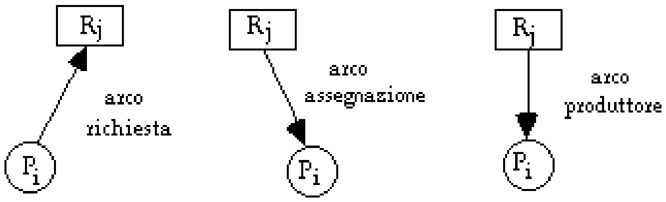

Figure 6.3: richiesta e acquisizione di risorse

Dalle definizioni precedenti segue che in un grafo bipartito con non più di un arco uscente da un nodo a e diretto verso un nodo b, ci sono al più $2 \times m \times n$ archi in A, avendo indicato con n e m gli elementi della ripartizione dell'insieme dei nodi N del grafo.

6.6 GRAFO DI ALLOCAZIONE DELLE RISORSE

Siamo ora in grado di introdurre un modello formale orientato alla descrizione astratta dell'interazione fra processi. Ogni stato del sistema è rappresentato da un grafo etichettato orientato i cui nodi corrispondono a processi o risorse. Le interazioni fra i processi sono descritte da archi uscenti da nodi processo e diretti su nodi risorsa, o viceversa. Possiamo, quindi, dare le seguente definizione.

Si dice **sistema di allocazione delle risorse** una struttura caratterizzata da un insieme non vuoto di processi $\Pi \equiv \{ P_1, P_2, ..., P_n \}$, un insieme non vuoto di risorse $\mathrm{R} \equiv \{ R_1, R_2, ..., R_m \}$, una partizione di R in due sottoinsiemi disgiunti R_u e R_c delle risorse riusabili, rispettivamente, consumabili. Inoltre, per ciascuna risorsa riusabile R_j è associato un intero positivo T_j, per indicare il numero *totale di istanze* della risorsa R_j mentre per ogni risorsa consumabile è associato un sottoinsieme di processi produttori.

L'insieme degli stati Σ di un sistema di allocazione delle risorse è l'insieme di tutti i grafi costruibili a partire dai nodi precedentemente introdotti, secondo la definizione seguente.

Si dice **grafo di allocazione delle risorse** un grafo etichettato orientato bipartito i cui insiemi disgiunti di nodi sono $\Pi \equiv \{ P_1, P_2, ..., P_n \}$ e $\mathrm{R} \equiv \{ R_1, R_2, ..., R_m \}$, al quale si associa il vettore delle istanze disponibili, i cui elementi $\{ w_1, w_2, ..., w_m \}$ sono tutti interi non negativi. Gli archi uscenti dai nodi processo si dicono archi richiesta, mentre quelli uscenti dai nodi risorsa si dicono archi assegnazione se la risorsa è riusabile, archi produttore se la risorsa è consumabile.

La figura seguente illustra la situazione di un generico grafo di allocazione delle risorse in cui sono evidenziati i tre possibili tipi di archi.

Si noti che la distinzione fra archi assegnazione e archi produttore è fatta semplicemente tenendo conto del fatto che le risorse possono essere riusabili, rispettivamente, consumabili. Nel modello proposto, dunque, un *processo che produce una risorsa consumabile* è del tutto equivalente ad un *processo che acquisisce una risorsa.*

Non è difficile convincersi, inoltre, che ogni grafo di allocazione delle risorse deve godere delle seguenti proprietà. Per ogni nodo R_j che rappresenta una *risorsa riusabile*:

- **a.** - il numero totale di archi assegnazione uscenti da R_j non può superare il numero totale T_j di unità disponibili;

- **b.** - il numero di unità disponibili w_j sommato con il numero di archi uscenti da R_j deve essere sempre uguale al numero T_j di unità totali;

- **c.** - per ciascun nodo processo P_i, il numero di archi richiesta ($P_i,\ R_j$) sommato con il numero di archi assegnazione ($R_j,\ P_i$) non può mai superare il numero T_j di unità totali.

Inoltre, per ogni nodo R_j che individua una *risorsa consumabile*:

- **d.** - esiste un arco produttore uscente da R_j e diretto verso il processo P_i se e solo se P_i è uno dei produttori di R_j;

- **e.** - il numero w_j di unità disponibili è un valore intero non negativo.

La proprietà (c) implica che i processi non possono richiedere un numero complessivo di istanze di una certa risorsa riusabile che superi il numero complessive di unità di quella risorsa. La proprietà (e) implica che i sistemi di allocazione delle risorse, il cui modello comprenda anche risorse consumabili, sono caratterizzati da un insieme infinito numerabile di stati.

Si è visto in precedenza che i processi, nei sistemi di allocazione delle risorse, sono modellati da relazioni sugli stati del sistema. Specificheremo, ora, queste relazioni descrivendo le operazioni che possono essere effettuate dai processi. Si noti, innanzittutto, che i processi possono eseguire tre tipi di operazioni: *richieste*, *acquisizioni* e *rilasci* di risorse secondo le condizioni precedentemente stabilite. Quindi, il grafo di allocazione delle risorse[14] può essere modificato da un'operazione di richiesta, che *aumenta il numero di archi di richiesta*, da un'operazione di assegnazione, che diminuisce contemporaneamente il numero di archi richiesta e il numero di unità disponibili e da un'operazione di rilascio che aumenta il numero di unità disponibili. Pertanto, se $s,\ t \in \Sigma$[15] sono due stati del sistema rappresentabili come grafi di allocazione delle risorse, definiamo le operazioni su di essi come segue.

- Richiesta.- Se nello stato $s \in \Sigma$ non sono presenti archi richiesta uscenti dal nodo P_i, allora la relazione $P_i(\ s,\ t\ )$ rappresenta una transizione di stato verso $t \in \Sigma$ dove t differisce da s solamente per uno o più archi richiesta uscenti da P_i.

- Acquisizione.- Se nello stato $s \in \Sigma$ sono presenti archi richiesta uscenti da P_i, e per ciascuna risorsa R_j, il numero w_j di istanze diponibili è maggiore del numero di archi richiesta ($P_i,\ R_j$), allora la relazione $P_i(\ s,\ t\ )$ rappresenta una transizione di stato dove $t \in \Sigma$ è identico a s tranne che per ogni arco richiesta ($P_i,\ R_j$) in s il numero di istanze disponbili w_j è decrementato di una unità. Inoltre, se R_j è una risorsa riusabile, ciascun arco richiesta ($P_i,\ R_j$) è sostituito da un arco assegnazione ($R_j,\ P_i$), mentre se R_j è una risorsa consumabile l'arco richiesta ($P_i,\ R_j$) viene cancellato.

- Rilascio.- Se nello stato $s \in \Sigma$ non sono presenti archi richiesta uscenti dal nodo P_i e c'è qualche arco[16] diretto verso P_i, allora la relazione $P_i(\ s,\ t\ )$ rappresenta una transizione di stato dove $t \in \Sigma$ differisce da s per avere incrementato il numero w_j di unità disponibili di una risorsa R_j che ha, in s, qualche arco[17] ($R_j,\ Pi$). Inoltre, se R_j è una risorsa riusabile, il numero di archi assegnazione cancellati è pari all'incremento del valore w_j di istanze disponibili.

Dalle definizioni precedenti segue che un processo P_i è bloccato, in un certo grafo s di allocazione delle risorse, se e solo se esiste un nodo risorsa R_j tale che il numero di archi richiesta ($P_i,\ R_j$) eccede w_j, cioè quando P_i richiede un numero di istanze della risorsa R_j maggiore di quelle attualmente disponibili.

[14] che rappresenta lo stato del sistema
[15] si noti che, ora, Σ rappresenta un insieme di grafi
[16] assegnazione o produttore
[17] assegnazione, o produttore

6.7 CONDIZIONI PER LO STALLO

Le nozioni di *sistema* e di *stallo* sono state introdotte in modo del tutto generale, mentre nella sezione precedente si è dato un esempio di sistema di allocazione delle risorse come possibile modello di interazione fra processi nell'ambito dei sistemi operativi. In questa sezione verranno utilizzate queste definizioni per sviluppare condizioni necessarie e sufficienti per lo stallo nei sistemi di allocazione delle risorse.

Si è visto che un processo è in stallo quando non c'è alcun modo affinchè il processo non si blocchi. Pertanto, se siamo in grado di determinare una sequenza di operazioni che non blocchi tale processo, allora abbiamo verificato che il processo non è in stallo. A tale scopo verrà introdotto il *metodo di riduzione* per i grafi di allocazione delle risorse. La riduzione, riferita ad un particolare processo P_i, corrisponde alla *migliore sequenza di operazioni* che P_i può eseguire per favorire che altri processi non si blocchino. Tale metodo si basa sul *forzare il rilascio del maggior numero possibile di risorse* da parte di P_i.

Affinchè il metodo di riduzione si possa applicare nei casi più generali possibili, è necessario introdurre il simbolo speciale Ω a significare un intero positivo di valore arbitrariamente grande. A questo proposito vale la seguente definizione.

Con il simbolo Ω si intende un intero positivo tale che

- $(\forall k \in N)[\ \Omega > k\ ]$
- $(\forall k \in N)[\ \Omega + k \equiv k + \Omega \equiv \Omega\ ]$

ossia, un intero che segue ogni altro intero positivo e che è idempotente rispetto all'operazione di somma.

L'introduzione di un tale simbolo si è resa necessaria per permettere che durante la riduzione del grafo, il numero w_j di unità disponibili per una risorsa consumabile possa assumere un valore arbitrariamente grande. Questa situazione può essere interpretata dicendo che *i produttori di una risorsa consumabile hanno rilasciato un numero sufficiente di istanze della risorsa da soddisfare ogni successiva richiesta.*

Nella definizione di grafo di allocazione delle risorse si è visto che ogni risorsa consumabile R_j deve essere associata ad un sottoinsieme non vuoto di produttori. Con la convenzione precedente, invece, si potrà supporre che R_j possa non avere produttori nell'ipotesi che si abbia $w_j \equiv \Omega$. Si può, allora, dare la seguente definizione.

Ogni processo P_i che non è nè un nodo isolato nè un nodo bloccato, può **ridurre** il grafo di allocazione delle risorse cancellando tutti gli archi $(\ P_i,\ R_j\ )$ e $(\ R_j,\ P_i\ )$ per ogni nodo risorsa R_j connesso con P_i. Inoltre, se R_j è una risora riusabile, ogni arco assegnazione $(\ R_j,\ P_i\ )$ cancellato, determina l'incremento del corrispondente w_j di una unità. Infine, se R_j è una risorsa consumabile, il valore di w_j è decrementato per ogni arco richiesta $(\ P_i,\ R_j\ )$ cancellato oppure diventa Ω se P_i è un produttore di R_j.

Si dice che il grafo è **completamente riducibile** se esiste una sequenza di riduzioni che cancella tutti gli archi del grafo.

Con la convenzione che le risorse consumabili R_j possano non avere produttori se $w_j \equiv \Omega$, allora ogni riduzione trasforma un grafo di allocazione delle risorse in un altro grafo di allocazione delle risorse. Il teorema che segue dà le condizioni necessarie e sufficienti per lo stallo.

Teorema 1.-*Si dice che il processo P_i* **non è in stallo** *relativamente al grafo di allocazione delle risorse $s \in \Sigma$ se e solo se esiste una sequenza di riduzioni applicate ad s che trasforma s in un nuovo grafo $t \in \Sigma$ in cui P_i non è bloccato.*

Dim. - Si supponga, dapprima, che P_i non è in stallo in $s \in \Sigma$. Questo significa che deve esistere una sequenza di processi P^*, le cui operazioni $P^*(s, t)$ portano s in $t \in \Sigma$ ed, inoltre, P_i non è bloccato in t. Si può, allora, modificare P^*

- cancellando ogni processo in P^* che è un nodo isolato di $s \in \Sigma$;
- omettendo, per ciascun processo P_i che appare in P^*, tutte le occorrenze tranne la prima.

E' facile verificare, a questo punto, che qualunque sequenza di operazioni effettuate da un certo insieme di processi, farà uso, al più, di tante unità disponibili di risorse quanto sono quelle che si avrebbero con una sequenza di riduzioni per i processi dello stesso insieme. Pertanto, ogni riduzione[18] nella sequenza modificata produrrà almeno tante istanze di risorse di quelle nella corrispondente operazione per la sequenza non modificata. Questo significa che ogni riduzione, relativa ad un certo processo nella sequenza modificata, metterà a disposizione abbastanza risorse da permettere al processo successivo[19] di non rimanere bloccato e, quindi, di potersi ridurre. In conclusione, se il processo P_i non è in stallo in $s \in \Sigma$, allora il grafo s può essere ridotto in un nuovo grafo $t \in \Sigma$ in cui P_i non è bloccato.

Si supponga, ora, che P^* sia una sequenza di riduzioni che, applicata, allo stato $s \in \Sigma$, determina un nuovo stato $u \in \Sigma$, in cui P_i non è bloccato, ossia $(\exists u)P^*(s, u)\&\neg \mathrm{B}_i(u)$. Si osservi che la riduzione per ciascun processo P_j può essere realizzata con un'acquisizione e un rilascio di risorse per cui la sequenza di operazioni può essere *simulata* da una sequenza di operazioni che portano lo stato $s \in \Sigma$ nel nuovo stato $t \in \Sigma$ dove P_i non è bloccato. Si può, allora, concludere che P_i non è in stallo nello stato s.

Questa dimostrazione permette di affermare che, se esiste una sequenza di trasformazioni per cui P_i non è bloccato, allora tale sequenza può essere determinata usando la *riduzione su grafi*. Dal teorema precedente segue immediatamente il seguente

Corollario 1.-*Se un grafo di allocazione delle risorse è* **completamente riducibile***, allora non è uno stato di stallo.*

Comunque, nè il *Teorema 1*, nè il relativo *Corollario* sono in grado di suggerire qualche metodo rapido ed efficiente per verificare se un certo grafo di allocazione delle risorse rappresenta uno stato di stallo. Questo è dovuto al fatto che tale verifica deve essere condotta in modo pressocchè esaustivo, comportando l'analisi di $n!$[20] possibili sequenze di riduzione. D'altra parte, poichè la riduzione applicata a grafi con risorse consumabili, può comportare la riduzione del numero delle istanze disponibili, l'ordine di riduzione diventa importante.

Consideriamo ora una famiglia importante di grafi di allocazione delle risorse, per i quali è possibile individuare una semplice condizione sufficiente per lo stallo. A questo proposito si ha la seguente definizione.

Si dice **espediente** uno stato $s \in \Sigma$ in cui tutti i processi che hanno richieste in corso sono bloccati. Qualunque stato che non è espediente può diventarlo se tutte le possibili richieste sono garantite.

Molti sistemi di elaborazione impiegano una strategia di allocazione delel risorse *espediente*, in cui tutte le richieste sono immediatamente garantite. In tali sistemi le istanze di risorse vengono assegnate solo immediatamente dopo una richiesta o un rilascio, per cui il sistema si trova sempre in uno stato espediente[21].

In questo caso la situazione generale si semplifica abbastanza da rendere valido il seguente

[18] relativa ad un certo processo P_i
[19] in detta sequenza
[20] essendo n il numero dei processi
[21] tranne per il breve intervallo prima dell'assegnamento della risorsa

Teorema 2.- *Condizione necessaria per lo stallo, in un grafo di allocazione delle risorse, è la presenza in esso di un* **ciclo**. *Inoltre, se il grafo è* **espediente**, *allora la presenza di un* **annodamento** *è una condizione sufficiente.*

Dim.- Se il grafo non contiene cicli si può costruire un ordinamento lineare dei processi con la seguente proprietà. Il processo P_i precede P_j nell'ordinamento se esiste un cammino nel grafo che origina in P_i e termina in P_j. Si può allora verificare che se si cancellano da questo ordinamento i processi che hanno nodi isolati, si ottiene una sequenza di riduzioni che riducono completamente il grafo. Poichè il grafo è completamente riducibile segue, anche, che non è uno stato di stallo.

Se un grafo espediente contiene un annodamento, allora tutti i processi dell'annodamento sono tutti bloccati in attesa di acquisire istanze di risorse facenti parte dell'annodamento. Ma queste risorse possono vedere incrementato il numero w_j di unità disponibili solamente dai processi[22] nell'annodamento. Quindi, tutti i processi nell'annodamento sono in stallo e lo stato, rappresentato dal grafo di allocazione delle risorse, è uno stato di stallo.

Se tutte le possibili richieste non sono state garantite, cioè lo stato non è *espediente*, il grafo di allocazione delle risorse può contenere un annodamento ma non, per questo, essere uno stato di stallo. Utilizzando le proprietà degli annodamenti nei grafi è possibile provare il seguente

Corollario 2.- *Sia $s \in \Sigma$ un grafo di allocazione delle risorse* **espediente**. *Se il processo P_i non è un nodo terminale e non esiste alcun cammino uscente da P_i verso un nodo terminale, allora il processo P_i non è in stallo.*

6.8 RICHIESTE DI UNA SOLA ISTANZA

Per alcuni casi speciali di grafi di allocazione delle risorse è possibile individuare semplici condizioni necessarie e sufficienti per lo stallo e la sicurezza dallo stallo. Iniziamo, imponendo la seguente restrizione.

Si dice che un sistema di allocazione delle risorse modella **richieste per istanze singole** se un processo può richiedere solamente una unità di risorsa per volta.

In questo caso il grafo di allocazione delle risorse presenta al più un arco richiesta uscente da ogni nodo processo. Tale restrizione ha il vantaggio pratico di semplificare gli algoritmi usati per realizzare le operazioni di richiesta e di acquisizione.

Teorema 3.- *Un grafo di allocazione delle risorse espediente con richiesta di singole istanze di risorse è uno stato di* **stallo** *se e solo se contiene un* **annodamento**.

Dim.- Poichè nel teorema precedente si afferma che un annodamento è una condizione sufficiente per lo stallo, basta verificare chein presenza di richieste di risorse per unità singole, l'annodamento è anche una condizione necessaria per lo stallo. Se il grafo non contiene un annodamento, allora da ciascun nodo che è un processo bloccato, esiste un cammino diretto verso un nodo terminale, il quale deve essere necessariamente un nodo processo. D'altra parte, ogni tale processo bloccato non può essere in stallo perchè il grafo può venire successivamente ridotto da ciascun processo che appare nel cammino, iniziando dal nodo terminale e muovendosi all'indietro. Quindi, quando non c'è un annodamento, nessun processo è in stallo e lo stato non è uno stato di stallo.

I seguenti risultati sono strettamente correlati al *Teorema 3*. Sia $s \in \Sigma$ un grafo di allocazione delle risorse espediente con richiesta di istanze singole. Valgono, allora, le seguenti proprietà.

- Il grafo $s \in \Sigma$ non è uno stato di stallo se e solo se s è *completamente riducibile*.

[22] bloccati

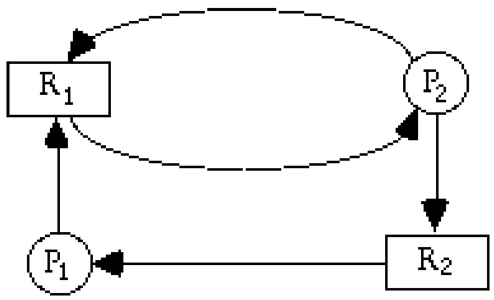

Figure 6.4: grafo di allocazione delle risorse

- Si supponga che esistono diverse sequenze di riduzioni che applicate ad s, trasformino quest'ultimo in un grafo che non può essere ulteriormente ridotto. Allora, tutti questi stati risultanti sono tra di loro *equivalenti*.
- Il processo P_i non è in stallo nello stato $s \in \Sigma$ se e solo se il nodo P_i è un nodo terminale oppure ha un cammino diretto verso un nodo terminale.

Il fatto di permettere solamente richieste di istanza singola gioca un ruolo essenziale nel *Teorema 3*. Si consideri, come esempio, un sistema avente due risorse consumabili R_1 e R_2 i cui unici produttori sono, rispettivamente, i processi P_1 e P_2.

Si supponga che non sia pendente alcuna richiesta e, contemporaneamente, non sia disponibile nessuna istanza di risorse. Si assuma, ora, che ci sia da parte del processo P_1 una doppia richiesta di una unità per la risorsa R_1 e un'altra unità per la risorsa R_2. E' presto visto che P_1 si trova in uno stato di stallo anche se il grafo di allocazione delle risorse non contiene alcun annodamento.

Il *Teorema 3* e i relativi corollari sono molto importanti perchè danno la possibilità di determinare algoritmi efficienti per il rilevamento dello stallo.

- Algoritmo 1.- La dimostrazione del *Teorema 3* può essere presa come punto di partenza per realizzare un algoritmo che **riconosca** se un grafo di allocazione delle risorse è uno stato di stallo, verificando se il grafo contiene un *annodamento*. Il funzionamento dell'algoritmo si basa sulla trasformazione in nodi terminali di tutti i nodi padre di nodi terminali. Il grafo non contiene annodamenti se e solo se tutti i nodi diventano nodi terminali.
- Algoritmo 2.- L'ultimo dei tre corollari del *Teorema 3* può essere utilizzato per ottenere un algoritmo che **determina** se un particolare processo bloccato è in stallo. Il suo funzionamento si basa sull'analisi sistematica di tutti i cammini uscenti dal nodo processo. Il processo non è in stallo se e solo se esiste qualche cammino che porta verso un nodo terminale.

Il massimo tempo di esecuzione di tali algoritmi è proporzionale al numero di archi nel grafo.

Se il numero degli archi uscenti da un certo nodo è maggiore di uno, allora questi archi possono essere rappresentati da un singolo arco etichettato da un intero positivo che dà il numero di archi presenti. Chiameremo questa rappresentazione alternativa *grafo di allocazione delel risorse* con **archi pesati**. Dunque, se il grafo impiega archi pesati, il numero degli archi è al più $2 \times m \times n$, per cui il massimo tempo di esecuzione degli algoritmi precedenti è proporzionale a $m \times n$[23]. Tempi di esecuzione così veloci permettono un loro impiego pratico nei sistemi di elaborazione.

[23] con n numero di processi ed m numero di risorse

In alcuni sistemi può essere desiderabile la verifica *continua* dello stato di stallo, cioè dopo ogni operazione che può provocare lo stallo. Si può verificare facilmente che i sistemi nei quali è consentito effettuare solamente richieste di istanze singole di risorse e le unità disponibili vengono immediatamente acquisite, allora l'unica operazione che può causare lo stallo è la *richiesta di unità non attualmente disponibile*. Appare ovvio che lo stallo coinvolgerà necessariamente il processo che effettua la richiesta. Pertanto, il rilevamento continuo dello stallo sarà effettuato applicando l'*Algoritmo 2* ad ogni processo che richiede una istanza non disponibile di qualche risorsa.

Il sovraccarico di sistema prodotto dall'algoritmo di verifica continua dello stallo può essere notevolmente ridotto verificando lo stallo **occasionalmente**, ad esempio, ad intervalli di tempo prefissati sufficientemente lunghi, oppure quando qualche processo rimane bloccato per un tempo ritenuto troppo lungo. Una verifica di questo tipo può essere realizzata dall'*Algoritmo 1*.

6.9 SISTEMI DI RISORSE CONSUMABILI

Un altro caso speciale di grafo di allocazione delle risorse è quello in cui compaiono solamente risorse consumabili. Si ha, pertanto, la seguente definizione.

Si dice sistema di allocazione di risorse **consumabili** un sistema il cui sottoinsieme di risorse riusabili è vuoto.

Secondo quanto si è già osservato in precedenza, tutte le comunicazioni fra processi in un sistema di questo tipo sono *esplicite* per cui possiamo supporre che i processi interagiscano secondo la relazione *produttore/consumatore*. Sotto queste condizioni è possibile associare a ciascuna risorsa consumabile R_j l'insieme dei processi **consumatori** di R_j. Supporremo, nel seguito, che ciascun processo sia un produttore o un consumatore di almeno una risorsa.

A questo punto conviene introdurre un'ulteriore restrizione per permettere una semplice caratterizzazione degli stati sicuri.

Assegnato un sistema di allocazione di risorse consumabili, si dice che questo rappresenta un **grafo di richieste limitate** se, per ogni processo P_i che non è un consumatore della risorsa R_j non compaiono in esso archi richiesta ($P_i,\ R_j$).

Naturalmente, è possibile caratterizzare un grafo di allocazione delle risorse in modo che rappresenti un sistema di richieste limitate e determinare se questo definisce uno stato sicuro rispetto allo stallo. Per un dato sistema di allocazione a richieste limitate, il *grafo di richieste limitate* rappresenta quello stato del sistema che non ha alcuna risorsa disponibile ed è presente in esso un arco richiesta ($P_i,\ R_j$) se e solo se P_i è un consumatore di R_j.

Per quanto riguarda gli archi produttore di un grafo di richieste limitate, questi sono già stati descritti al punto **d** di una sezione precedente. Inoltre, tale grafo caratterizza completamente un sistema di risorse consumabili a richieste limitate perchè, a partire da esso, si possono determinare[24], per ciascuna risorsa R_j, gli insiemi dei processi produttori e dei processi consumatori associati a quella risorsa. Nella figura che segue è riportato un grafo di richieste limitate in cui sono presenti due processi P_1 e P_2 e due risorse R_1 e R_2. .sv 10 Relativamente alla prima risorsa i produttori e i consumatori sono, rispettivamente, $\{P_1,\ P_2\}$ e $\{P_2\}$, mentre per la seconda risorsa sono $\{P_1\}$ e $\{P2\}$, rispettivamente.

Il seguente teorema permette di determinare se è possibile che si verifichi lo stallo in un sistema di risorse consumabili a richieste limitate.

Teorema 4.- *Un sistema di risorse consumabili a richieste limitate è* **sicuro** *se e solo se il relativo grafo è completamente riducibile.*

[24]oltrechè, naturalmente, i processi e le risorse

Dim.- Sia $v \in \Sigma$ un grafo di richieste limitate. Innanzittutto, bisogna far vedere che qualunque sequenza di riduzioni applicate a v produce un unico grafo che non può essere ulteriormente ridotto. si può, allora, verificare che v[25] non rappresenta uno stato di stallo se e solo se è completamente riducibile. Il completamento della dimostrazione si ottiene assumendo che v non sia completamente riducibile[26] e mostrando che per qualunque stato sicuro s, di cui si ipotizza l'esistenza, deve essere $P^*(s, v)$. Supponendo, allora, che la sequenza di riduzioni riduce completamente v, basta mostrare che ogni sequenza similare riduce completamente qualunque grafo che rappresenta uno stato del sistema.

Nel grafo a richieste limitate illustrato nella figura precedente, la sequenza P_1P_2 lo riduce completamente, cosicchè il sistema è sicuro, cioè nè P_1 nè P_2 sono in stallo. Questo teorema potrebbe venire utilizzato, in linea di principio, per determinare se un semplice sistema di elaborazione si trova in stato di stallo. In pratica, però, la condizione di sicurezza[27] è troppo forte e potrebbe non essere verificata. Questo significa che i processi nei sistemi normalmente in uso sono in grado di evitare lo stallo regolando opportunamente l'uso delle risorse secondo una qualche conoscenza dello stato del sistema. La teoria proposta, comunque, non fà alcun riferimento a tale comportamento "intelligente" dei processi.

6.10 SISTEMI DI RISORSE RIUSABILI

Considereremo ora un altro caso speciale di grafo di allocazione delle risorse, nel quale sono presenti solamente risorse riusabili. Diamo, a questo proposito, la seguente definizione.

Si dice sistema di allocazione di risorse **riusabili** un sistema il cui sottoinsieme delle risorse consumabili è nullo.

Sotto queste condizioni è possibile enunciare e dimostrare il seguente

Teorema 5.- *Sia $s \in \Sigma$ un qualunque stato di un sistema di risorse riusabili. Allora si può affermare che:*

- *lo stato $s \in \Sigma$ non è uno stato di stallo se e solo se il corrispondente grafo non è completamente riducibile;*
- *tutti gli stati che si ottengono applicando ad $s \in \Sigma$ differenti sequenze di riduzioni, che trasformino s in un altro grafo non ulteriormente riducibile, sono fra di loro equivalenti.*

Dim. - Il secondo punto di questo Teorema segue dall'osservazione che una riduzione non può mai decrementare il numero di istanze di risorse disponibili e che la riduzione effettuata da un dato processo P_j cancellerà gli stessi archi indipendentemente dalle riduzioni fatte in precedenza. Poichè il *Corollario 1* afferma che la completa riducibilità è una condizione sufficiente affinchè uno stato non rappresenti uno stato di stallo, si deve solo far vedere che, in questo caso, la completa riducibilità è anche una condizione necessaria. La prova richiesta segue dal fatto che quando il grafo non è completamente riducibile, i processi bloccati che non possono essere ridotti sono in stallo.

Non è difficile, a questo punto, rendersi conto della validità del seguente

Corollario 3.- Se ciascuna risorsa riusabile è costituita esattamente da un'unica istanza, allora la **non completa riducibilità** del grafo e l'**esistenza di un ciclo** nel grafo sono condizioni equivalenti per lo stallo.

La dimostrazione del secondo punto del *Teorema 5* implica che esiste un algoritmo per il riconoscimento dello stallo *ragionevolmente veloce* nel caso di sistemi di risorse riusabili. L'algoritmo riduce

[25] visto come stato del sistema

[26] cosicchè v rappresenta uno stato di stallo

[27] espressa come completa riducibilità del corrispondente grafo di richieste limitate

successivamente il grafo per quanto è possibile; il grafo originale non rappresenta uno stato di stallo se e solo se queste riduzioni cancellano tutti gli archi del grafo. Poichè qualunque sequenza di riduzioni dà luogo allo stesso grafo finale, l'algoritmo non necessita di *backtracking*. Tale algoritmo, comunque, è rallentato dal fatto che durante ciascuna riduzione è necessario eseguire una ricerca per determinare quale processo[28] deve essere ridotto al passo successivo.

Queste considerazioni ci permettono di definire opportunamente il seguente Algoritmo 3.- Per verificare se il grafo che rappresenta lo stato di un sistema di risorse riusabili è completamente riducibile, si costruisce, innanzittutto, la lista dei processi del grafo che non sono in attesa ma hanno istanze di risorse allocate. I processi che non sono in attesa e non hanno unità di risorse allocate, sono considerati già ridotti. Per evitare la ricerca sopra menzionata, lo stato del sistema viene rappresentato utilizzando

- un contatore di attese che dà, per ciascun processo, il numero di risorse le cui unità disponibili sono minori di quelle richieste dal processo;
- una lista ordinata delle richieste che fornisce, per ciascuna risorsa, la lista dei processi che richiedono quella risorsa, ordinata per valori crescenti di istanze della risorsa.

L'algoritmo riduce successivamente i processi che hanno nullo il contatore delle attese ma non nullo il numero delle istanze di risorse allocate. Ciascuna risorsa incrementa il numero di unit'a disponibili di almeno una unità che, a sua volta, può decrementare in contatori di attesa per altri processi che stanno richiedendo istanze di risorse. Si noti che non è necessario eseguire alcuna ricerca per individuare il processo P_j dal momento che P_j è automaticamente determinato come il successivo processo nella lista ordinata dei processi che richiedono istanze della risorsa R_k.

Si tenga presente che il numero totale di esecuzioni relativamente ai processi le cui richieste possono essere garantite è limitato dal numero m di liste ordinate[29] di richieste moltiplicato per il numero n di processi in ciascuna lista. Pertanto, il massimo tempo di esecuzione dell'*Algoritmo 3* è proporzionale a $m \times n$ perchè la lista dei processi, che devono essere ridotti, può contenere al più tutti i processi del grafo[30] mentre le operazioni da effettuarsi su ciascuna risorsa assegnata o rilasciata, relativamente a ciascun processo della lista, sono al più in numero di $m \times n$[31].

Se l'*Algoritmo 3* non facesse uso di contatori d'attesa e liste di richieste ordinate, allora il tempo di esecuzione sarebbe proporzionale a $m \times n^2$. Sebbene possa apparire oneroso mantenere i contatori d'attesa e le liste ordinate di richieste durante le operazioni del sistema. il vantaggio significativo che si ottiene, dopo ogni rilascio di risorsa, è la completa assenza di una fase di ricerca per determinare quale processo è in grado di vedere soddisfatte le proprie richieste. Tali processi sono proprio quelli i cui contatori di attesa sono nulli in seguito al rilascio di qualche risorsa.

Si può verificare che in un sistema di risorse riusabili solamente le richieste di unità non disponibili possono dar luogo allo stato di stallo. Perciò, per realizzare la verifica *continua* dello stallo, è sufficiente applicare l'*Algoritmo 3* solamente in presenza di richieste per unità non disponibili. Tali richieste daranno luogo allo stallo solo se il processo richiedente è entrato in uno stato di stallo. Pertanto, l'*Algoritmo 3* può venire fermato[32] non appena il contatore d'attesa del processo richiedente assume il valore zero.

In generale, comunque, i processi che interagiscono solamente attraverso un sistema di risorse riusabili, si presentano come *job* distinti di utenti i cui programmi non interagiscono fra di loro,

[28] se esiste
[29] una per ogni risorsa
[30] in numero di n
[31] una volta per ciascun arco assegnazione/richiesta pesato
[32] con la conclusione che lo stallo non si è verificato

sottomessi all'elaborazione in modo *batch*. In sistemi di questo tipo, l'impiego dell'*Algoritmo 3* può essere utile nella verifica di stati di stallo e, in caso affermativo, effettuare alcune operazioni per la sua rimozione come la *terminazione forzata dei job* oppure l'*acquisizione incondizionata delle risorse* allocate ai job, da parte del sistema.

6.11 PREVENZIONE DELLO STALLO

Si può verificare facilmente che, nel caso di un sistema di risorse riusabili, lo stato di stallo è addirittura *prevenibile* qualora si imponga un limite massimo alle richieste di risorse per ciascun processo. Tale metodo di prevenzione è basato sull'*Algoritmo 3*, del quale ne costituisce un miglioramento, ed è noto in letteratura col nome di **Algoritmo del Banchiere**.

Definiamo, innanzittutto, la *matrice delle richieste* C come una matrice bidimensionale di ordine $n \times m$ il cui generico elemento C_{ij} fornisce il massimo numero di unità della risorsa R_j che il processo P_i richiederà durante la sua esecuzione. Naturalmente, deve risultare $0 \leq C_{ij} \leq w_j$ per ciascun processo P_i. Inoltre, per ciascun processo P_i, si richiede che si abbia $C_{ij} < 0$ per almeno una risorsa R_j; questo significa che ogni processo può richiedere almeno una istanza di qualche risorsa. Si può, allora, dare la seguente definizione.

Si dice **sistema di risorse riusabili con richieste limitate** un sistema di allocazione di risorse riusabili dal quale sono state eliminate tutte quelle richieste che, per ciascun processo P_i, fanno si che, in corrispondenza di qualche risorsa R_j, il numero di archi richiesta (P_i, R_j) sommato con il numero di archi assegnazione (R_j, P_i) non superi mai il valore C_{ij}.

La precedente equivale ad assumere che i processi non richiederanno mai un numero di istanze di ciascuna risorsa che non è già stato preventivato.

Per ogni stato di un sistema di questo tipo, la costruzione del *grafo di richieste limitate* è ottenuta dal grafo originale aggiungendo archi richiesta del tipo (P_i, R_j)[33] fino a quando il loro numero, sommato con il numero degli archi assegnazione (R_j, P_i) già presenti, non è uguale a C_{ij}. Intuitivamente, il grafo di richieste limitate[34] viene generato in modo da permettere che le richieste di tutti i processi possano essere soddisfatte almeno in numero di unità pari alle loro *richieste preventive*.

Per prevenire lo stallo, si deve garantire che, anche se tutti i processi richiedono tutte le istanze di risorse permesse dalle loro *richieste preventive*, lo stallo non si potrà mai verifichare in nessuno modo. Ci si potrebbe aspettare, dunque, che per evitare lo stallo è sufficiente che il sistema non possa mai giungere in uno stato il cui grafo di richieste limitate rappresenti uno stato di stallo. Questo è, in effetti, ciò che afferma il *Teorema 6*. L'algoritmo che si ottiene[35] fà uso della capacità del sistema operativo di determinare quando garantire quali richieste. In termini del presente modello formale, questo significa che qualcuno degli stati di transizione definiti da *operazioni di acquisizione* possono essere eliminati dal sistema operativo. Si può, pertanto, dare le seguenti definizioni.

- Si definisce **politica di acquisizione** qualunque regola che elimina alcune operazioni di acquisizione del sistema, generando un nuovo sistema che è *sicuro*.
- Si dice che una politica di acquisizione è **ottima** se non esiste nessun'altra politica di acquisizione che elimina un numero minore di acquisizioni.
- Infine, si dice che un'operazione di acquisizione è **salva** se dà luogo ad uno stato il cui grafo di richieste limitate è completamente riducibile.

[33] per ogni i e j
[34] che rappresenta un certo stato del sistema
[35] dovuto ad Habermann come variante dell'*Algoritmo 3*

Si può, allora, enunciare il seguente

Teorema 6.- *La politica di acquisizione ottima per un sistema di risorse riusabili a richieste limitate è quella che elimina tutte le acquisizioni che non sono salve.*

Dim.- Per la dimostrazione bisogna verificare che se la politica di acquisizione ne permette una che non è salva, allora necessariamente esiste una sequenza di operazioni che conduce in uno stato di stallo. Inoltre, se si permettono soltanto acquisizioni salve, allora qualunque stato il cui grafo di richieste limitate è completamente riducibile, non è uno stato di stallo.

Questo *Teorema* sta a significare che lo stallo può essere prevenuto[36] rifiutando tutte quelle richieste che, se soddisfatte, portano il sistema in uno stato che potrebbe condurre in uno stato di stallo. Per determinare se uno stato *può condurre in uno stato di stallo* basta verificare che il corrispondente grafo di richieste limitate sia completamente riducibile. Tale verifica può effettuarsi efficientemente utilizzando l'*Algoritmo 3* il quale si presta ugualmente bene sia per prevenire che per rivelare lo stallo.

[36]garantendo il maggior numero di richieste possibili